ON SOLVING LARGE-SCALE SYSTEM OF NONLINEAR EQUATIONS VIA DOUBLE DIRECTION AND STEP LENGTH APPROACH

BY

ABUBAKAR SANI HALILU
SPS/13/MMT/00010

BAYERO UNIVERSITY, KANO.

ON SOLVING LARGE-SCALE SYSTEM OF NONLINEAR EQUATIONS VIA DOUBLE DIRECTION AND STEP LENGTH APPROACH

By

ABUBAKAR SANI HALILU
SPS/13/MMT/00010

B.Sc. (Hons.), (B.U.K)

A Dissertation Submitted to the Department of Mathematical Sciences, Bayero University, Kano, in Partial fulfillment of the requirements for the award of the degree of
MASTERS IN MATHEMATICS.

DECEMBER, 2016

Declaration

I hereby declare that this work is the product of my research efforts undertaken under the supervision of Muhammad Waziri Yusuf (Ph.D) and has not been presented anywhere for the award of a degree or certificate. All sources have been duly acknowledged.

ABUBAKAR SANI HALILU	———————————-
(Name)	(Signature)
SPS/13/MMT/00010	———————————-
(Reg. Number)	(Date)

Certification

This is to certify that the research work for this dissertation and the subsequent write-up were carried out by Abubakar Sani Halilu (SPS/13/MMT/00010) under my supervision.

Dr. Muhammad Waziri Yusuf	____________________	____________________
(Supervisor)	Signature	Date
Dr. Bashir Ali	____________________	____________________
(Head of Department)	Signature	Date

Approval

This dissertation has been examined and approved for the award of the degree of MASTERS OF SCIENCE IN MATHEMATICS.

Dr. Muhammad Yusuf Waziri	——————————	——————————
(Supervisor)	Signature	Date
Dr. Bashir Ali	——————————	——————————
(Head of Department)	Signature	Date
Dr. Sirajo Lawan Bichi	——————————-	——————————-
(Internal Examiner)	Signature	Date
Dr. Hamisu Musa	——————————-	——————————-
(External Examiner)	Signature	Date
Dr. Nafiu Hussaini	——————————-	——————————-
(S P S Representative)	Signature	Date

Acknowledgments

In the name of Allah, the Most Beneficent, the Most Merciful. All Praises are to Almighty Allah, the creator and Sustainer of the Universe. Peace be upon his messenger Muhammad (S A W). First of all, my deep appreciation goes to my supervisor in person of Muhammad Waziri Yusuf (Ph.D) for his guidance and assistance in conducting the needed work of this dissertation. I personally appreciate the positive contribution of my head of department Bashir Ali (Ph.D). The generosity and positive contribution of the entire staff of the department is highly appreciable. I specially acknowledged the courage and motivation given to me by Ibrahim Yusuf Fagge (Ph.D). I am also indebted to Ma'aruf Shehu Minjibir (Ph.D) for his support and cooperation. I wish to express my appreciation to Dr. Usman Bello Abdulmalik from Federal University Dutsin-ma and Auwalu Sulaiman (Overall) who have been so helpful.
Notwithstanding, i will also like to express my gratitude to Sule Lamido University, Kafin-Hausa, Jigawa state, for giving me the opportunity to pursue this study. My unfathomable credit goes to my colleagues Jamilu Sabiu, Lawan Muhammad Gimi, Ya'u Balarabe Musa and Muhammad Kabir Dauda (MKD) for their effort, feedback and motivation in writing this dissertation. My special words of thanks go to my class mates, Nura Garba Khalil, Muhammad Jada Aliyu, Habibu Abdullahi, Idris Ahmad, Alhassan Ibrahim Waziri, Yusuf Ibrahim, Mustapha Umar Adamu, Hajara Shu'aibu, and Rahma Gatawa.
Nevertheless, my profound gratitude goes to my beloved wife Saddiqa Abbas Mu'azu for her understating, prayer and patience throughout my studies. I will also like to express my appreciation to my biological son Muhammad Bashir (Halifa). I am also indebted to my parents and family for their support, encouragement and understanding during the period of my studies.

Dedication

This work, is humbly dedicated to my parents.

Contents

List of Tables

List of Figures

Abbreviation

NI	Number of iteration.
CPU	Central Processing Unit.
CG	Conjugate Gradient.
PRP	polak-Rebière-Polyak.
HS	Hestenes-Stiefel.
FR	Fletcher-Reeves.
D	Daniel.
LS	Liu-Storey.
DY	Dai-Yuan.
BFGS	Broyden-Fletcher-Goldfarb-Shanno.

Abstract

In this work, we present a double direction with single step length, a single direction with double step length and a transformed double step length methods for solving large-scale system of nonlinear equations. These methods use the special form of iteration with two directions and two step lengths. The approximation to the Jacobian matrix is done by sufficiently constructed diagonal matrix via acceleration parameter, as well as inexact line search procedure. The proposed methods are proved to be globally convergent under mild conditions. Finally, numerical comparison using a large scale benchmark test problems show that the proposed methods are very promising.

CHAPTER ONE

INTRODUCTION

1.1 INTRODUCTION

This chapter shade more light on derivative-free double direction and step length methods for solving system of nonlinear equations. It contains among other things, motivation of the study, the aim and objectives, scope and limitations, methodology and some basic definitions.

1.2 NONLINEAR SYSTEM OF EQUATIONS

The general form of a system of nonlinear equations is

$$F(x) = 0, \tag{1.2.1}$$

where $F : \mathbb{R}^n \to \mathbb{R}^n$, x $= (x_1, x_2, ..., x_n)^T$ and $F = (f_1, f_2, ...f_n)^T$. Each function $f_i(i = 1, 2, ..., n)$ can be thought of as mapping a vector x $= (x_1, x_2, ..., x_n)^T$ of the n-dimensional space $\mathbb{R}^n$ into the real line $\mathbb{R}$ and is assumed to satisfy the following:

- There exists an $x^* \in R^n$ s.t. $F(x^*) = 0$.

- F is a continuously differentiable mapping in a neighborhood of x^*.

One of the renowned methods for finding the solution to (1.2.1) is the Newton's method. The method is simple to implement, and generates an iterative sequences $\{x_k\}$ from a given initial guess x_0 in a neighborhood of x^* (Kelley, 2000) via

$$x_{k+1} = x_k - \left(F'(x_k)\right)^{-1} F(x_k), \tag{1.2.2}$$

where $k = 0, 1, 2, ...$ and $F'(x_k)$ is the Jacobian matrix of F at x_k.
Some of the attractive features of Newton's method are easy implementation and rapid convergence (Kelley, 2000). However , it requires the computation of Jacobian matrix, which demands the first-order derivative of the system. The computation of the derivative of some functions are costly in practice, sometimes they are not even available or could not be obtained exactly. In this case Newton's method cannot be applied directly (Waziri, Leong, & Hassan, 2011). To overcome such difficulty, the simple modification on the Newton method is the fixed Newton method (Natasa & Zorna, 2001) for the determination of solution to (1.2.1) is given by

$$x_{k+1} = x_k - \left(F'(x_0)\right)^{-1} F(x_k), \tag{1.2.3}$$

where $k = 0, 1, 2, ...$ The method avoids the computation and storing the Jacobian in each iteration (except at k=0), but it still requires solving the system of n linear equations which consumes more CPU time as the system's dimension increases (Natasa & Zorna, 2001).
A quasi-Newton's method is another variant of Newton-type methods and replace the Jacobian or its inverse with an approximation which can be updated at each iteration (Yuan* & Lu, 2008), and its updating scheme is given by

$$x_{k+1} = x_k - B_k^{-1} F(x_k), \tag{1.2.4}$$

where B_k is the approximation of Jacobian at x_k. The main idea behind quasi-Newton method is to eliminate the evaluation cost of the Jacobian matrix (Broyden, 1965).

Due to the well known shortcomings of Newton method, a double direction and step length method has been proposed by Petrovic (2015) and the iterative procedure is given as

$$x_{k+1} = x_k + \alpha_k b_k + \beta_k c_k, \tag{1.2.5}$$

where x_{k+1} represents a new iterative point, x_k is the previous iterative point, α_k and β_k

denote the step lengths,while b_k and c_k are search directions respectively.

1.3 MOTIVATION OF THE STUDY

It is very important to state that the double direction and step length methods was used in unconstrained optimization problem. They are particularly efficient due to their convergence properties, simple implementation and low storage requirement (Stanimirovic, Milovanovic, Petrovic, & Kontrec, 2015). Nevertheless, the study of double direction and step length methods for solving system of nonlinear equations is very scanty, so for this reason we are motivated to incorporate this idea to system of nonlinear equations, in order to have derivative-free and matrix-free methods and to enhance convergence properties of the proposed methods. Moreover, most iterative methods for handling system of nonlinear equations are one direction-type. We felt that to have two directions at a time independently will improve the convergence of any given method at a point. This is what motivated us to have this work.

1.4 STATEMENT OF THE PROBLEM

Most of the single direction/step length methods for solving system of nonlinear equations require Jacobian matrix computation in each iteration (Kelley, 2000). Also some single direction/step length methods require solving n-linear equations per iteration. Furthermore, the convergence of some single direction or single step length is very poor due to less information at a point or singularity of the matrix at a given point. Therefore, in this research, we proposed new and enhanced acceleration parameter γ_k, so that $\gamma_k I$ approximate the Jacobian matrix. So our methods are derivative-free, singularity-free and less computational cost compared to some existing methods.

1.5 AIM AND OBJECTIVES

The aim of this work is:

To present some double direction/step length approach for solving system of nonlinear equations.

While the objectives of this research are:

1. To present matrix free methods for handling system of nonlinear equations via acceleration parameter.
2. To develop a numerical algorithm for solving the methods derived.
3. To provide the convergence analysis of the proposed methods.

1.6 SCOPE AND LIMITATIONS

This research is only limited to nonlinear system of equations consisting of n-equations and n-variables.

1.7 DEFINITIONS OF SOME BASIC TERMS

In order to guide us through the understanding of this work, below are the definitions of some basic terms;

Definition 1.7.1 (Jacobian Matrix) *Let f_i be the ith component of $F : \mathbb{R}^n \to \mathbb{R}^n$. If the components of F are differentiable at $x \in \mathbb{R}^n$, then the Jacobian matrix $F'(x)_{ij}$ can be define by*

$$F'(x)_{ij} = \frac{\partial f_i}{\partial x_j}(x).$$

Definition 1.7.2 (Norm) *A Norm for an arbitrary finite-dimensional vector $x \in \mathbb{R}^n$, denoted by $\|x\|$ is a real-valued function satisfying the following four conditions for all vectors x and y of the same dimension:*

(N1): $\|x\| \geq 0$.

(N2): $\|x\| = 0$ *if and only if* $x = 0$.

(N3): $\|cx\| = |c|\|x\|$ *for any scalar c.*

(N4): $\|x+y\| \leq \|x\| + \|y\|$.

Definition 1.7.3 (Convex Set) *Let X be a normed linear space, a subset C of X is said to be* ***convex*** *if for each $x, y \in C$ and $\lambda \in [0,1]$*

$$\lambda x + (1-\lambda)y \in C.$$

Definition 1.7.4 (a continuously differentiable function) *A function that has derivative that itself is a continuous function is said to be a continuously differentiable function, i.e*
$F : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ is continuously differentiable if F' exists and is continuous.

Definition 1.7.5 (Lipschitz Continuity) *Let $C \subset \mathbb{R}^n$ and let $F : C \to \mathbb{R}^n$. F is Lipschitz continuous on C with Lipschitz constant γ if*

$$\|F(x) - F(y)\| \leq \gamma \|x - y\|$$

for all $x, y \in C$.

Definition 1.7.6 (Convergence) *A sequence $\{x_k\}$ of n-vectors converges to x^* if*

$$\lim_{k \to \infty} \|x_k - x^*\| = 0.$$

Definition 1.7.7 (q-quadratically convergence) *Let $\{x_k\} \subset \mathbb{R}^n$ and $x^* \in \mathbb{R}^n$. Then, $x_k \to x^*$ q-quadratically if $x_k \to x^*$ and there exists $\lambda > 0$ such that $\|x_{k+1} - x^*\| \leq \lambda \|x_k - x^*\|^2$.*

Definition 1.7.8 (q-superlinearly convergence) *Let* $\{x_k\} \subset \mathbb{R}^n$ *and* $x^* \in \mathbb{R}^n$*. Then,* $x_k \to x^*$ *q-superlineally if* $\lim\limits_{k\to\infty} \dfrac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} = 0.$

Definition 1.7.9 (Linear Convergence) *Let* $\{x_k\} \subset \mathbb{R}^n$ *and* $x^* \in \mathbb{R}^n$*. Then* $x_k \to x^*$ linearly *if* $x_k \to x^*$ *and there exist* λ *in* $(0,1)$ *such that*

$$\|x_{k+1} - x^*\| \leq \lambda \|x_k - x^*\|.$$

Definition 1.7.10 (A merit function) *This is a square norm function of the form* $f(x) = \|F(x)\|^2$ *, where,* $F : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ *and* $\|\cdot\|$ *is Euclidean norm.*

Definition 1.7.11 (l_1 Norm, Euclidean Norm and l_∞ Norm) *For a vector* $x \in \mathbb{R}^n$*,*

- *The* l_1 *norm is defined as* $\|x\| = \sum\limits_{i=1}^{n} (|x_i|)$.
- *The Euclidean norm (or* l_2 *norm) is defined as* $\|x\|_2 = \left(\sum\limits_{i=1}^{n} (x_i^2) \right)^{\frac{1}{2}}$.
- *The* l_∞ *norm is defined as* $\|x\| = \max_{1\leq i\leq n} (|x_i|)$.

Definition 1.7.12 (Induced Matrix Norm) *Let* $\|.\|$ *be the norm in* $\mathbb{R}^n$*. The induced matrix norm of* $n \times n$ *matrix* A *is defined by*

$$\|A\| = \max_{\|x\|=1} \|Ax\|.$$

Definition 1.7.13 (A nonmonotone line search technique) *Nonmonotone approach is that type of line search that causes the maximum of recent function values to decrease.*

Definition 1.7.14 (Quasi-Newton Condition)

$$B_{k+1}s_k = y_k,$$

where $s_k = x_{k+1} - x_k$*,* $y_k = F(x_{k+1}) - F(x_k)$ *and* B_k *is the approximate Jacobian. For* $k = 0, 1, 2, ...$

CHAPTER TWO

LITERATURE REVIEW

In this chapter, we present reviews of the related literatures more especially those related to Newton, Quasi-Newton, double direction, double step length methods e.t .c.

2.1 A BRIEF HISTORY OF NEWTON'S METHOD

The historical road to Newton's method is interesting in its own right, setting aside the technical details. Sir Isaac Newton did indeed play a role in its development, but there were certainly other key players. Despite the uncertainties, most experts agree that the road begins with a perturbation technique for solving scalar polynomial equations, pioneered by Francois Vieta in 1600 (Deuhard, 2012). The method explained by Deuhard (2012) used one decimal place of the computed solution on each step, and converged linearly (Deuhard, 2012). In 1664 Isaac Newton learned of Vieta's method, and by 1669 he had improved it by linearizing the resulting equations (Deuhard, 2012). One of his fist applications of this newfangled scheme was on the numerical solution of the cubic polynomial equation below

$$f(x) = x^3 - 2x - 5 = 0. \tag{2.1.1}$$

By using an initial guess x_0 of 2, he created an iterative method by substituting x_k= $x_{k-1} + \delta$ into the above equation and solving for δ (neglecting terms higher than first order), and repeating, etc, which produced a better and better approximation to the true root (Deuhard, 2012). Newton realized that by keeping all decimal places of the corrections, the number of accurate places would double, which means quadratic convergence. However, there is no evidence that Newton incorporated derivatives in his method. In 1690, Joseph Raphson improved Newton's method of 1669 by making the scheme fully iterative, making computation of successive polynomials unnecessary (Deuhard, 2012). But it was in 1740 when the derivative aspect of ”Newton's Method” was brought to the fore by Thomas Simpson (Deuhard, 2012). Simpson extended Newton's formulation of 1669 by introducing derivatives, and wrote the mathematical formula for the iterative method for both a scalar nonlinear equation, and a system of two equations and two unknowns (Deuhard, 2012). For a scalar equation and some initial guess x_0, this formulation was given by the equation below

$$x_k = x_{k-1} - \frac{f(x_{k-1})}{f'(x_{k-1})}, \quad for \quad k = 1,2,... \tag{2.1.2}$$

Thus, it was Simpson who made the leap from using this procedure on scalar equations to systems of equations.

2.1.1 Secant Updating

Variations of what is now known as the secant method date back to the Babylonian era some 4000 years ago, for there is evidence that they used the secant updating formula to solve equations in one dimension (Griewank, 2012). This requires two initial guesses x_0 and x_1 and computes the updated solution as in equation below.

$$x_{k+1} = x_k - \frac{f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})}, \quad for \quad k = 1,2,... \tag{2.1.3}$$

In that time, the conception of differential calculus was almost certainly unknown. Today, derivatives are understood very well, but analytical evaluation of derivatives can be exorbitant if the underlying functions themselves are complicated. This was in fact a problem for Charles Broyden in the 1960's, when he was working with nonlinear reactor models for the English Electric Company in Leicester (Griewank, 2012). When trying to solve nonlinear systems that arose from the discretization of these models, he realized that instead of repeatedly evaluating and factoring the Jacobian, he could use secant information (function value differences and the previous solution information) to directly compute an approximation to the Jacobian (Griewank, 2012). In the one-dimensional case, the problem is trivial because a unique approximation, B_k, can be computed by dividing the function difference by the solution difference. Thus, letting $\triangle_k = F_k - F_{k-1}$ and $\delta_k = x_k - x_{k-1}$,

$$B_k = \frac{\triangle_k}{\delta_k} \approx F'(x_k). \tag{2.1.4}$$

However, for multi-dimensional systems, the secant condition provides n conditions for the n^2 degrees of freedom in the new Jacobian approximation (Kelley, 2000). Broyden's stroke of genius was realizing that the Jacobian approximation from the previous iteration, B_{k-1}, could be updated with a certain rank-one update. This update would satisfy not only the secant condition, but also the condition that $B_k w = B_{k-1} w$ for all directions $w \in \mathbb{R}^n$ orthogonal to δ_k (i.e. $w_k^T \delta_k = 0$) (Griewank, 2012). This idea led Broyden to develop three methods that use secant information in lieu of derivative information to solve nonlinear systems. Broyden published these results in his paper (Broyden, 1965).

2.1.2 Newton's Algorithm

Newton method is the most widely used algorithm for solving nonlinear system of equations (Kelley, 2000). The Newton direction, d_k is defined at each iteration by:

$$d_k = -\left(F'(x_k)\right)^{-1} F(x_k), \tag{2.1.5}$$

where $k = 0, 1, 2, \ldots$ and $F'(x_k)$ is the Jacobian matrix of F at x_k.
However, in order to generate an algorithm, d_k is used as search direction and the Newtonian iteration takes the form

$$x_{k+1} = x_k + \alpha_k d_k, \tag{2.1.6}$$

where α_k is the step length.

The algorithm of the Newton's method is given below:
Algorithm 1

```
STEP 1: Given $x_0$, stopping criterion $\varepsilon$,s et $k = 0$.
STEP 2: Compute $F(x_k)$ and check if $\|F(x_k)\| \leq \varepsilon$, then stop, else go to STEP 3.
STEP 3: Compute search direction $d_k = -(F'(x_k))^{-1}F(x_k)$.
STEP 4: Compute a step length $\alpha_k$.
STEP 4: Set $x_{k+1} = x_k + \alpha_k d_k$.
```

Where, $\alpha_k d_k$ is called the Newton correction (Krejic & Luzanin, 2001 ; Waziri, Leong, Hassan & Monsi, 2010). The Newton method possesses good theoretical characteristics such as quadratic convergence for any sufficiently good initial guess say x_0. Despite its qualities, this method has a number of disadvantages in practice as given below:

- The cost of computation and storage of the Jacobian matrix.
- Solving of n-linear equations and moreover, in some systems, analytic derivative could not be done precisely.

System of equations arise so frequently in not just pure mathematics, but also in the physical science and engineering disciplines. In fact, nonlinear systems abound in mathematical problems. This makes for the phenomena being modeled more diverse and interesting, but the price to pay is that finding solutions is non-trivial. Then how are systems of nonlinear equations ultimately solved, and do the solvers always succeed?
The two main class of methods used for solving nonlinear system of equations are Newton and quasi-Newton. Equivalently, it could be said that the two classes of methods are derivative and derivative-free, respectively.

2.2 QUASI-NEWTON METHOD

A quasi-Newton's method is another variant of Newton-type methods and replace the Jacobian or its inverse with an approximation which can be updated at each iteration (Griewank, 1986 ; Mascarenhas, 2004 ; Powell, 2000). This was done as a result of the above-mentioned shortcomings of the Newton method.

The Quasi-Newton methods have general equation given by

$$x_{k+1} = x_k - B_k^{-1} F(x_k), \tag{2.2.1}$$

where B_k is the approximation of Jacobian at x_k. The main idea behind quasi-Newton method is to eliminate the evaluation cost of the Jacobian matrix (Broyden, 1965).
With each successive iteration, the goal is to force B_k to become closer to the true Jacobian, and thus also force x_k to become closer to x^*. The class of methods written in the form of equation (2.2.1) are called quasi-Newton methods, and in general, their convergence rate is q-superlinearly. Interestingly, most of the known quasi-Newton methods do not require consistency (i.e. that the sequence $\{B_k\}$ converge to the Jacobian matrix at the root) for convergence, as posited by Dennis and More (1974).
The general scheme for the quasi Newton's method is given by:

$$\begin{cases} d_k = -B_k^{-1} F(x_k) \\ x_{k+1} = x_k + \alpha_k d_k. \end{cases} \tag{2.2.2}$$

.

There are many types of quasi-Newton methods, but the most efficient and successful Quasi-Newton method is Broyden, Fletcher,Goldfarb, Shanno method , denoted as BFGS (Yuan* & Lu, 2008), where the BFGS update is defined as:

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T S_k}, \tag{2.2.3}$$

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

2.3 DOUBLE DIRECTION METHOD

Double direction method has been proposed by Dbaruranovic (2008), using multi-step iterative information and curve search to generate new iterative points. Recently, Petrovic and Stanimirovic (2014) presented a double direction to solve unconstrained optimization problem. The double direction iterative scheme is given as:

$$x_{k+1} = x_k + \alpha_k b_k + \alpha_k^2 c_k, \tag{2.3.1}$$

where x_{k+1} represents a new iterative point, x_k is the previous iteration, and α_k denotes the step length, while b_k and c_k are search directions.
However, a double step length method has been proposed by Petrovic (2015) and the iterative procedure is given as:

$$x_{k+1} = x_k + \alpha_k g_k + \beta_k h_k, \tag{2.3.2}$$

where x_{k+1} represents a new iterative point, x_k is the previous iterative point, α_k and β_k denote the step lengths, while g_k and h_k are search directions.
It is very important to state that the derivative-free double direction and step length methods are severally used in unconstrained optimization problems. They are particularly efficient due to their convergence properties, simple implementation, and low storage requirement (Petrovic & Stanimirovic, 2014 ; Petrovic, 2015) though their variant for solving nonlinear system of equations are rare. It is vital to mention that there are several procedure for the choice of the search direction (Li & Fukushima, 1999 ; Zang, Zhou, & Li, 2005).
In steepest descent method the direction d_k is defined as:

$$d_k = -F(x_k) \tag{2.3.3}$$

(Petrovic, 2015). The conjugate gradient direction for solving system of nonlinear equations has received a good attention and take an appropriate progress, where the direction d_k is defined as:

$$d_k = \begin{cases} -F(x_k) & if \quad k = 0 \\ -F(x_k) + \beta_k d_{k-1} & if \quad k \geq 1, \end{cases} \tag{2.3.4}$$

where β_k is a CG-parameter. A crucial element in any conjugate gradient algorithm is the formulation of β_k. Below are some popular choices for the conjugate gradient update parameters;

- $\beta_k^{HS} = \frac{g_{k+1}^T y_k}{d^T y_k}$ (Hestenes & Stiefel, 1952).
- $\beta_k^{FR} = \frac{||g_{k+1}||^2}{||g_k||^2}$ (Fletcher & Reeves, 1964)
- $\beta_k^{D} = \frac{g_{k+1}^T \nabla^2 f(x_k) d_k}{d^T \nabla^2 f(x_k) d_k}$ (Daniel, 1967). requires evaluation of the Hessian $\nabla^2 f(x)$.
- $\beta_k^{LS} = \frac{g_{k+1}^T y_k}{-d^T g_k}$ (Liu & Storey, 1991).
- $\beta_k^{DY} = \frac{||g_{k+1}||^2}{d^T y_k}$ (Dai & Yuan, 1999).

2.4 LINE SEARCH

A line search is a technique that computes the step length α_k along the direction d_k. The choice of the step length affect both the convergence and speed of convergence of the algorithm.
The step length α_k can also be computed either exact or in exact. The exact line search is based on well-known formula (Mascarenhas, 2004).

$$\alpha_k = \arg\min_{\alpha>0} f(x_k + \alpha_k d_k). \tag{2.4.1}$$

Theoretically, exact optimal step length is difficult to be found in practical computation, and it is also expensive to find almost exact step length. Therefore the most frequently used line search in practice is inexact line search (Raydan, 1993), which sufficiently decrease the function values i.e $||F(x_{k+1})|| \leq ||F(x_k)||$.

2.4.1 Backtracking line search

In most of the nonlinear problems, the search direction, d_k, must be controlled with some parameter α_k, called the step length. For this, a derivative-free line search must be employed. The purpose of a line search is to modify a hitherto locally convergent scheme and make it globally convergent (Waziri & Sabiu, 2015). An effective line search controls the value of α_k in order to ensure that the root solver is pushing $F(x)$ "downhill". The line search employed here is a derivative-free line search (Li & Fukushima, 1999), with the condition given by

$$f(x_k + \alpha_k d_k) - f(x_k) \leq -\omega_1 \|\alpha_k F(x_k)\|^2 - \omega_2 \|\alpha_k d_k\|^2 + \eta_k f(x_k),$$

where $f : \mathbb{R}^n \to \mathbb{R}$, $\omega_1 > 0$, $\omega_2 > 0$ be constants and $\{\eta_k\}$ be a given positive sequence for some $\eta > 0$ such that

$$\sum_{k=0}^{\infty} \eta_k < \eta < \infty.$$

The procedure for enforcing the above-mentioned line search is to use a backtracking process, where the algorithm checks if the condition is satisfied. If not, then α_k is reduced to some α_{k+1}, and so on and so forth, until the condition is satisfied. Since there is no guarantee that the condition will be satisfied, this loop will only run for a specified number of iterations before it terminates and reports a line search failure.

2.5 WHY DOUBLE DIRECTION AND STEP LENGTH?

Most of the methods for solving system of nonlinear equations are single direction methods. One of the drawback of these methods is that, they have only one correction factor, so the system will automatically collapse when the correction fails. But in double direction methods there are two corrections (Petrovic & Stanimirovic, 2014), so if one correction fails the other correction will correct the system.

In double step length methods there are two step length (Petrovic, 2015) that can be determined concurrently so that, if one step length fails during the evaluation of line search the other will help to decrease the function value.

CHAPTER THREE

METHODOLOGY

3.1 INTRODUCTION

In this chapter, we present our three proposed methods for solving system of nonlinear equations. The methods are based on approximation of the Jacobian in Newton method via acceleration parameter $\gamma_k > 0$ i.e

$$F'(x_k) \approx \gamma_k I, \tag{3.1.1}$$

where I is an identity matrix.
Furthermore (1.2.1) can come from an unconstrained optimization problem, a saddle point, and equality constrained problem (Li & Fukushima, 1999). Let f be a norm function defined by

$$f(x) = \frac{1}{2}\|F(x)\|^2, \tag{3.1.2}$$

where $f : \mathbb{R}^n \to \mathbb{R}$.
The nonlinear problem (1.2.1) is equivalent to the following global optimization problem

$$min f(x), \quad x \in \mathbb{R}^n. \tag{3.1.3}$$

In the first method, we propose the double direction technique for solving system of nonlinear equations. In the second method, we introduce a double step length method for solving system of nonlinear equations. We finally transformed a double step length into a single step length in the last method.

3.2 AN IMPROVED DERIVATIVE-FREE METHOD VIA DOUBLE DIRECTION APPROACH (IDFDD)

In this method, we propose the two directions vectors for solving derivative-free method via acceleration parameter. In order to incorporate more information of the iterates at each iteration and to improve good direction towards the solution, we suggest new directions b_k and c_k in (2.3.1) to be defined as:

$$b_k = -\gamma_k^{-1} F(x_k), \tag{3.2.1}$$

and

$$c_k = -F(x_k), \tag{3.2.2}$$

where $\gamma_k > 0$ is an acceleration parameter.

By putting (3.2.1) and (3.2.2) into (2.3.1) we obtained

$$x_{k+1} = x_k - (\alpha_k + \alpha_k^2 \gamma_k)\gamma_k^{-1} F(x_k), \tag{3.2.3}$$

from (3.2.3) we can easily see that,our direction is :

$$d_k = -\gamma_k^{-1} F(x_k), \tag{3.2.4}$$

then using (3.2.3) and (3.2.4) we have the general scheme as:

$$x_{k+1} = x_k + (\alpha_k + \alpha_k^2 \gamma_k) d_k. \tag{3.2.5}$$

We proceed to obtain the acceleration parameter by using Taylor's expansion of the first order given by the following approximation:

$$F(x_{k+1}) \approx F(x_k) + F'(\xi)(x_{k+1} - x_k), \tag{3.2.6}$$

where the parameter $\xi \in [x_k, x_{k+1}]$,

$$\xi = x_k + \delta(x_{k+1} - x_k), \quad 0 \leqslant \delta \leqslant 1. \tag{3.2.7}$$

Putting in mind that the distance between x_k and x_{k+1} is small enough, we can take $\delta = 1$ in (3.2.7) and get $\xi = x_{k+1}$. Thus we assume:

$$F'(\xi) \approx \gamma_{k+1} I. \tag{3.2.8}$$

Now from (3.2.6) and (3.2.8) its not difficult to verify that:

$$F(x_{k+1}) - F(x_k) = \gamma_{k+1}(x_{k+1} - x_k). \tag{3.2.9}$$

Taking $y_k = F(x_{k+1}) - F(x_k)$ and $s_k = x_{k+1} - x_k$, we have

$$y_k = \gamma_{k+1} s_k, \tag{3.2.10}$$

by multiplying y_k^T to the both side of (3.2.10) the acceleration parameter yields:

$$\gamma_{k+1} = \frac{y_k^T y_k}{y_k^T s_k}. \tag{3.2.11}$$

We use the derivative-free line search proposed by Li and Fukushima (1999) in order to compute our step length α_k.
Let $\omega_1 > 0$, $\omega_2 > 0$ and $r \in (0,1)$ be constants and let $\{\eta_k\}$ be a given positive sequence such that

$$\sum_{k=0}^{\infty} \eta_k < \eta < \infty, \tag{3.2.12}$$

and

$$\begin{aligned} f(x_k + (\alpha_k + \alpha_k^2 \gamma_k) d_k) - f(x_k) \leq & -\omega_1 \|\alpha_k F(x_k)\|^2 - \omega_2 \|\alpha_k d_k\|^2 \\ & + \eta_k f(x_k). \end{aligned} \tag{3.2.13}$$

Let i_k be the smallest non negative integer i such that (3.2.13) holds for $\alpha = r^i$. Let $\alpha_k = r^{i_k}$.

Now we describe the algorithm of the proposed method as follows:

```
Algorithm 2 (IDFDD)
STEP 1: Given $x_0$, $\gamma_0 = 0.01$, $\varepsilon = 10^{-4}$, set $k = 0$.
STEP 2: Compute $F(x_k)$.
STEP 3: If$\|F(x_k)\| \leq \varepsilon$ then stop, else goto STEP 4.
STEP 4: Compute search direction $d_k = -\gamma_k^{-1} F(x_k)$.
STEP 5: Compute step length $\alpha_k$(using (3.2.13)).
STEP 6: Set $x_{k+1} = x_k + (\alpha_k + \alpha_k^2 \gamma_k) d_k$.
STEP 7: Compute $F(x_{k+1})$.
STEP 8: Determine $\gamma_{k+1} = \frac{y_k^T y_k}{y_k^T s_k}$.
STEP 9: Set k=k+1, and go to STEP 3.
```

3.3 A DOUBLE STEP LENGTH DERIVATIVE-FREE METHOD (DSDF)

In this section we present the two step lengths α_k and β_k in (2.3.2) using inexact line search procedure. This is made possible by making the two directions to be equal, i.e g_k = h_k. In order to incorporate more information of the iterates at each iteration and to improve good direction towards the solution, we suggest a new directions g_k and h_k in (2.3.2) to be defined as:

$$g_k = -\gamma_k^{-1} F(x_k), \tag{3.3.1}$$

and

$$h_k = -\gamma_k^{-1} F(x_k), \tag{3.3.2}$$

where $\gamma_k > 0$ is an acceleration parameter.

By putting (3.3.1) and (3.3.2) into (2.3.2) we obtained

$$x_{k+1} = x_k - (\alpha_k + \beta_k) \gamma_k^{-1} F(x_k). \tag{3.3.3}$$

Now, from (3.3.3) we can easily see that, our direction is

$$d_k = -\gamma_k^{-1} F(x_k), \tag{3.3.4}$$

then using (3.3.3) and (3.3.4) we have the general scheme as:

$$x_{k+1} = x_k + (\alpha_k + \beta_k)d_k. \tag{3.3.5}$$

We use the derivative-free line search proposed by Li and Fukushima (1999) in order to compute our step lengths α_k and β_k.
Let $\omega_1 > 0$, $\omega_2 > 0$ and $q, r \in (0,1)$ be constants and let $\{\eta_k\}$ be a given positive sequence such that

$$\sum_{k=0}^{\infty} \eta_k < \eta < \infty, \tag{3.3.6}$$

and

$$\begin{aligned} f(x_k + (\alpha_k + \beta_k)d_k) - f(x_k) \leq &- \omega_1 \|(\alpha_k + \beta_k)F(x_k)\|^2 \\ &- \omega_2 \|(\alpha_k + \beta_k)d_k\|^2 + \eta_k f(x_k). \end{aligned} \tag{3.3.7}$$

Let i_k be the smallest non negative integer i such that (3.3.7) holds for $\alpha = r^i$ and $\beta = q^i$. Let $\alpha_k = r^{i_k}$ and $\beta_k = q^{i_k}$.
Now we describe the algorithm of the proposed method as follows:
Algorithm 3 (DSDF)

```
STEP 1: Given $x_0$, $\gamma_0 = 0.01$, $\varepsilon = 10^{-4}$, set $k = 0$.
STEP 2: Compute $F(x_k)$ .
STEP 3: If$\|F(x_k)\| \leq \varepsilon$,then stop,else goto STEP 4.
STEP 4: Compute search direction $d_k = -\gamma_k^{-1}F(x_k)$.
STEP 5: Compute step lengths $\alpha_k + \beta_k$ (using (3.3.7)).
STEP 7: Set $\lambda_k = \alpha_k + \beta_k$.
STEP 8: Compute $x_{k+1} = x_k + \lambda_k d_k$.
STEP 9: Compute $F(x_{k+1})$.
STEP 10: Determine $\gamma_{k+1} = \frac{y_k^T y_k}{y_k^T s_k}$.
STEP 11: Set k=k+1, and go to STEP 3.
```

3.4 A TRANSFORMED DOUBLE STEP LENGTH METHOD (TDS)

In this section we propose to reduce the two step lengths α_k and β_k in (2.3.2) into a single step length. This reduction is made possible by an additional assumption:

$$\beta_k = \frac{1}{2}\alpha_k. \tag{3.4.1}$$

we suggest a new directions g_k and h_k in (2.3.2) to be defined as:

$$g_k = -\gamma_k^{-1}F(x_k), \tag{3.4.2}$$

and

$$h_k = -F(x_k), \tag{3.4.3}$$

where $\gamma_k > 0$ is an acceleration parameter.

So by putting (3.4.1),(3.4.2) and (3.4.3) into (2.3.2) we obtained:

$$x_{k+1} = x_k - \left(\alpha_k + \frac{1}{2}\alpha_k\gamma_k\right)\gamma_k^{-1}F(x_k). \tag{3.4.4}$$

From (3.4.4) we can easily see that,our direction is :

$$d_k = -\gamma_k^{-1}F(x_k), \tag{3.4.5}$$

then using (3.4.4) and (3.4.5) we have the general scheme as:

$$x_{k+1} = x_k + \left(\alpha_k + \frac{1}{2}\alpha_k\gamma_k\right)d_k. \tag{3.4.6}$$

We use the derivative-free line search proposed by Li and Fukushima (1999) in order to compute our step length α_k.

Let $\omega_1 > 0$, $\omega_2 > 0$ and $r \in (0,1)$ be constants and let $\{\eta_k\}$ be a given positive sequence such that

$$\sum_{k=0}^{\infty}\eta_k < \eta < \infty, \tag{3.4.7}$$

and

$$
\begin{aligned}
f(x_k + (\alpha_k + \frac{1}{2}\alpha_k\gamma_k)d_k) - f(x_k) \leq & -\omega_1\|\alpha_k F(x_k)\|^2 \\
& -\omega_2\|\alpha_k d_k\|^2 + \eta_k f(x_k).
\end{aligned} \tag{3.4.8}
$$

Let i_k be the smallest non negative integer i such that (3.4.8) holds for $\alpha = r^i$. Let $\alpha_k = r^{i_k}$.

Now we describe the algorithm of the proposed method as follows:

```
Algorithm 4 (TDS)
STEP 1: Given $x_0$, $\gamma_0 = 0.01$, $\varepsilon = 10^{-4}$, set $k = 0$.
STEP 2: Compute $F(x_k)$.
STEP 3: If$\|F(x_k)\| \leq \varepsilon$ then stop, else goto STEP 4.
STEP 4: Compute search direction $d_k = -\gamma_k^{-1}F(x_k)$.
STEP 5: Compute step length $\alpha_k$(using (3.4.8))
STEP 7: Set $x_{k+1} = x_k + \left(\alpha_k + \frac{1}{2}\alpha_k\gamma_k\right)d_k$.
STEP 8: Compute $F(x_{k+1})$.
STEP 9: Determine $\gamma_{k+1} = \frac{y_k^T y_k}{y_k^T s_k}$.
STEP 20: Set k=k+1, and go to STEP 3.
```

3.5 CONVERGENCE ANALYSIS

In this section, we present the global convergence of our propose methods. To begin with, let us define the level set

$$
\Omega = \{x : \|F(x)\| \leq \|F(x_0)\|\}. \tag{3.5.1}
$$

In order to analyze the convergence of algorithms 2,3 and 4, we need the following assumptions:

Assumption 1.

(1) There exists $x^* \in \mathbb{R}^n$ such that $F(x^*) = 0$.

(2) F is continuously differentiable in some neighborhood say N of x^* containing Ω.

(3) The Jacobian of F is bounded and positive definite on N. i.e there exists a positive constants $M > m > 0$ such that

$$\|F'(x)\| \leq M \quad \forall x \in N, \tag{3.5.2}$$

and

$$m\|d\|^2 \leq d^T F'(x)d \quad \forall x \in N, d \in \mathbb{R}^n. \tag{3.5.3}$$

Remarks:

Assumption 1 implies that there exists a constants $M > m > 0$ such that

$$m\|d\| \leq \|F'(x)d\| \leq M\|d\| \quad \forall x \in N, d \in \mathbb{R}^n. \tag{3.5.4}$$

$$m\|x-y\| \leq \|F(x)-F(y)\| \leq M\|x-y\| \quad \forall x,y \in N. \tag{3.5.5}$$

In particular $\forall x \in N$ we have

$$m\|x-x^*\| \leq \|F(x)\| = \|F(x)-F(x^*)\| \leq M\|x-x^*\|, \tag{3.5.6}$$

where x^* stands for the unique solution of (1.2.1) in N.

Since $\gamma_k I$ approximates $F'(x_k)$ along direction s_k, we can give the following assumption.

Assumption 2.

$\gamma_k I$ is a good approximation to $F'(x_k)$, i.e

$$\|(F'(x_k) - \gamma_k I)d_k\| \leq \varepsilon\|F(x_k)\|, \tag{3.5.7}$$

where $\varepsilon \in (0,1)$ is a small quantity (Yuan* & Lu, 2008).

3.6 CONVERGENCE RESULT OF AN IMPROVED DERIVATIVE-FREE METHOD VIA DOUBLE DIRECTION APPROACH (IDFDD)

In this section we prove the global convergence of IDFDD algorithm.

Lemma 3.1. Suppose assumption 2 holds and let $\{x_k\}$ be generated by IDFDD algorithm . Then d_k is a descent direction for $f(x_k)$ at x_k i.e

$$\nabla f(x_k)^T d_k < 0. \tag{3.6.1}$$

Proof. From (3.1.2) and (3.2.4), we have

$$\begin{aligned}\nabla f(x_k)^T d_k &= F(x_k)^T F'(x_k) d_k \\ &= F(x_k)^T [(F'(x_k) - \gamma_k I) d_k - F(x_k)] \\ &= F(x_k)^T (F'(x_k) - \gamma_k I) d_k - \|F(x_k)\|^2,\end{aligned} \tag{3.6.2}$$

by cauchy-schwaz we have,

$$\begin{aligned}\nabla f(x_k)^T d_k &\leq \|F(x_k)\| \|(F'(x_K) - \gamma_k I) d_k\| - \|F(x_k)\|^2 \\ &\leq -(1-\varepsilon)\|F(x_k)\|^2.\end{aligned} \tag{3.6.3}$$

Hence for $\varepsilon \in (0,1)$ this lemma is true.

By the above lemma, we can deduce that the norm function $f(x_k)$ is a descent along d_k, which means that $\|F(x_{k+1})\| \leq \|F(x_k)\|$ is true for all k.

Lemma 3.2. Suppose assumption 1 holds and let $\{x_k\}$ be generated by IDFDD algorithm. Then $\{x_k\} \subset \Omega$.

Proof. By lemma 3.1 we have $\|F(x_{k+1})\| \leq \|F(x_k)\|$. Moreover, we have for all k,

$$\|F(x_{k+1})\| \leq \|F(x_k)\| \leq \|F(x_{k-1})\| \ldots \leq \|F(x_0)\|.$$

This implies that $\{x_k\} \subset \Omega$.

Lemma 3.3 (Yuan* & Lu, 2008). Suppose assumption 1 holds and $\{x_k\}$ be generated by IDFDD algorithm. Then there exists a constant $m > 0$ such that for all k,

$$y_k^T s_k \geq m\|s_k\|^2. \tag{3.6.4}$$

Proof. By mean-value theorem and (3.5.3) we have,
$y_k^T s_k = s_k^T(F(x_{k+1}) - F(x_k)) = s_k^T F'(\zeta) s_k \geq m\|s_k\|^2$,
where $\xi = x_k + \zeta(x_{k+1} - x_k)$, $\zeta \in (0,1)$. The proof is complete.

Using $y_k^T s_k \geq m\|s_k\|^2 > 0$, γ_{k+1} is always generated by the update formula (3.2.11), and we can deduce that $\gamma_{k+1} I$ inherits the positive definiteness of $\gamma_k I$. By the above lemma and (3.5.5), we obtained

$$\frac{y_k^T s_k}{\|s_k\|^2} \geq m, \qquad \frac{\|y_k\|^2}{y_k^T s_k} \leq \frac{M^2}{m}. \tag{3.6.5}$$

Lemma 3.4. Suppose that assumption 1 holds and $\{x_k\}$ is generated by IDFDD algorithm. Then we have

$$\lim_{k\to\infty} \|\alpha_k d_k\| = \lim_{k\to\infty} \|s_k\| = 0, \tag{3.6.6}$$

and

$$\lim_{k\to\infty} \|\alpha_k F(x_k)\| = 0. \tag{3.6.7}$$

Proof. By (3.2.13) we have for all $k > 0$

$$\begin{aligned} \omega_2\|\alpha_k d_k\|^2 &\leq \omega_1\|\alpha_k F(x_k)\|^2 + \omega_2\|\alpha_k d_k\|^2 \\ &\leq \|F(x_k)\|^2 - \|F(x_{k+1})\|^2 + \eta_k\|F(x_k)\|^2. \end{aligned} \tag{3.6.8}$$

By summing the above inequality, we have

$$\begin{aligned}\omega_2 \sum_{i=0}^{k} \|\alpha_i d_i\|^2 &\leq \sum_{i=0}^{k} \left(\|F(x_i)\|^2 - \|F(x_{i+1})\|^2\right) + \sum_{i=0}^{k} \eta_i \|F(x_i)\|^2 \\ &= \|F(x_0)\|^2 - \|F(x_{k+1})\|^2 + \sum_{i=0}^{k} \eta_i \|F(x_i)\|^2 \\ &\leq \|F(x_0)\|^2 + \|F(x_0)\|^2 \sum_{i=0}^{k} \eta_i \\ &\leq \|F(x_0)\|^2 + \|F(x_0)\|^2 \sum_{i=0}^{\infty} \eta_i.\end{aligned} \tag{3.6.9}$$

So from the level set and the fact that $\{\eta_k\}$ satisfies (3.2.12) then the series $\sum_{i=0}^{\infty} \|\alpha_i d_i\|^2$ is convergent. This implies (3.6.6). By similar arguments as above but with $\omega_1 \|\alpha_k F(x_k)\|^2$ on the left hand side, we obtain (3.6.7).

Lemma 3.5. Suppose assumption 1 holds and let $\{x_k\}$ be generated by IDFDD algorithm. Then there exists a constant $m_3 > 0$ such that for all $k > 0$,

$$\|d_k\| \leq m_3. \tag{3.6.10}$$

Proof. From (3.5.5) we have,

$$\begin{aligned}\|d_k\| &= \left\| -\frac{y_{k-1}^T s_{k-1} F(x_k)}{\|y_{k-1}\|^2} \right\| \\ &\leq \frac{\|F(x_k)\| \|s_{k-1}\| \|y_{k-1}\|}{m^2 \|s_{k-1}\|^2} \\ &\leq \frac{\|F(x_k)\| M \|s_{k-1}\|}{m^2 \|s_{k-1}\|} \\ &\leq \frac{\|F(x_k)\| M}{m^2} \\ &\leq \frac{\|F(x_0)\| M}{m^2}.\end{aligned} \tag{3.6.11}$$

Taking $m_3 = \frac{\|F(x_0)\|M}{m^2}$, we have (3.6.10).

We now establish the global convergence theorem for IDFDD Algorithm .

Theorem 3.1. Suppose that assumption 1 holds,$\{x_k\}$ is generated by IDFDD algorithm . Assume further for all $k > 0$,

$$\alpha_k \geq c\frac{|F(x_k)^T d_k|}{\|d_k\|^2}, \tag{3.6.12}$$

where c is some positive constant. Then

$$\lim_{k\to\infty} \|F(x_k)\| = 0. \tag{3.6.13}$$

Proof. From lemma 3.5 we have (3.6.10). Therefore by (3.6.6) and the boundedness of $\{\|d_k\|\}$, we have

$$\lim_{k\to\infty} \alpha_k \|d_k\|^2 = 0, \tag{3.6.14}$$

from (3.6.12) and (3.6.14) we have

$$\lim_{k\to\infty} |F(x_k)^T d_k| = 0. \tag{3.6.15}$$

On the other hand, from (3.2.4) we have,

$$F(x_k)^T d_k = -\gamma_k^{-1}\|F(x_k)\|^2, \tag{3.6.16}$$

$$\begin{aligned} \|F(x_k)\|^2 &= \| -F(x_k)^T d_k \gamma_k\| \\ &\leq |F(x_k)^T d_k||\gamma_k|. \end{aligned} \tag{3.6.17}$$

But

$$\gamma_k^{-1} = \frac{y_{k-1}^T s_{k-1}}{\|y_{k-1}\|^2} \geq \frac{m\|s_{k-1}\|^2}{\|y_{k-1}\|^2} \geq \frac{m\|s_{k-1}\|^2}{M^2\|s_{k-1}\|^2} = \frac{m}{M^2}.$$

Then,

$$|\gamma_k^{-1}| \geq \frac{m}{M^2},$$

so from (3.6.17) we have,

$$\|F(x_k)\|^2 \leq |F(x_k)^T d_k| \left(\frac{M^2}{m}\right). \tag{3.6.18}$$

Thus,

$$0 \leq \|F(x_k)\|^2 \leq |F(x_k)^T d_k| \left(\frac{M^2}{m}\right) \longrightarrow 0. \tag{3.6.19}$$

Therefore,

$$\lim_{k\to\infty} \|F(x_k)\| = 0. \tag{3.6.20}$$

The proof is completed.

3.7 CONVERGENCE RESULT OF A DOUBLE STEP LENGTH DERIVATIVE-FREE METHOD (DSDF)

In this section we prove the global convergence of DSDF algorithm.

Lemma 3.6. Suppose assumption 2 holds and let $\{x_k\}$ be generated by DSDF algorithm. Then d_k is a descent direction for $f(x_k)$ at x_k i.e

$$\nabla f(x_k)^T d_k < 0. \tag{3.7.1}$$

Proof. From (3.1.2) and (3.3.4), we have

$$\begin{aligned}\nabla f(x_k)^T d_k &= F(x_k)^T F'(x_k) d_k \\ &= F(x_k)^T [(F'(x_k) - \gamma_k I) d_k - F(x_k)] \\ &= F(x_k)^T (F'(x_k) - \gamma_k I) d_k - \|F(x_k)\|^2,\end{aligned} \tag{3.7.2}$$

by cauchy-schwaz we have,

$$\begin{aligned}\nabla f(x_k)^T d_k &\leq \|F(x_k)\|\|(F'(x_K)-\gamma_k I)d_k\| - \|F(x_k)\|^2 \\ &\leq -(1-\varepsilon)\|F(x_k)\|^2.\end{aligned} \tag{3.7.3}$$

Hence for $\varepsilon \in (0,1)$ this lemma is true.
By the above lemma, we can deduce that the norm function $f(x_k)$ is a descent along d_k, which means that $\|F(x_{k+1})\| \leq \|F(x_k)\|$ is true for all k.

Lemma 3.7. Suppose assumption 1 holds and let $\{x_k\}$ be generated by DSDF algorithm. Then $\{x_k\} \subset \Omega$.
Proof. By lemma 3.6 we have $\|F(x_{k+1})\| \leq \|F(x_k)\|$. Moreover, we have for all k,

$$\|F(x_{k+1})\| \leq \|F(x_k)\| \leq \|F(x_{k-1})\| \ldots \leq \|F(x_0)\|.$$

This implies that $\{x_k\} \subset \Omega$.

Lemma 3.8 (Yuan* & Lu, 2008). Suppose assumption 1 holds and $\{x_k\}$ be generated by DSDF algorithm. Then there exists a constant $m > 0$ such that for all k,

$$y_k^T s_k \geq m\|s_k\|^2. \tag{3.7.4}$$

Proof. By mean-value theorem and (3.5.3) we have,
$y_k^T s_k = s_k^T(F(x_{k+1}) - F(x_k)) = s_k^T F'(\zeta)s_k \geq m\|s_k\|^2$,
where $\xi = x_k + \zeta(x_{k+1} - x_k)$, $\zeta \in (0,1)$. The proof is complete.

Using $y_k^T s_k \geq m\|s_k\|^2 > 0$, γ_{k+1} is always generated by the update formula (3.2.11), and we can deduce that $\gamma_{k+1}I$ inherits the positive definiteness of $\gamma_k I$. By the above lemma and (3.5.5), we obtained

$$\frac{y_k^T s_k}{\|s_k\|^2} \geq m, \qquad \frac{\|y_k\|^2}{y_k^T s_k} \leq \frac{M^2}{m}. \tag{3.7.5}$$

Lemma 3.9. Suppose that assumption 1 holds and $\{x_k\}$ is generated by DSDF algorithm. Then we have

$$\lim_{k\to\infty} \|\lambda_k d_k\| = \lim_{k\to\infty} \|s_k\| = 0, \tag{3.7.6}$$

and

$$\lim_{k\to\infty} \|\lambda_k F(x_k)\| = 0. \tag{3.7.7}$$

Proof. By (3.3.7) we have for all $k > 0$

$$\begin{aligned} \omega_2\|\lambda_k d_k\|^2 &\leq \omega_1\|\lambda_k F(x_k)\|^2 + \omega_2\|\lambda_k d_k\|^2 \\ &\leq \|F(x_k)\|^2 - \|F(x_{k+1})\|^2 + \eta_k\|F(x_k)\|^2. \end{aligned} \tag{3.7.8}$$

By summing the above inequality, we have

$$\begin{aligned} \omega_2 \sum_{i=0}^{k} \|\lambda_i d_i\|^2 &\leq \sum_{i=0}^{k} \left(\|F(x_i)\|^2 - \|F(x_{i+1})\|^2\right) + \sum_{i=0}^{k} \eta_i \|F(x_i)\|^2 \\ &= \|F(x_0)\|^2 - \|F(x_{k+1})\|^2 + \sum_{i=0}^{k} \eta_i \|F(x_i)\|^2 \\ &\leq \|F(x_0)\|^2 + \|F(x_0)\|^2 \sum_{i=0}^{k} \eta_i \\ &\leq \|F(x_0)\|^2 + \|F(x_0)\|^2 \sum_{i=0}^{\infty} \eta_i. \end{aligned} \tag{3.7.9}$$

So from the level set and the fact that $\{\eta_k\}$ satisfies (3.3.6) then the series $\sum_{i=0}^{\infty} \|\lambda_i d_i\|^2$ is convergent. This implies (3.7.6). By similar arguments as above but with $\omega_1\|\lambda_k F(x_k)\|^2$ on the left hand side, we obtain (3.7.7).

Lemma 3.10. Suppose assumption 1 holds and let $\{x_k\}$ be generated by DSDF algorithm. Then there exists a constant $m_3 > 0$ such that for all $k > 0$,

$$\|d_k\| \le m_3. \tag{3.7.10}$$

Proof. From (3.5.5) we have,

$$\begin{aligned}
\|d_k\| &= \left\| -\frac{y_{k-1}^T s_{k-1} F(x_k)}{\|y_{k-1}\|^2} \right\| \\
&\le \frac{\|F(x_k)\| \|s_{k-1}\| \|y_{k-1}\|}{m^2 \|s_{k-1}\|^2} \\
&\le \frac{\|F(x_k)\| M \|s_{k-1}\|}{m^2 \|s_{k-1}\|} \\
&\le \frac{\|F(x_k)\| M}{m^2} \\
&\le \frac{\|F(x_0)\| M}{m^2}.
\end{aligned} \tag{3.7.11}$$

Taking $m_3 = \frac{\|F(x_0)\| M}{m^2}$, we have (3.7.10).

We now establish the global convergence theorem for DSDF Algorithm .

Theorem 3.2. Suppose that assumption 1 holds,$\{x_k\}$ is generated by DSDF algorithm . Assume further for all $k > 0$,

$$\lambda_k \ge c \frac{|F(x_k)^T d_k|}{\|d_k\|^2}, \tag{3.7.12}$$

where c is some positive constant. Then

$$\lim_{k \to \infty} \|F(x_k)\| = 0. \tag{3.7.13}$$

Proof. From lemma 3.10 we have (3.7.10). Therefore by (3.7.6) and the boundedness of $\{\|d_k\|\}$, we have

$$\lim_{k \to \infty} \lambda_k \|d_k\|^2 = 0, \tag{3.7.14}$$

from (3.7.12) and (3.7.14) we have

$$\lim_{k\to\infty} |F(x_k)^T d_k| = 0. \tag{3.7.15}$$

On the other hand, from (3.3.4) we have,

$$F(x_k)^T d_k = -\gamma_k^{-1} \|F(x_k)\|^2, \tag{3.7.16}$$

$$\begin{aligned} \|F(x_k)\|^2 &= \| - F(x_k)^T d_k \gamma_k\| \\ &\leq |F(x_k)^T d_k||\gamma_k|. \end{aligned} \tag{3.7.17}$$

But

$$\gamma_k^{-1} = \frac{y_{k-1}^T s_{k-1}}{\|y_{k-1}\|^2} \geq \frac{m\|s_{k-1}\|^2}{\|y_{k-1}\|^2} \geq \frac{m\|s_{k-1}\|^2}{M^2\|s_{k-1}\|^2} = \frac{m}{M^2}.$$

Then,

$$|\gamma_k^{-1}| \geq \frac{m}{M^2},$$

so from (3.7.17) we have,

$$\|F(x_k)\|^2 \leq |F(x_k)^T d_k| \left(\frac{M^2}{m}\right). \tag{3.7.18}$$

Thus,

$$0 \leq \|F(x_k)\|^2 \leq |F(x_k)^T d_k| \left(\frac{M^2}{m}\right) \longrightarrow 0. \tag{3.7.19}$$

Therefore,

$$\lim_{k\to\infty} \|F(x_k)\| = 0. \tag{3.7.20}$$

The proof is completed.

3.8 CONVERGENCE RESULT OF A TRANSFORMED DOUBLE STEP LENGTH METHOD (TDS)

In this section we prove the global convergence of TDS algorithm.

Lemma 3.11. Suppose assumption 2 holds and let $\{x_k\}$ be generated by TDS algorithm . Then d_k is a descent direction for $f(x_k)$ at x_k i.e

$$\nabla f(x_k)^T d_k < 0. \tag{3.8.1}$$

Proof. From (3.1.2) and (3.4.5), we have

$$\begin{aligned}\nabla f(x_k)^T d_k &= F(x_k)^T F'(x_k) d_k \\ &= F(x_k)^T[(F'(x_k) - \gamma_k I)d_k - F(x_k)] \\ &= F(x_k)^T(F'(x_k) - \gamma_k I)d_k - \|F(x_k)\|^2,\end{aligned} \tag{3.8.2}$$

by cauchy-schwaz we have,

$$\begin{aligned}\nabla f(x_k)^T d_k &\leq \|F(x_k)\| \|(F'(x_K) - \gamma_k I)d_k\| - \|F(x_k)\|^2 \\ &\leq -(1-\varepsilon)\|F(x_k)\|^2.\end{aligned} \tag{3.8.3}$$

Hence for $\varepsilon \in (0,1)$ this lemma is true.
By the above lemma, we can deduce that the norm function $f(x_k)$ is a descent along d_k, which means that $\|F(x_{k+1})\| \leq \|F(x_k)\|$ is true for all k.

Lemma 3.12. Suppose assumption 1 holds and let $\{x_k\}$ be generated by TDS algorithm. Then $\{x_k\} \subset \Omega$.
Proof. By lemma 3.11 we have $\|F(x_{k+1})\| \leq \|F(x_k)\|$. Moreover, we have for all k,

$$\|F(x_{k+1})\| \leq \|F(x_k)\| \leq \|F(x_{k-1})\| \ldots \leq \|F(x_0)\|.$$

This implies that $\{x_k\} \subset \Omega$.

Lemma 3.13 (Yuan* & Lu, 2008). Suppose assumption 1 holds and $\{x_k\}$ be generated by TDS algorithm. Then there exists a constant $m > 0$ such that for all k,

$$y_k^T s_k \geq m\|s_k\|^2. \tag{3.8.4}$$

Proof. By mean-value theorem and (3.5.3) we have,
$y_k^T s_k = s_k^T(F(x_{k+1}) - F(x_k)) = s_k^T F'(\zeta)s_k \geq m\|s_k\|^2$,
where $\xi = x_k + \zeta(x_{k+1} - x_k)$, $\zeta \in (0,1)$. The proof is complete.

Using $y_k^T s_k \geq m\|s_k\|^2 > 0$, γ_{k+1} is always generated by the update formula (3.2.11), and we can deduce that $\gamma_{k+1}I$ inherits the positive definiteness of $\gamma_k I$. By the above lemma and (3.5.5), we obtained

$$\frac{y_k^T s_k}{\|s_k\|^2} \geq m, \qquad \frac{\|y_k\|^2}{y_k^T s_k} \leq \frac{M^2}{m}. \tag{3.8.5}$$

Lemma 3.14. Suppose that assumption 1 holds and $\{x_k\}$ is generated by TDS algorithm. Then we have

$$\lim_{k\to\infty} \|\alpha_k d_k\| = \lim_{k\to\infty} \|s_k\| = 0, \tag{3.8.6}$$

and

$$\lim_{k\to\infty} \|\alpha_k F(x_k)\| = 0. \tag{3.8.7}$$

Proof. By (3.4.8) we have for all $k > 0$

$$\begin{aligned}\omega_2\|\alpha_k d_k\|^2 &\leq \omega_1\|\alpha_k F(x_k)\|^2 + \omega_2\|\alpha_k d_k\|^2 \\ &\leq \|F(x_k)\|^2 - \|F(x_{k+1})\|^2 + \eta_k\|F(x_k)\|^2.\end{aligned} \tag{3.8.8}$$

By summing the above inequality, we have

$$\begin{aligned}
\omega_2 \sum_{i=0}^{k} \|\alpha_i d_i\|^2 &\leq \sum_{i=0}^{k} \left(\|F(x_i)\|^2 - \|F(x_{i+1})\|^2\right) + \sum_{i=0}^{k} \eta_i \|F(x_i)\|^2 \\
&= \|F(x_0)\|^2 - \|F(x_{k+1})\|^2 + \sum_{i=0}^{k} \eta_i \|F(x_i)\|^2 \\
&\leq \|F(x_0)\|^2 + \|F(x_0)\|^2 \sum_{i=0}^{k} \eta_i \\
&\leq \|F(x_0)\|^2 + \|F(x_0)\|^2 \sum_{i=0}^{\infty} \eta_i .
\end{aligned} \tag{3.8.9}$$

So from the level set and the fact that $\{\eta_k\}$ satisfies (3.4.7) then the series $\sum_{i=0}^{\infty} \|\alpha_i d_i\|^2$ is convergent. This implies (3.8.6). By similar arguments as above but with $\omega_1 \|\alpha_k F(x_k)\|^2$ on the left hand side, we obtain (3.8.7).

Lemma 3.15. Suppose assumption 1 holds and let $\{x_k\}$ be generated by TDS algorithm. Then there exists a constant $m_3 > 0$ such that for all $k > 0$,

$$\|d_k\| \leq m_3. \tag{3.8.10}$$

Proof. From (3.5.5) we have,

$$\begin{aligned}
\|d_k\| &= \left\| -\frac{y_{k-1}^T s_{k-1} F(x_k)}{\|y_{k-1}\|^2} \right\| \\
&\leq \frac{\|F(x_k)\| \|s_{k-1}\| \|y_{k-1}\|}{m^2 \|s_{k-1}\|^2} \\
&\leq \frac{\|F(x_k)\| M \|s_{k-1}\|}{m^2 \|s_{k-1}\|} \\
&\leq \frac{\|F(x_k)\| M}{m^2} \\
&\leq \frac{\|F(x_0)\| M}{m^2} .
\end{aligned} \tag{3.8.11}$$

Taking $m_3 = \frac{\|F(x_0)\|M}{m^2}$, we have (3.8.10).

We now establish the global convergence theorem for TDS Algorithm .

Theorem 3.3. Suppose that assumption 1 holds,$\{x_k\}$ is generated by TDS algorithm. Assume further for all $k > 0$,

$$\alpha_k \geq c\frac{|F(x_k)^T d_k|}{\|d_k\|^2}, \tag{3.8.12}$$

where c is some positive constant. Then

$$\lim_{k\to\infty} \|F(x_k)\| = 0. \tag{3.8.13}$$

Proof. From lemma 3.15 we have (3.8.10). Therefore by (3.8.6) and the boundedness of $\{\|d_k\|\}$, we have

$$\lim_{k\to\infty} \alpha_k \|d_k\|^2 = 0, \tag{3.8.14}$$

from (3.8.12) and (3.8.14) we have

$$\lim_{k\to\infty} |F(x_k)^T d_k| = 0. \tag{3.8.15}$$

On the other hand, from (3.4.5) we have,

$$F(x_k)^T d_k = -\gamma_k^{-1} \|F(x_k)\|^2, \tag{3.8.16}$$

$$\begin{aligned} \|F(x_k)\|^2 &= \| - F(x_k)^T d_k \gamma_k\| \\ &\leq |F(x_k)^T d_k||\gamma_k|. \end{aligned} \tag{3.8.17}$$

But

$$\gamma_k^{-1} = \frac{y_{k-1}^T s_{k-1}}{\|y_{k-1}\|^2} \geq \frac{m\|s_{k-1}\|^2}{\|y_{k-1}\|^2} \geq \frac{m\|s_{k-1}\|^2}{M^2\|s_{k-1}\|^2} = \frac{m}{M^2}.$$

Then,

$$|\gamma_k^{-1}| \geq \frac{m}{M^2},$$

so from (3.8.17) we have,

$$\|F(x_k)\|^2 \leq |F(x_k)^T d_k| \left(\frac{M^2}{m}\right). \tag{3.8.18}$$

Thus,

$$0 \leq \|F(x_k)\|^2 \leq |F(x_k)^T d_k| \left(\frac{M^2}{m}\right) \longrightarrow 0. \tag{3.8.19}$$

Therefore,

$$\lim_{k\to\infty} \|F(x_k)\| = 0. \tag{3.8.20}$$

The proof is completed.

CHAPTER FOUR

NUMERICAL RESULTS

4.1 INTRODUCTION

In this chapter, the performance of the proposed methods so far introduced in this dissertation, (i.e, an improved derivative-free method via double direction approach (IDFDD), a double step length derivative-free method (DSDF) and a transformed double step length method (TDS) for solving large-scale systems of nonlinear equations) are compared with that of an inexact PRP (Polak-Rebiȅre-Polyak) conjugate gradient (CG) method for symmetric nonlinear equations (IPRP) (Zhou & Shen, 2014), a derivative-free CG method and its global convergence for solving symmetric nonlinear equations (DFCG) (Waziri & Sabiu, 2015) and spectral DY-Type projection method for nonlinear monotone systems of equations (SDYP) (Liu & Li, 2015).
The codes, were written in Matlab 7.9.0 (R2009b) and run on a computer of 2.00 GHz CPU processor and 3 GB RAM memory. We use 20 test problems with dimension between 10 to 10,000 in order to test the advantages of the proposed methods in terms of number of iterations (NI) and the CPU Time (in seconds). We declare a termination of the methods whenever

$$\|F(\mathrm{x}_k)\| \leq 10^{-4}.$$

We however declared a failure if any of the following two situations occurred during iteration process:

1. The number of iteration reaches 1000, but no x_k satisfying $\|F(x_k)\| \leq 10^{-4}$ is obtained.

2. Failure on code execution due to insufficient memory.

We use the symbol (—) if the algorithm fails to find a solution.
Details of the test functions are given in the appendix A. Subsequently, in order to show the performance of the proposed methods clearly, we plotted figures according to the data in Tables 4.1-4.3 by using the performance profiles of Dolan and More (2002). In each of the tested problem, we used the dimensions 10, 100, 1000 and 10000 in order to test the performance of the proposed methods.
In Tables 4.4-4.6 we also report the behavior of the IDFDD algorithm, DSDF algorithm and TDS algorithm for Functions 1, 6, 9 and 15 with some different initial points, in order to illustrate the global convergence.

4.2 COMPUTATIONAL EXPERIMENTS

In this section, we present the numerical results as follows:

Table 4.1: Numerical Result for IDFDD and IPRP of problems 1-20.

.

		IDFDD				IPRP			
Problems	Dimension	NI	CPU time	$\|F(x_k)\|$	x^{*A}	NI	CPU time	$\|F(x_k)\|$	x^{*B}
1	10	6	0.003035	1.22E-05	$(1,1,...,1)^T$	13	0.005344	5.99E-05	$(-1,-1,...,-1)^T$
	100	6	0.004822	3.87E-05	$(1,1,...,1)^T$	14	0.011389	7.36E-07	$(-1,-1,...,-1)^T$
	1000	7	0.018732	1.26E-06	$(1,1,...,1)^T$	16	0.031001	4.85E-05	$(-1,-1,...,-1)^T$
	10000	7	0.116143	3.98E-06	$(1,1,...,1)^T$	19	0.158476	2.92E-05	$(-1,-1,...,-1)^T$
2	10	6	0.004375	2.49E-05	$(1,1,...,1)^T$	74	0.037693	8.54E-05	$(-2,-2,...,-2)^T$
	100	6	0.005477	7.88E-05	$(1,1,...,1)^T$	84	0.053628	8.85E-05	$(-2,-2,...,-2)^T$
	1000	7	0.023009	2.53E-06	$(1,1,...,1)^T$	94	0.141518	9.17E-05	$(-2,-2,...,-2)^T$
	10000	7	0.108244	8.01E-06	$(1,1,...,1)^T$	104	1.224821	9.50E-05	$(-2,-2,...,-2)^T$
3	10	7	0.004297	2.39E-06	$(1,1,...,1)^T$	10	0.006374	4.20E-05	$(1,1,...,1)^T$
	100	6	0.007428	6.70E-06	$(1,1,...,1)^T$	25	0.029915	7.22E-05	$(1,1,...,1)^T$
	1000	6	0.037075	9.50E-06	$(1,1,...,1)^T$	20	0.076751	5.65E-05	$(1,1,...,1)^T$
	10000	7	0.183632	3.64E-06	$(1,1,...,1)^T$	30	0.816934	2.20E-05	$(1,1,...,1)^T$
4	10	6	0.005103	3.47E-06	$(1,1,...,1)^T$	19	1.98E-02	5.66E-05	$(1,1,...,1)^T$
	100	6	0.013766	1.83E-06	$(1,1,...,1)^T$	22	0.044508	2.34E-05	$(1,1,...,1)^T$
	1000	7	0.061831	2.32E-06	$(1,1,...,1)^T$	—	—	—	—
	10000	8	0.300182	3.44E-05	$(1,1,...,1)^T$	—	—	—	—
5	10	6	0.006879	3.08E-06	$(1,1,...,1)^T$	19	0.011114	7.02E-05	$(1,1,...,1)^T$
	100	6	0.006364	9.75E-06	$(1,1,...,1)^T$	21	0.016205	6.88E-05	$(1,1,...,1)^T$
	1000	6	0.011127	3.08E-05	$(1,1,...,1)^T$	23	0.048532	6.74E-05	$(1,1,...,1)^T$
	10000	6	0.096672	9.75E-05	$(1,1,...,1)^T$	25	0.321242	6.60E-05	$(1,1,...,1)^T$
6	10	6	0.004391	2.27E-06	$(1,1,...,1)^T$	15	0.012938	2.97E-06	$(1,1,...,1)^T$
	100	6	0.007188	7.17E-06	$(1,1,...,1)^T$	15	0.0102	9.38E-06	$(1,1,...,1)^T$
	1000	6	0.028501	2.27E-05	$(1,1,...,1)^T$	15	0.054797	2.97E-05	$(1,1,...,1)^T$
	10000	6	0.141485	7.17E-05	$(1,1,...,1)^T$	15	0.201566	9.38E-05	$(1,1,...,1)^T$
7	10	6	0.004575	1.02E-05	$(1,1,...,1)^T$	10	0.005527	7.07E-05	$(1,1,...,1)^T$
	100	6	0.005721	3.21E-05	$(1,1,...,1)^T$	12	0.009458	3.70E-05	$(1,1,...,1)^T$
	1000	7	0.024393	1.02E-06	$(1,1,...,1)^T$	12	0.043562	5.77E-05	$(1,1,...,1)^T$
	10000	7	0.130745	3.22E-06	$(1,1,...,1)^T$	14	0.183229	4.93E-05	$(1,1,...,1)^T$
8	10	6	0.004315	4.30E-05	$(2,2,...,2)^T$	25	0.017891	7.04E-05	$(-0.0714,...,-0.0714)^T$
	100	7	0.009012	2.58E-05	$(2,2,...,2)^T$	14	0.022899	5.85E-05	$(-0.0071,...,-0.0071)^T$
	1000	8	0.040418	1.27E-05	$(2,2,...,2)^T$	25	0.131472	8.22E-05	$(-0.7081,...,-0.7081)^T$
	10000	8	0.248943	7.87E-05	$(2,2,...,2)^T$	—	—	—	—
9	10	5	0.00999	7.03E-05	$(-2.224x10^{-5},...)^T$	7	0.003152	1.04E-07	$(3.273x10^{-8},...)^T$
	100	6	0.004478	8.79E-05	$(8.795x10^{-6},...)^T$	7	0.004767	3.27E-07	$(3.273x10^{-8},...)^T$
	1000	7	0.019501	8.34E-06	$(-8.795x10^{-8},...)^T$	7	0.017545	1.04E-06	$(3.273x10^{-8},...)^T$
	10000	7	0.088119	2.64E-05	$(-8.795x10^{-8},...)^T$	7	0.101793	3.27E-06	$(3.273x10^{-8},...)^T$
10	10	6	0.003427	3.48E-05	$(-0.0034,...,-0.0034)^T$	331	0.085082	9.97E-05	$(-0.0995,...,-0.0995)^T$
	100	12	0.016601	4.32E-05	$(0.0019,...,0.0019)^T$	—	—	—	—
	1000	13	0.110745	5.51E-05	$(-0.0016,...,-0.0016)^T$	—	—	—	—
	10000	22	0.755329	8.17E-05	$(0.4030,...,0.4030)^T$	—	—	—	—

Table 4.1: Continued.

		IDFDD				IPRP			
Problems	Dimension	NI	CPU time	$\|F(x_k)\|$	x^{*A}	NI	CPU time	$\|F(x_k)\|$	x^{*B}
11	10	7	0.003457	2.52E-06	$(-7.972x10^{-7},...)^T$	14	0.012328	8.15E-05	$(-2.577x10^{-5},...)^T$
	100	7	0.004497	7.97E-06	$(-7.972x10^{-7},...)^T$	15	0.006579	9.82E-05	$(-9.823x10^{-6},...)^T$
	1000	7	0.016999	2.52E-05	$(-7.972x10^{-7},...)^T$	17	0.027774	4.53E-05	$(-1.432x10^{-6},...)^T$
	10000	7	0.095132	7.97E-05	$(-7.972x10^{-7},...)^T$	18	0.143145	5.49E-05	$(-1.432x10^{-6},...)^T$
12	10	21	0.017244	5.22E-05	$(0.0041,...,0.0041)^T$	—	—	—	—
	100	22	0.012887	7.93E-05	$(0.0028,...,0.0028)^T$	—	—	—	—
	1000	25	0.083906	8.83E-05	$(0.0017,...,0.0017)^T$	—	—	—	—
	10000	35	1.208837	9.35E-05	$(0.9660,...,0.9660)^T$	—	—	—	—
13	10	5	0.022512	5.54E-05	$(1.591x10^{-6},...)^T$	14	0.027285	3.61E-05	$(1.038x10^{-6},...)^T$
	100	6	0.007518	1.82E-06	$(-1.655x10^{-8},...)^T$	15	0.01453	2.73E-05	$(2.480x10^{-7},...)^T$
	1000	6	0.031373	5.76E-06	$(-1.655x10^{-8},...)^T$	15	0.052513	8.63E-05	$(2.480x10^{-7},...)^T$
	10000	6	0.173182	1.82E-05	$(-1.655x10^{-8},...)^T$	17	0.327498	1.59E-05	$(1.446x10^{-7},...)^T$
14	10	6	0.003469	8.19E-05	$(1,1,...,1)^T$	10	0.023028	5.83E-05	$(-0.1809,...,-0.1809)^T$
	100	7	0.005333	3.04E-06	$(1,1,...,1)^T$	12	0.009055	2.38E-05	$(-0.1809,...,-0.1809)^T$
	1000	7	0.032281	9.60E-06	$(1,1,...,1)^T$	12	0.042073	7.51E-05	$(-0.1809,...,-0.1809)^T$
	10000	7	0.125245	3.04E-05	$(1,1,...,1)^T$	14	0.241338	3.06E-05	$(-0.1809,...,-0.1809)^T$
15	10	6	3.04E-05	3.04E-05	$(2,2,...,2)^T$	14	0.023201	9.76E-05	$(2,2,...,2)^T$
	100	6	0.004215	2.59E-05	$(2,2,...,2)^T$	16	0.008327	9.54E-05	$(2,2,...,2)^T$
	1000	6	0.011554	8.20E-05	$(2,2,...,2)^T$	18	0.036078	9.33E-05	$(2,2,...,2)^T$
	10000	7	0.069841	2.60E-06	$(2,2,...,2)^T$	20	0.146703	9.13E-05	$(2,2,...,2)^T$
16	10	6	0.003141	1.23E-05	$(1,1,...,1)^T$	11	0.023251	9.94E-05	$(1,1,...,1)^T$
	100	6	0.004415	3.88E-05	$(1,1,...,1)^T$	12	0.006591	6.98E-07	$(1,1,...,1)^T$
	1000	7	0.014586	1.27E-06	$(1,1,...,1)^T$	12	0.025101	2.21E-06	$(1,1,...,1)^T$
	10000	7	0.083405	4.00E-06	$(1,1,...,1)^T$	12	0.112848	6.98E-06	$(1,1,...,1)^T$
17	10	13	0.088757	9.03E-05	$(8.0x10^{-6},...)^T$	31	0.141133	8.18E-05	$(1.863x10^{-5},...)^T$
	100	16	0.090698	8.37E-05	$(3.83x10^{-6},...)^T$	46	0.270288	7.74E-05	$(6.14x10^{-7},...)^T$
	1000	18	0.665361	7.55E-05	$(1.424x10^{-6},...)^T$	56	2.988004	9.44E-05	$(-2.11x10^{-6},...)^T$
	2000	20	2.369097	2.30E-05	$(-1.648x10^{-6},...)^T$	63	11.296114	9.98E-05	$(-1.44x10^{-6},...)^T$
18	10	6	0.004241	2.80E-05	$(3,3,...,3)^T$	9	0.005741	7.70E-05	$(3,3,...,3)^T$
	100	6	0.006511	8.86E-05	$(3,3,...,3)^T$	13	0.013249	7.96E-05	$(3,3,...,3)^T$
	1000	7	0.026646	2.81E-06	$(3,3,...,3)^T$	14	0.039753	6.28E-05	$(3,3,...,3)^T$
	10000	7	0.143298	8.88E-06	$(3,3,...,3)^T$	15	2.97E-01	2.63E-05	$(3,3,...,3)^T$
19	10	13	0.009969	4.54E-05	$(1,1,...,1)^T$	64	0.055103	7.85E-05	$(1,1,...,1)^T$
	100	13	0.017874	4.75E-05	$(1,1,...,1)^T$	77	0.088946	8.44E-05	$(1,1,...,1)^T$
	1000	13	0.068931	4.36E-05	$(1,1,...,1)^T$	70	0.362809	6.84E-05	$(1,1,...,1)^T$
	10000	12	0.357745	2.23E-05	$(1,1,...,1)^T$	73	3.383924	8.73E-05	$(1,1,...,1)^T$
20	10	6	0.006427	1.95E-06	$(1,1,...,1)^T$	15	0.010447	4.50E-05	$(1,1,...,1)^T$
	100	6	0.007574	6.16E-05	$(1,1,...,1)^T$	16	0.016103	7.55E-05	$(1,1,...,1)^T$
	1000	6	0.39667	1.95E-06	$(1,1,...,1)^T$	18	0.060695	5.38E-05	$(1,1,...,1)^T$
	10000	6	0.172924	6.16E-06	$(1,1,...,1)^T$	19	0.251278	7.06E-05	$(1,1,...,1)^T$

From Table 4.1, problems 1 and 2 indicate that the IDFDD and IPRP methods converge to different solutions due to non linearity. However the number of iteration and CPU time are less in our method than the compared method, even though is inconclusive.
From problems 3, 4, 5, 6, 7, 15, 16, 18, 19 and 20 we can easily see that the methods converge to the same solution. Indeed, the number of iteration and CPU time are less in our method than the compared method. Furthermore IPRP method fails in problem 4 with 1000 and 10000 dimensions.
Also in problem 8 and 14, IDFDD method converge to exact solution while the IPRP method is not, due to the round-off error. The IPRP method is however fails in problem 8 with 10000 dimension.
Despite the methods fails to reach the exact solution solution due to the round-off error in problems 9, 10, 11, 13 and 17, but the results are very close to the exact solutions. However, from problems 9, 10, 11, 13 and 17 our method is better than IPRP method in terms of number of iteration and CPU time. Moreover, IPRP method fails in problem 10 with 100, 1000 and 10000 dimensions respectively. In problem 12, the IPRP method fails completely.
Finally, on the average, $\|F(x_k)\|$ in our method is too small which signifies that the solution obtained is the true approximation of the exact solution compared to IPRP.
The exact solutions are reported in appendix A.

Table 4.2: Numerical Result for DSDF and DFCG of problems 1-20.

.

		DSDF				DFCG			
Problems	Dimension	NI	CPU time	$\|F(x_k)\|$	x^{*A}	NI	CPU time	$\|F(x_k)\|$	x^{*B}
1	10	4	0.034432	6.66E-05	$(1,1,...,1)^T$	120	0.042402	5.71E-05	$(-1,-1,...,-1)^T$
	100	5	0.003343	1.48E-05	$(1,1,...,1)^T$	121	0.034315	1.16E-07	$(-1,-1,...,-1)^T$
	1000	5	0.005957	4.67E-05	$(1,1,...,1)^T$	121	0.091319	3.68E-07	$(-1,-1,...,-1)^T$
	10000	6	0.066025	1.03E-05	$(1,1,...,1)^T$	121	0.716909	1.16E-06	$(-1,-1,...,-1)^T$
2	10	5	0.006911	8.60E-05	$(1,1,...,1)^T$	11	0.002788	1.27E-06	$(1,1,...,1)^T$
	100	6	0.003672	1.90E-05	$(1,1,...,1)^T$	11	0.002631	4.03E-06	$(1,1,...,1)^T$
	1000	6	0.008853	6.02E-05	$(1,1,...,1)^T$	11	0.023145	1.27E-05	$(1,1,...,1)^T$
	10000	7	0.073986	1.33E-05	$(1,1,...,1)^T$	11	0.119609	4.03E-05	$(1,1,...,1)^T$
3	10	7	0.011164	1.25E-05	$(1,1,...,1)^T$	92	0.023998	9.41E-05	$(1,1,...,1)^T$
	100	7	0.002915	2.45E-05	$(1,1,...,1)^T$	8	0.008652	3.76E-05	$(1,1,...,1)^T$
	1000	9	0.012155	2.07E-05	$(1,1,...,1)^T$	7	0.017195	3.11E-05	$(1,1,...,1)^T$
	10000	10	0.117188	7.51E-05	$(1,1,...,1)^T$	7	0.114274	9.46E-09	$(1,1,...,1)^T$
4	10	6	0.006629	9.54E-05	$(1,1,...,1)^T$	8	0.004607	4.39E-07	$(1,1,...,1)^T$
	100	9	0.005025	1.02E-05	$(1,1,...,1)^T$	10	0.004759	7.27E-08	$(1,1,...,1)^T$
	1000	11	0.018676	3.07E-05	$(1,1,...,1)^T$	—	—	—	—
	10000	13	0.182149	1.44E-05	$(1,1,...,1)^T$	—	—	—	—
5	10	5	0.003904	1.96E-05	$(1,1,...,1)^T$	11	0.008832	1.86E-05	$(1,1,...,1)^T$
	100	5	0.002268	6.20E-05	$(1,1,...,1)^T$	11	0.012392	5.88E-05	$(1,1,...,1)^T$
	1000	6	0.012521	1.37E-05	$(1,1,...,1)^T$	12	0.044482	2.70E-07	$(1,1,...,1)^T$
	10000	6	0.061625	4.34E-05	$(1,1,...,1)^T$	12	0.148847	8.55E-07	$(1,1,...,1)^T$
6	10	7	0.003097	4.90E-05	$(1,1,...,1)^T$	—	—	—	—
	100	8	0.004975	1.09E-05	$(1,1,...,1)^T$	—	—	—	—
	1000	8	0.020999	3.44E-05	$(1,1,...,1)^T$	—	—	—	—
	10000	9	0.110584	7.62E-06	$(1,1,...,1)^T$	—	—	—	—
7	10	6	0.006946	2.83E-05	$(1,1,...,1)^T$	8	0.003563	1.62E-07	$(1,1,...,1)^T$
	100	6	0.001604	8.95E-05	$(1,1,...,1)^T$	8	0.007023	5.12E-07	$(1,1,...,1)^T$
	1000	7	0.011042	1.98E-05	$(1,1,...,1)^T$	8	0.014292	1.62E-06	$(1,1,...,1)^T$
	10000	7	0.092938	6.27E-05	$(1,1,...,1)^T$	8	0.098526	5.12E-06	$(1,1,...,1)^T$
8	10	6	0.009596	9.26E-05	$(2,2,...,2)^T$	15	0.006778	2.61E-07	$(-0.0714,-0.0714,...)^T$
	100	7	0.004639	8.28E-05	$(2,2,...,2)^T$	—	—	—	—
	1000	10	0.024365	7.05E-06	$(2,2,...,2)^T$	—	—	—	—
	10000	11	0.137378	4.79E-05	$(2,2,...,2)^T$	—	—	—	—
9	10	6	0.002818	7.62E-06	$(2.411x10^{-6},...)^T$	6	0.003853	3.53E-11	$(-3.722x10^{-12},...)^T$
	100	6	0.003047	2.41E-05	$(2.411x10^{-6},...)^T$	6	0.003912	1.12E-10	$(-3.722x10^{-12},...)^T$
	1000	6	0.011982	7.62E-05	$(2.411x10^{-6},...)^T$	6	0.012268	3.53E-10	$(-3.722x10^{-12},...)^T$
	10000	7	0.067247	1.69E-05	$(1.688x10^{-6},...)^T$	6	0.085342	1.12E-09	$(-3.722x10^{-12},...)^T$
10	10	5	0.004807	4.46E-05	$(-0.0038,...,-0.0038)^T$	22	0.013748	3.97E-05	$(-0.0993,...,-0.0993)^T$
	100	14	0.009877	5.16E-05	$(0.0021,...,0.0021)^T$	28	0.029504	7.52E-05	$(-0.0107,...,-0.0107)^T$
	1000	20	0.055307	8.76E-05	$(0.0011,...,0.0011)^T$	127	0.459178	9.33E-05	$(-0.0019,...,-0.0019)^T$
	10000	25	0.453899	6.32E-05	$(0.3676,...,0.3676)^T$	—	—	—	—

Table 4.2: Continued.

		DSDF				DFCG			
Problems	Dimension	NI	CPU time	$\|F(x_k)\|$	x^{*A}	NI	CPU time	$\|F(x_k)\|$	x^{*B}
11	10	7	0.002218	1.15E-05	$(-3.630x10^{-6},...)^T$	9	0.005293	1.66E-06	$(5.238x10^{-7},...)^T$
	100	7	0.003184	3.63E-05	$(-3.630x10^{-6},...)^T$	9	0.004895	5.24E-06	$(5.238x10^{-7},...)^T$
	1000	8	0.011968	8.03E-06	$(-2.540x10^{-7},...)^T$	9	0.018659	1.66E-05	$(5.238x10^{-7},...)^T$
	10000	8	0.072248	2.54E-05	$(-2.540x10^{-7},...)^T$	9	0.097302	5.24E-05	$(5.238x10^{-7},...)^T$
12	10	19	0.008114	6.95E-05	$(-0.0047,...,-0.0047)^T$	27	0.004954	8.17E-05	$(0.0051,0.0051,...)^T$
	100	20	0.012999	8.92E-05	$(-0.0030,...,-0.0030)^T$	32	0.016539	9.08E-05	$(0.0030,...,0.0030)^T$
	1000	25	0.051278	8.74E-05	$(-0.0017,...,-0.0017)^T$	42	0.091716	9.74E-05	$(0.0018,...,0.0018)^T$
	10000	38	0.687416	9.85E-05	$(-0.9931,...,-0.9931)^T$	85	1.798643	9.87E-05	$(0.9929,...,0.9929)^T$
13	10	5	0.004339	6.82E-05	$(-1.960x10^{-6},...)^T$	8	0.003856	6.12E-07	$(-1.759x10^{-8},...)^T$
	100	6	0.004903	1.51E-05	$(-1.372x10^{-7},...)^T$	8	0.01206	1.93E-06	$(-1.759x10^{-8},...)^T$
	1000	6	0.019495	4.77E-05	$(-1.372x10^{-7},...)^T$	8	0.03233	6.12E-06	$(-1.759x10^{-8},...)^T$
	10000	7	0.103044	1.06E-05	$(-9.604x10^{-7},...)^T$	8	0.146992	1.93E-05	$(-1.759x10^{-8},...)^T$
14	10	5	0.002504	3.97E-05	$(1,1,...,1)^T$	8	0.006078	8.01E-06	$(-0.1809,...,-0.1809)^T$
	100	6	0.003653	8.76E-06	$(1,1,...,1)^T$	8	0.010101	2.53E-05	$(-0.1809,...,-0.1809)^T$
	1000	6	0.014831	2.77E-05	$(1,1,...,1)^T$	8	0.024225	8.01E-05	$(-0.1809,...,-0.1809)^T$
	10000	6	0.084180	8.76E-05	$(1,1,...,1)^T$	9	0.164258	9.61E-08	$(-0.1809,...,-0.1809)^T$
15	10	5	0.003478	4.50E-05	$(2,2,...,2)^T$	—	—	—	—
	100	6	0.002801	9.97E-06	$(2,2,...,2)^T$	—	—	—	—
	1000	6	0.006913	3.15E-05	$(2,2,...,2)^T$	—	—	—	—
	10000	6	0.042496	9.97E-05	$(2,2,...,2)^T$	—	—	—	—
16	10	5	0.003867	7.09E-05	$(1,1,...,1)^T$	—	—	—	—
	100	6	0.002938	1.57E-05	$(1,1,...,1)^T$	—	—	—	—
	1000	6	0.009187	4.96E-05	$(1,1,...,1)^T$	—	—	—	—
	10000	7	0.060654	1.10E-05	$(1,1,...,1)^T$	—	—	—	—
17	10	16	0.080326	3.47E-05	$(-3.420x10^{-6},...)^T$	34	0.13956	8.37E-05	$(-1.650x10^{-6},...)^T$
	100	19	0.098504	3.01E-05	$(-1.480x10^{-6},...)^T$	8	0.189075	9.55E-05	$(3.200x10^{-7},...)^T$
	1000	22	0.585713	7.61E-05	$(-2.210x10^{-7},...)^T$	53	2.471807	8.72E-05	$(-1.230x10^{-6},...)^T$
	2000	19	1.798509	4.75E-05	$(1.654x10^{-6},...)^T$	54	8.348075	8.10E-05	$(-1.080x10^{-6},...)^T$
18	10	6	0.014386	9.78E-05	$(3,3,...,3)^T$	12	0.055757	8.29E-05	$(-2.4329,...,-2.4329)^T$
	100	7	0.005728	2.17E-05	$(3,3,...,3)^T$	19	0.08408	6.06E-05	$(-2.4329,...,-2.4329)^T$
	1000	7	0.019926	6.85E-05	$(3,3,...,3)^T$	12	0.543682	7.45E-05	$(-2.4329,...,-2.4329)^T$
	10000	8	0.107731	1.52E-05	$(3,3,...,3)^T$	12	10.470351	7.33E-05	$(-2.4329,...,-2.4329)^T$
19	10	13	0.008485	4.57E-05	$(1,1,...,1)^T$	53	0.042589	9.65E-05	$(1,1,...,1)^T$
	100	13	0.012498	4.82E-05	$(1,1,...,1)^T$	55	0.061376	9.23E-05	$(1,1,...,1)^T$
	1000	13	0.055979	2.88E-05	$(1,1,...,1)^T$	43	0.190928	8.71E-05	$(1,1,...,1)^T$
	10000	12	0.273322	6.55E-05	$(1,1,...,1)^T$	48	1.22334	8.69E-05	$(1,1,...,1)^T$
20	10	6	0.003595	7.27E-06	$(1,1,...,1)^T$	6	0.004266	2.40E-05	$(1,1,...,1)^T$
	100	6	0.003024	2.30E-05	$(1,1,...,1)^T$	6	0.009643	7.58E-05	$(1,1,...,1)^T$
	1000	6	0.027041	7.27E-05	$(1,1,...,1)^T$	7	0.031537	1.99E-07	$(1,1,...,1)^T$
	10000	7	0.122074	1.61E-05	$(1,1,...,1)^T$	7	0.147862	6.28E-07	$(1,1,...,1)^T$

From Table 4.2, problem 1 shows that the DSDF and DFCG methods converge to different solutions due to non linearity. However the number of iteration and CPU time are less in our method than the compared method, even though is inconclusive.
Notwithstanding,from problems 2, 3, 4, 5, 7, 19 and 20, the methods converge to the same solution. In addition, the numerical results indicate that the proposed method, (i.e DSDF) has minimum number of iterations and CPU time, compared to DFCG method. Furthermore DFCG method fails in problem 4 with 1000 and 10000 dimensions respectively.
Also in problem 8, 14 and 18 DSDF method converge to exact solution while the DFCG method is not, due to the round-off error. The compared method is however fails in problem 8 with dimensions 100, 1000 and 10000 respectively.
Although, the methods fails to reach the exact solution solution due to the round-off error in problems 9, 10, 11, 12, 13 and 17, but the results are very close to the exact solutions. Indeed, from problems 9, 10, 11, 12, 13 and 17 our method is better than DFCG method in terms of number of iteration and CPU time. Moreover, DFCG method fails in problem 10 with 10000 dimension. Additionally, DFCG method fails completely in problems 6, 15 and 16 respectively.
Finally, on the average, $||F(x_k)||$ in our method is too small which means that the solution obtained is the true approximation of the exact solution compared to DFCG.
The exact solutions are reported in appendix A.

Table 4.3: Numerical Result for TDS and SDYP of problems 1-20.

		TDS				SDYP			
Problems	Dimension	NI	CPU time	$\|F(x_k)\|$	x^{*A}	NI	CPU time	$\|F(x_k)\|$	x^{*B}
1	10	5	0.001947	7.68E-06	$(1,1,...,1)^T$	8	0.033746	5.00E-05	$(1,1,...,1)^T$
	100	5	0.003802	2.43E-05	$(1,1,...,1)^T$	9	0.014682	5.13E-05	$(1,1,...,1)^T$
	1000	5	0.020704	7.68E-05	$(1,1,...,1)^T$	10	0.026028	4.22E-05	$(1,1,...,1)^T$
	10000	6	0.089733	1.31E-06	$(1,1,...,1)^T$	11	0.118971	4.33E-05	$(1,1,...,1)^T$
2	10	6	0.056535	1.34E-06	$(1,1,...,1)^T$	8	0.003843	9.14E-05	$(1,1,...,1)^T$
	100	6	0.005429	4.24E-06	$(1,1,...,1)^T$	9	0.006876	9.39E-05	$(1,1,...,1)^T$
	1000	6	0.006568	1.34E-05	$(1,1,...,1)^T$	1	0.012182	2.00E-05	$(1,1,...,1)^T$
	10000	6	0.080516	4.24E-05	$(1,1,...,1)^T$	10	0.110503	6.33E-05	$(1,1,...,1)^T$
3	10	5	0.000979	5.81E-05	$(1,1,...,1)^T$	8	0.002437	3.73E-05	$(1,1,...,1)^T$
	100	5	0.003297	3.31E-05	$(1,1,...,1)^T$	11	0.011825	6.16E-06	$(1,1,...,1)^T$
	1000	5	0.010823	3.10E-05	$(1,1,...,1)^T$	14	0.039315	1.96E-05	$(1,1,...,1)^T$
	10000	6	0.084062	6.72E-06	$(1,1,...,1)^T$	15	0.401206	3.29E-05	$(1,1,...,1)^T$
4	10	6	0.006566	6.80E-07	$(1,1,...,1)^T$	12	0.006463	1.29E-05	$(1,1,...,1)^T$
	100	5	0.001785	4.24E-06	$(1,1,...,1)^T$	12	0.009415	7.31E-05	$(1,1,...,1)^T$
	1000	6	0.010399	1.36E-06	$(1,1,...,1)^T$	16	0.086401	3.54E-05	$(1,1,...,1)^T$
	10000	7	0.10759	1.00E-05	$(1,1,...,1)^T$	17	0.969384	4.39E-05	$(1,1,...,1)^T$
5	10	7	0.002951	1.67E-05	$(1,1,...,1)^T$	8	0.004845	1.79E-05	$(1,1,...,1)^T$
	100	7	0.004831	5.29E-05	$(1,1,...,1)^T$	8	0.007936	5.67E-05	$(1,1,...,1)^T$
	1000	8	0.013615	7.90E-07	$(1,1,...,1)^T$	9	0.027438	6.04E-05	$(1,1,...,1)^T$
	10000	8	0.096588	2.50E-06	$(1,1,...,1)^T$	10	0.114138	7.85E-05	$(1,1,...,1)^T$
6	10	6	0.005406	3.24E-06	$(1,1,...,1)^T$	—	—	—	—
	100	6	0.004293	1.02E-05	$(1,1,...,1)^T$	—	—	—	—
	1000	6	0.022208	3.24E-05	$(1,1,...,1)^T$	—	—	—	—
	10000	7	0.095078	4.73E-07	$(1,1,...,1)^T$	—	—	—	—
7	10	5	0.012472	2.13E-06	$(1,1,...,1)^T$	9	0.002708	9.24E-05	$(1,1,...,1)^T$
	100	5	0.003244	6.73E-06	$(1,1,...,1)^T$	11	0.003351	3.55E-05	$(1,1,...,1)^T$
	1000	5	0.021118	2.13E-05	$(1,1,...,1)^T$	12	0.020874	4.65E-05	$(1,1,...,1)^T$
	10000	5	0.064784	6.73E-05	$(1,1,...,1)^T$	13	0.154279	5.92E-05	$(1,1,...,1)^T$
8	10	7	0.005563	2.69E-06	$(2,2,...,2)^T$	10	0.006897	1.88E-05	$(2,2,...,2)^T$
	100	7	0.002055	4.77E-06	$(2,2,...,2)^T$	10	0.005654	6.56E-05	$(2,2,...,2)^T$
	1000	7	0.018814	7.88E-05	$(2,2,...,2)^T$	12	0.048087	7.68E-05	$(2,2,...,2)^T$
	10000	9	0.118304	6.08E-07	$(2,2,...,2)^T$	18	0.620556	9.46E-05	$(2,2,...,2)^T$
9	10	4	0.003177	8.12E-05	$(2.567x10^{-5},...)^T$	5	0.003032	3.85E-05	$(-4.062x10^{-6},...)^T$
	100	6	0.005008	1.53E-06	$(-5.083x10^{-8},...)^T$	6	0.002621	5.97E-05	$(-1.989x10^{-6},...)^T$
	1000	6	0.009973	4.82E-06	$(-5.083x10^{-8},...)^T$	7	0.007421	6.01E-05	$(-6.331x10^{-7},...)^T$
	10000	6	0.064697	1.53E-05	$(-5.083x10^{-8},...)^T$	8	0.07514	3.80E-05	$(-1.267x10^{-7},...)^T$
10	10	10	0.003904	4.53E-05	$(0.0037,...,0.0037)^T$	8	0.006115	6.58E-05	$(-0.0994,...,-0.0994)^T$
	100	13	0.011268	4.84E-05	$(-0.0104,...,-0.0104)^T$	—	—	—	—
	1000	19	0.045298	8.51E-05	$(0.0011,...,0.0011)^T$	—	—	—	—
	10000	23	0.394541	8.07E-05	$(-0.4681,...,-0.4681)^T$	—	—	—	—

Table 4.3:Continued.

		TDS				SDYP			
Problems	Dimension	NI	CPU time	$\|F(x_k)\|$	x^{*A}	NI	CPU time	$\|F(x_k)\|$	x^{*B}
11	10	4	0.001752	1.83E-06	$(-5.775x10^{-7},...)^T$	6	0.002959	9.05E-05	$(2.861x10^{-5},...)^T$
	100	4	0.002128	5.78E-06	$(-5.775x10^{-7},...)^T$	8	0.002045	4.35E-05	$(4.353x10^{-6},...)^T$
	1000	4	0.007951	1.83E-05	$(-5.775x10^{-7},...)^T$	9	0.008723	5.38E-05	$(1.701x10^{-6},...)^T$
	10000	4	0.052802	5.78E-05	$(-5.775x10^{-7},...)^T$	10	0.083048	6.70E-05	$(6.696x10^{-6},...)^T$
12	10	19	0.006502	5.47E-05	$(0.0041,...,0.0041)^T$	—	—	—	—
	100	20	0.010267	6.60E-05	$(0.0026,...,0.0026)^T$	—	—	—	—
	1000	24	0.036764	9.73E-05	$(0.0081,...,0.0081)^T$	—	—	—	—
	10000	48	0.729827	1.00E-04	$(0.9992,...,0.9992)^T$	—	—	—	—
13	10	5	0.002217	4.12E-06	$(1.184x10^{-7},...)^T$	11	0.003377	3.64E-05	$(1.046x10^{-6},...)^T$
	100	5	0.003283	1.30E-05	$(1.184x10^{-7},...)^T$	12	0.004785	5.75E-05	$(5.231x10^{-7},...)^T$
	1000	5	0.021946	4.12E-05	$(1.184x10^{-7},...)^T$	13	0.02714	7.88E-05	$(2.264x10^{-7},...)^T$
	10000	6	0.083273	6.71E-07	$(-6.099x10^{-7},...)^T$	15	0.26382	2.57E-05	$(2.334x10^{-7},...)^T$
14	10	6	0.003982	8.82E-07	$(-0.1809,...,-0.1809)^T$	—	—	—	—
	100	6	0.006426	2.79E-06	$(-0.1809,...,-0.1809)^T$	—	—	—	—
	1000	6	0.022915	8.82E-06	$(-0.1809,...,-0.1809)^T$	—	—	—	—
	10000	6	0.081652	2.79E-05	$(-0.1809,...,-0.1809)^T$	—	—	—	—
15	10	7	0.002167	1.26E-06	$(2,2,...,2)^T$	13	0.003336	4.14E-05	$(2,2,...,2)^T$
	100	7	0.003098	3.98E-06	$(2,2,...,2)^T$	14	0.007815	5.37E-05	$(2,2,...,2)^T$
	1000	7	0.007226	1.26E-05	$(2,2,...,2)^T$	15	0.016273	6.83E-05	$(2,2,...,2)^T$
	10000	7	0.053145	3.98E-05	$(2,2,...,2)^T$	16	0.097079	8.93E-05	$(2,2,...,2)^T$
16	10	5	0.006786	2.76E-06	$(1,1,...,1)^T$	9	0.002488	7.45E-05	$(1,1,...,1)^T$
	100	5	0.002514	8.73E-06	$(1,1,...,1)^T$	10	0.004225	6.66E-05	$(1,1,...,1)^T$
	1000	5	0.012019	2.76E-05	$(1,1,...,1)^T$	11	0.014426	6.39E-05	$(1,1,...,1)^T$
	10000	5	0.047607	8.73E-05	$(1,1,...,1)^T$	12	0.098096	5.71E-05	$(1,1,...,1)^T$
17	10	14	0.059241	2.79E-05	$(1.184x10^{-6},...)^T$	32	0.176467	8.12E-05	$(7.34x10^{-6},...)^T$
	100	15	0.071557	1.49E-05	$(-1.341x10^{-6},...)^T$	31	0.218382	8.65E-05	$(6.7x10^{-6},...)^T$
	1000	16	0.490706	9.60E-05	$(-7.21x10^{-7},...)^T$	46	2.683304	9.94E-05	$(3.581x10^{-6},...)^T$
	2000	15	1.473685	8.21E-05	$(4.267x10^{-6},...)^T$	35	6.782476	8.96E-05	$(4.165x10^{-6},...)^T$
18	10	5	0.003728	1.23E-05	$(3,3,...,3)^T$	10	0.005918	7.12E-05	$(3,3,...,3)^T$
	100	5	0.003541	3.90E-05	$(3,3,...,3)^T$	12	0.010502	4.57E-05	$(3,3,...,3)^T$
	1000	6	0.016985	6.18E-07	$(3,3,...,3)^T$	13	0.034412	9.71E-06	$(3,3,...,3)^T$
	10000	6	0.088663	1.95E-06	$(3,3,...,3)^T$	13	2.53E-01	3.07E-05	$(3,3,...,3)^T$
19	10	13	0.002563	9.61E-05	$(1,1,...,1)^T$	25	0.015077	9.19E-05	$(1,1,...,1)^T$
	100	14	0.011541	6.54E-05	$(1,1,...,1)^T$	22	0.031304	5.38E-05	$(1,1,...,1)^T$
	1000	12	0.027394	2.36E-05	$(1,1,...,1)^T$	21	0.106358	6.23E-05	$(1,1,...,1)^T$
	10000	11	0.221293	6.89E-05	$(1,1,...,1)^T$	29	1.194646	8.29E-05	$(1,1,...,1)^T$
20	10	3	0.000693	2.69E-05	$(1,1,...,1)^T$	9	0.001701	4.71E-05	$(1,1,...,1)^T$
	100	3	0.001153	8.52E-05	$(1,1,...,1)^T$	10	0.004271	5.26E-05	$(1,1,...,1)^T$
	1000	4	0.020554	1.90E-06	$(1,1,...,1)^T$	11	0.020028	3.78E-05	$(1,1,...,1)^T$
	10000	4	0.067597	6.02E-06	$(1,1,...,1)^T$	12	0.148869	4.22E-05	$(1,1,...,1)^T$

From Table 4.3, in problems 1, 2, 3, 4, 5, 7, 8, 15, 16, 18, 19 and 20, the methods converge to the same solution. Even so, the numerical results indicate that the proposed method, (i.e TDS) has minimum number of iterations and CPU time, compared to SDYP method.
However, in problem 8, 14 and 18 TDS method converge to exact solution while the SDYP method is not, due to the round-off error. The compared method is however fails in problem 8 with dimensions 100, 1000 and 10000 respectively.
Furthermore, in problems 9, 10, 11, 13 and 17 the methods fails to reach the exact solution solution due to the round-off error, but the results are very close to the exact solutions. However, from problems 9, 10, 11, 13 and 17 our method is better than SDYP method in terms of number of iteration and CPU time. Moreover, SDYP method fails in problem 10 with dimensions 100, 1000 and 10000 respectively. Nevertheless, SDYP method fails completely in problems 6, 12 and 14 respectively.
Finally, on the average, $||F(x_k)||$ in our method is too small which means that the solution obtained is the true approximation of the exact solution compared to SDYP.
The exact solutions are reported in appendix A.

Table 4.4: IDFDD algorithm for problems 1, 6, 9 and 15 with different Initial points.

			IDFDD			
Problems	Dimension	Initial points	NI	CPU time	$\|F(x_k)\|$	x^{*A}
1	10	$2x10^8$	45	0.021513	5.53E-05	$(1,1,...,1)^T$
	100	$2x10^8$	46	0.022959	1.58E-06	$(1,1,...,1)^T$
	1000	$2x10^8$	46	0.080024	5.01E-06	$(1,1,...,1)^T$
	10000	$2x10^8$	46	0.325314	1.58E-05	$(1,1,...,1)^T$
1	10	$3x10^{12}$	64	0.023367	1.01E-05	$(-1,-1,...,-1)^T$
	100	$3x10^{12}$	64	0.027017	3.21E-05	$(-1,-1,...,-1)^T$
	1000	$3x10^{12}$	65	0.088216	9.89E-07	$(-1,-1,...,-1)^T$
	10000	$3x10^{12}$	65	0.385467	3.13E-06	$(-1,-1,...,-1)^T$
1	10	$(-4)x10^{20}$	102	0.042763	4.53E-05	$(1,1,...,1)^T$
	100	$(-4)x10^{20}$	103	0.045667	1.30E-06	$(1,1,...,1)^T$
	1000	$(-4)x10^{20}$	103	0.094779	4.12E-06	$(1,1,...,1)^T$
	10000	$(-4)x10^{20}$	103	0.596261	1.30E-05	$(1,1,...,1)^T$
6	10	$2x10^8$	47	2.28E-02	1.64E-06	$(2,2,...,2)^T$
	100	$2x10^8$	47	0.020724	5.18E-06	$(2,2,...,2)^T$
	1000	$2x10^8$	47	0.180902	1.64E-05	$(2,2,...,2)^T$
	10000	$2x10^8$	47	0.672407	5.18E-05	$(2,2,...,2)^T$
6	10	$3x10^{12}$	65	0.030165	3.11E-05	$(1,1,...,1)^T$
	100	$3x10^{12}$	65	0.048648	9.82E-05	$(1,1,...,1)^T$
	1000	$3x10^{12}$	66	0.145873	2.70E-06	$(1,1,...,1)^T$
	10000	$3x10^{12}$	66	0.837567	8.54E-06	$(1,1,...,1)^T$
6	10	$(-4)x10^{20}$	104	0.042548	1.53E-06	$(1,1,...,1)^T$
	100	$(-4)x10^{20}$	104	0.063996	4.83E-06	$(1,1,...,1)^T$
	1000	$(-4)x10^{20}$	104	0.156617	1.53E-05	$(1,1,...,1)^T$
	10000	$(-4)x10^{20}$	104	1.248241	4.83E-05	$(1,1,...,1)^T$
9	10	$2x10^8$	9	0.006881	8.22E-05	$(2.601X10^{-5},...)^T$
	100	$2x10^8$	10	0.015525	7.78E-06	$(-2.593X10^{-7},...)^T$
	1000	$2x10^8$	10	0.078372	2.46E-05	$(-2.593X10^{-7},...)^T$
	10000	$2x10^8$	10	0.299733	7.78E-05	$(-2.593X10^{-7},...)^T$
9	10	$3x10^{12}$	11	0.007541	1.22E-05	$(3.845X10^{-6},...)^T$
	100	$3x10^{12}$	11	0.019144	3.84E-05	$(3.845X10^{-6},...)^T$
	1000	$3x10^{12}$	12	0.073349	3.64E-06	$(-3.838X10^{-6},...)^T$
	10000	$3x10^{12}$	12	0.343572	1.15E-05	$(-3.838X10^{-6},...)^T$
9	10	$(-4)x10^{20}$	17	0.007253	2.62E-05	$(2.765X10^{-6},...)^T$
	100	$(-4)x10^{20}$	17	0.010059	8.29E-05	$(2.765X10^{-6},...)^T$
	1000	$(-4)x10^{20}$	19	0.049788	6.83E-05	$(2.160X10^{-6},...)^T$
	10000	$(-4)x10^{20}$	20	0.464201	6.48E-06	$(-2.160X10^{-6},...)^T$
15	10	$2x10^8$	44	0.011376	3.88E-05	$(2,2,...,2)^T$
	100	$2x10^8$	45	0.027331	1.21E-06	$(2,2,...,2)^T$
	1000	$2x10^8$	45	0.051583	3.82E-06	$(2,2,...,2)^T$
	10000	$2x10^8$	45	0.286555	1.21E-05	$(2,2,...,2)^T$
15	10	$3x10^{12}$	63	0.009233	1.10E-05	$(-2,-2,...,-2)^T$
	100	$3x10^{12}$	63	0.012168	3.47E-05	$(-2,-2,...,-2)^T$
	1000	$3x10^{12}$	64	0.056415	1.10E-06	$(-2,-2,...,-2)^T$
	10000	$3x10^{12}$	64	0.360139	3.46E-06	$(-2,-2,...,-2)^T$
15	10	$(-4)x10^{20}$	101	0.027604	3.75E-05	$(-2,-2,...,-2)^T$
	100	$(-4)x10^{20}$	102	0.020796	1.17E-06	$(-2,-2,...,-2)^T$
	1000	$(-4)x10^{20}$	102	0.064015	3.70E-06	$(-2,-2,...,-2)^T$
	10000	$(-4)x10^{20}$	102	0.538615	1.17E-05	$(-2,-2,...,-2)^T$

Table 4.5: DSDF algorithm for problems 1, 6, 9 and 15 with different Initial points.

			DSDF			
Problems	Dimension	Initial points	NI	CPU time	$\|F(x_k)\|$	x^{*A}
1	10	$2x10^8$	46	0.008618	7.00E-05	$(-1,-1,...,-1)^T$
	100	$2x10^8$	47	0.016264	1.55E-05	$(-1,-1,...,-1)^T$
	1000	$2x10^8$	47	0.029034	4.91E-05	$(-1,-1,...,-1)^T$
	10000	$2x10^8$	48	0.210159	1.09E-05	$(-1,-1,...,-1)^T$
1	10	$3x10^{12}$	70	0.013752	4.55E-05	$(-1,-1,...,-1)^T$
	100	$3x10^{12}$	71	0.012343	1.01E-05	$(-1,-1,...,-1)^T$
	1000	$3x10^{12}$	71	0.046831	3.19E-05	$(-1,-1,...,-1)^T$
	10000	$3x10^{12}$	72	0.277617	7.06E-06	$(-1,-1,...,-1)^T$
1	10	$(-4)x10^{20}$	112	0.015536	1.08E-05	$(-1,-1,...,-1)^T$
	100	$(-4)x10^{20}$	112	0.022725	3.41E-05	$(-1,-1,...,-1)^T$
	1000	$(-4)x10^{20}$	113	0.055993	7.55E-06	$(-1,-1,...,-1)^T$
	10000	$(-4)x10^{20}$	113	0.411928	2.39E-05	$(-1,-1,...,-1)^T$
6	10	$2x10^8$	47	0.008120	7.36E-05	$(1,1,...,1)^T$
	100	$2x10^8$	48	0.017502	1.64E-05	$(1,1,...,1)^T$
	1000	$2x10^8$	48	0.055036	5.17E-05	$(1,1,...,1)^T$
	10000	$2x10^8$	49	0.422352	1.15E-05	$(1,1,...,1)^T$
6	10	$3x10^{12}$	71	0.011287	4.79E-05	$(1,1,...,1)^T$
	100	$3x10^{12}$	72	0.019641	1.06E-05	$(1,1,...,1)^T$
	1000	$3x10^{12}$	72	0.076576	3.36E-05	$(1,1,...,1)^T$
	10000	$3x10^{12}$	73	0.599059	7.45E-06	$(1,1,...,1)^T$
6	10	$(-4)x10^{20}$	113	0.012437	1.14E-05	$(1,1,...,1)^T$
	100	$(-4)x10^{20}$	113	0.029742	3.60E-05	$(1,1,...,1)^T$
	1000	$(-4)x10^{20}$	114	0.107006	7.97E-06	$(1,1,...,1)^T$
	10000	$(-4)x10^{20}$	114	0.901013	2.52E-05	$(1,1,...,1)^T$
9	10	$2x10^8$	14	0.005992	2.57E-05	$(8.118X10^{-6},...)^T$
	100	$2x10^8$	14	0.005961	8.12E-05	$(8.118X10^{-6},...)^T$
	1000	$2x10^8$	15	0.027872	1.80E-05	$(5.682X10^{-7},...)^T$
	10000	$2x10^8$	15	0.203518	5.68E-05	$(5.682X10^{-7},...)^T$
9	10	$3x10^{12}$	17	0.006249	2.00E-05	$(6.338X10^{-6},...)^T$
	100	$3x10^{12}$	17	0.008417	6.34E-05	$(6.338X10^{-6},...)^T$
	1000	$3x10^{12}$	18	0.031213	1.40E-05	$(4.437X10^{-7},...)^T$
	10000	$3x10^{12}$	18	0.281194	4.44E-05	$(4.437X10^{-7},...)^T$
9	10	$(-4)x10^{20}$	25	0.008456	7.51E-06	$(2.376X10^{-6},...)^T$
	100	$(-4)x10^{20}$	25	0.021883	2.38E-05	$(2.376X10^{-6},...)^T$
	1000	$(-4)x10^{20}$	25	0.089905	7.51E-05	$(2.376X10^{-6},...)^T$
	10000	$(-4)x10^{20}$	26	0.492803	1.66E-05	$(1.663X10^{-6},...)^T$
15	10	$2x10^8$	45	0.005796	6.64E-05	$(-2,-2,...,-2)^T$
	100	$2x10^8$	46	0.016889	1.47E-05	$(-2,-2,...,-2)^T$
	1000	$2x10^8$	46	0.046618	4.65E-05	$(-2,-2,...,-2)^T$
	10000	$2x10^8$	47	0.233606	1.03E-05	$(-2,-2,...,-2)^T$
15	10	$3x10^{12}$	69	0.017915	4.32E-05	$(-2,-2,...,-2)^T$
	100	$3x10^{12}$	70	0.020344	9.56E-06	$(-2,-2,...,-2)^T$
	1000	$3x10^{12}$	70	0.062932	3.02E-05	$(-2,-2,...,-2)^T$
	10000	$3x10^{12}$	70	0.326998	9.56E-05	$(-2,-2,...,-2)^T$
15	10	$(-4)x10^{20}$	111	0.033687	1.02E-05	$(-2,-2,...,-2)^T$
	100	$(-4)x10^{20}$	111	0.026974	3.23E-05	$(-2,-2,...,-2)^T$
	1000	$(-4)x10^{20}$	112	0.093523	7.16E-06	$(-2,-2,...,-2)^T$
	10000	$(-4)x10^{20}$	112	0.408447	2.26E-05	$(-2,-2,...,-2)^T$

Table 4.6: TDS algorithm for problems 1, 6, 9 and 15 with different Initial points.

			TDS			
Problems	Dimension	Initial points	NI	CPU time	$\|F(x_k)\|$	x^{*A}
1	10	$2x10^8$	43	0.009243	9.42E-07	$(1,1,...,1)^T$
	100	$2x10^8$	43	0.011546	2.98E-06	$(1,1,...,1)^T$
	1000	$2x10^8$	43	0.044272	9.42E-06	$(1,1,...,1)^T$
	10000	$2x10^8$	43	0.194454	2.98E-05	$(1,1,...,1)^T$
1	10	$3x10^{12}$	63	0.012183	1.01E-06	$(1,1,...,1)^T$
	100	$3x10^{12}$	63	0.024787	3.19E-06	$(1,1,...,1)^T$
	1000	$3x10^{12}$	63	0.068797	1.01E-05	$(1,1,...,1)^T$
	10000	$3x10^{12}$	63	0.266476	3.19E-05	$(1,1,...,1)^T$
1	10	$(-4)x10^{20}$	103	0.019983	1.64E-06	$(-1,-1,...,-1)^T$
	100	$(-4)x10^{20}$	103	0.035487	5.20E-06	$(-1,-1,...,-1)^T$
	1000	$(-4)x10^{20}$	103	0.085952	1.64E-05	$(-1,-1,...,-1)^T$
	10000	$(-4)x10^{20}$	103	0.361002	5.20E-05	$(-1,-1,...,-1)^T$
6	10	$2x10^8$	44	0.010007	2.24E-07	$(2,2,...,2)^T$
	100	$2x10^8$	44	0.021101	7.10E-07	$(2,2,...,2)^T$
	1000	$2x10^8$	44	0.073591	2.24E-06	$(2,2,...,2)^T$
	10000	$2x10^8$	44	0.374183	7.10E-06	$(2,2,...,2)^T$
6	10	$3x10^{12}$	64	0.016068	3.85E-07	$(2,2,...,2)^T$
	100	$3x10^{12}$	64	0.030709	1.22E-06	$(2,2,...,2)^T$
	1000	$3x10^{12}$	64	0.091592	3.85E-06	$(2,2,...,2)^T$
	10000	$3x10^{12}$	64	0.499028	1.22E-05	$(2,2,...,2)^T$
6	10	$(-4)x10^{20}$	104	0.022649	1.10E-05	$(1,1,...,1)^T$
	100	$(-4)x10^{20}$	104	0.048804	3.48E-05	$(1,1,...,1)^T$
	1000	$(-4)x10^{20}$	105	0.104202	4.83E-07	$(1,1,...,1)^T$
	10000	$(-4)x10^{20}$	105	0.790452	1.53E-06	$(1,1,...,1)^T$
9	10	$2x10^8$	8	0.002741	1.12E-05	$(-1.186X10^{-6},...)^T$
	100	$2x10^8$	8	0.007295	3.56E-05	$(-1.186X10^{-6},...)^T$
	1000	$2x10^8$	9	0.046912	7.55E-05	$(2.387X10^{-6},...)^T$
	10000	$2x10^8$	10	0.194614	9.45E-05	$(9.450X10^{-6},...)^T$
9	10	$3x10^{12}$	13	0.005075	5.46E-06	$(-5.756X10^{-7},...)^T$
	100	$3x10^{12}$	13	0.008163	1.73E-05	$(-5.756X10^{-7},...)^T$
	1000	$3x10^{12}$	13	0.053262	5.46E-05	$(-5.756X10^{-7},...)^T$
	10000	$3x10^{12}$	15	0.271377	4.55E-05	$(4.549X10^{-7},...)^T$
9	10	$(-4)x10^{20}$	16	0.005913	5.02E-06	$(-5.288X10^{-7},...)^T$
	100	$(-4)x10^{20}$	16	0.015446	1.59E-05	$(-5.288X10^{-7},...)^T$
	1000	$(-4)x10^{20}$	16	0.069115	5.02E-05	$(-5.288X10^{-7},...)^T$
	10000	$(-4)x10^{20}$	18	0.325522	4.18E-05	$(4.180X10^{-7},...)^T$
15	10	$2x10^8$	41	0.014528	4.08E-05	$(2,2,...,2)^T$
	100	$2x10^8$	42	0.018187	6.03E-07	$(2,2,...,2)^T$
	1000	$2x10^8$	42	0.033786	1.91E-06	$(2,2,...,2)^T$
	10000	$2x10^8$	42	0.187819	6.03E-06	$(2,2,...,2)^T$
15	10	$3x10^{12}$	61	0.018482	4.87E-05	$(2,2,...,2)^T$
	100	$3x10^{12}$	62	0.019051	7.16E-07	$(2,2,...,2)^T$
	1000	$3x10^{12}$	62	0.056192	2.26E-06	$(2,2,...,2)^T$
	10000	$3x10^{12}$	62	0.257251	7.16E-06	$(2,2,...,2)^T$
15	10	$(-4)x10^{20}$	102	0.031476	1.83E-06	$(-2,-2,...,-2)^T$
	100	$(-4)x10^{20}$	102	0.032243	5.78E-06	$(-2,-2,...,-2)^T$
	1000	$(-4)x10^{20}$	102	0.071718	1.83E-05	$(-2,-2,...,-2)^T$
	10000	$(-4)x10^{20}$	102	0.376826	5.78E-05	$(-2,-2,...,-2)^T$

From Tables 4.4-4.6, we report the behavior of the IDFDD , DSDF and TDS algorithms for Functions 1, 6, 9 and 15 in order to illustrate the global convergence. Though, we take three different initial points (i.e $2x10^8$, $3x10^{12}$ and $-4x10^{20}$) which are far away from the solutions and efficiently converge to the exact solutions, although it requires more iterations when the initial guess is further away from the solution. Indeed, it is very competitive for large-scale problems. Despite the methods did not converge to the exact solution in problem 9 due to the round-off error, but the results are very close to the exact solution. However, IDFDD , DSDF and TDS iterations are very inexpensive and that explains the advantages in CPU time. This clarify that our three propose methods are globally convergent.

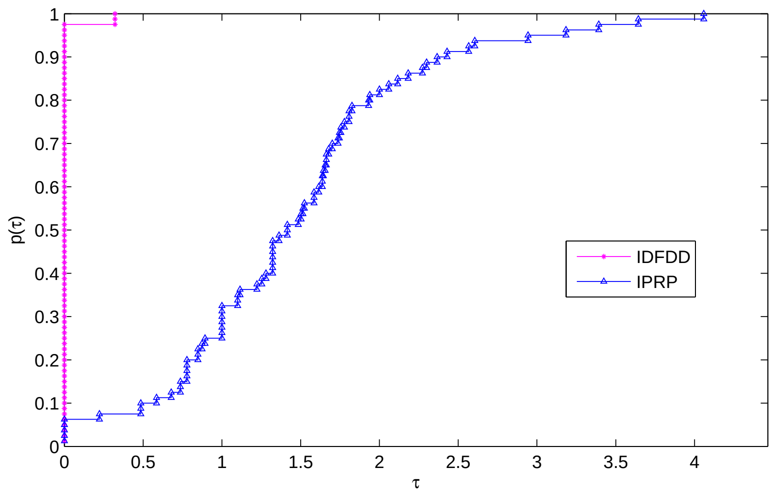

Figure 4.1: Performance profile of IDFDD and IPRP methods with respect to the number of iteration for the problems 1-20.

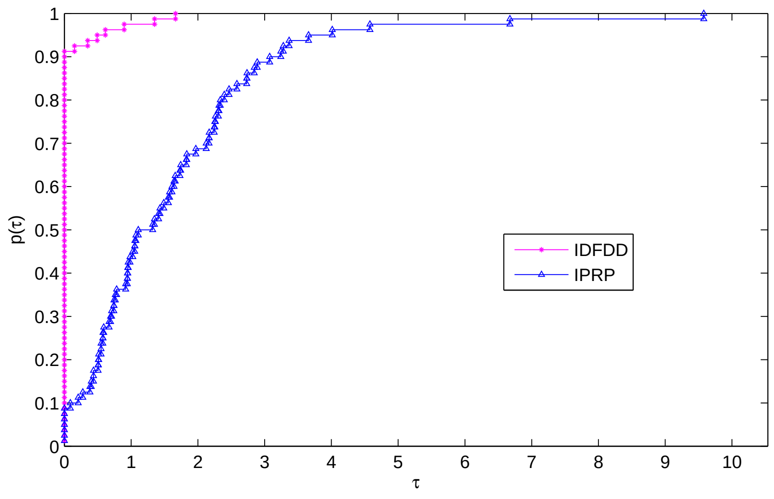

Figure 4.2: Performance profile of IDFDD and IPRP methods with respect to the CPU time (in second) for the problems 1-20.

we plotted Figures 4.1 and 4.2 according to the data in Tables 4.1 by using the performance profiles of Dolan and Mor$\acute{e}$ (2002).
From Figures 4.1 and 4.2, the performance profiles of IDFDD and IPRP method are compared with respect to the number of iterates and CPU time (in second) respectively. it is easy to see that the performance of IDFDD method, in most cases, is better than that of IPRP method, which show that IDFDD method grows up faster than IPRP method and so reaches to 1 before it. Thus, from the technique of Dolan and Mor$\acute{e}$ (2002), the performance curves in Figures 4.1-4.2, for the number of iterates and CPU time (in second), demonstrate that the better results are achieved by IDFDD method.

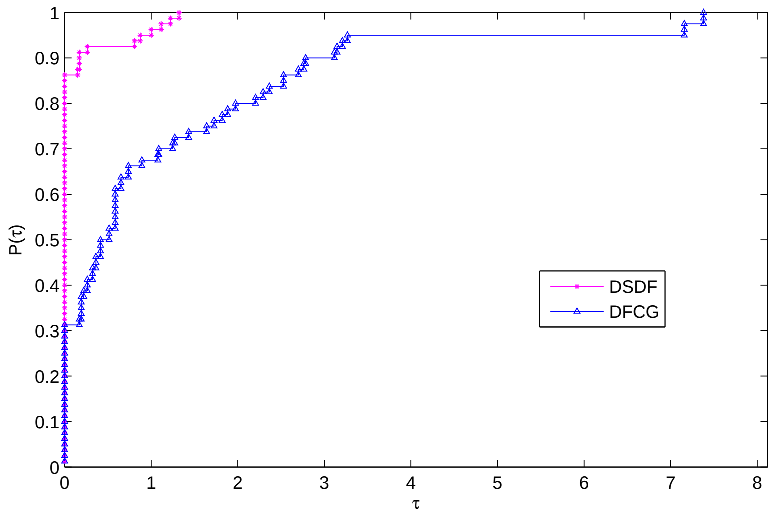

Figure 4.3: Performance profile of DSDF and DFCG methods with respect to the number of iteration for the problems 1-20.

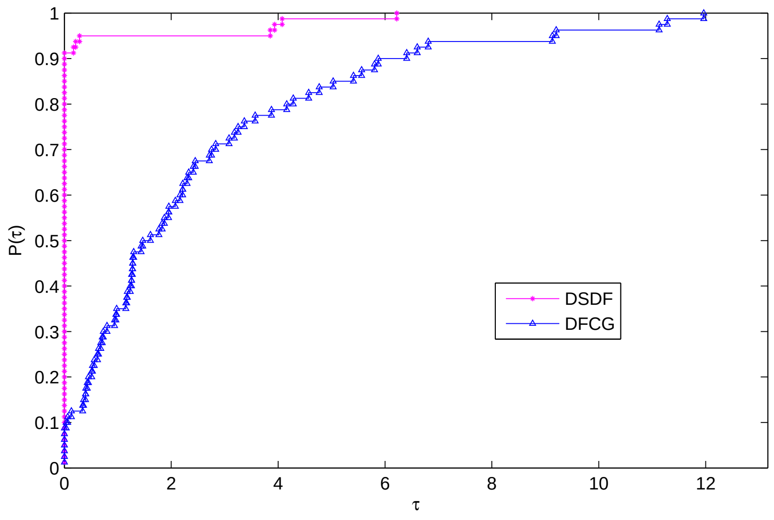

Figure 4.4: Performance profile of DSDF and DFCG methods with respect to the CPU time (in second) for the problems 1-20.

we plotted Figures 4.3 and 4.4 according to the data in Tables 4.2 by using the performance profiles of Dolan and Mor*é* (2002).
From Figures 4.3-4.4, the performance profiles of DSDF and DFCG method are compared with respect to the number of iterates and CPU time (in second) respectively. it is easy to see that the performance of DSDF method, in most cases, is better than that of DFCG method, which show that DSDF method grows up faster than DFCG method and so reaches to 1 before it. Thus, from the technique of Dolan and Mor*é* (2002), the performance curves in Figures. 4.3-4.4, for the number of iterates and CPU time (in second), demonstrate that the better results are achieved by DSDF method.

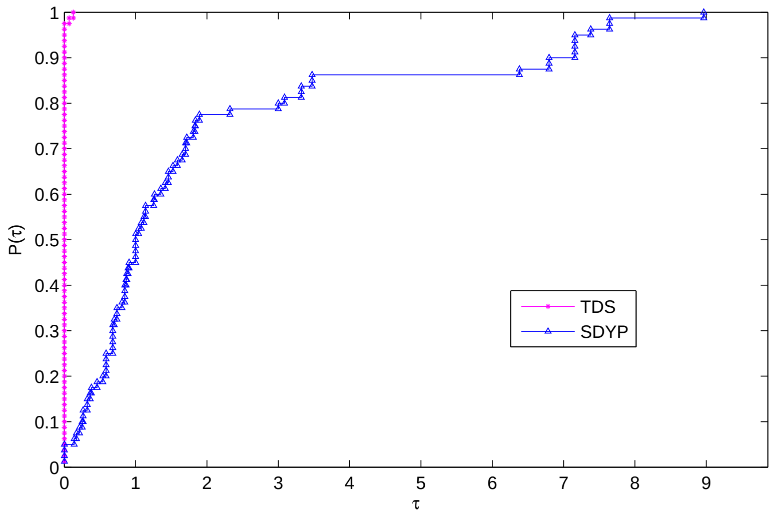

Figure 4.5: Performance profile of TDS and SDYP methods with respect to the number of iteration for the problems 1-20.

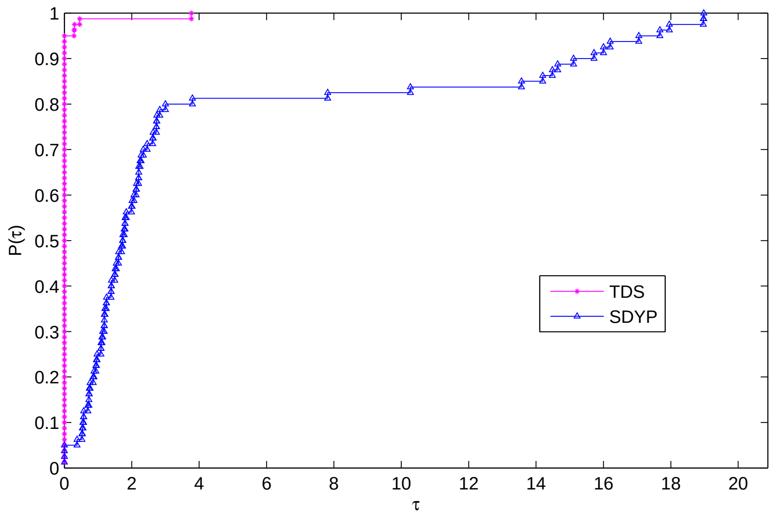

Figure 4.6: Performance profile of TDS and SDYP methods with respect to the CPU time (in second) for the problems 1-20.

we plotted Figures 4.5 and 4.6 according to the data in Tables 4.3 by using the performance profiles of Dolan and Mor$\acute{e}$ (2002).
From Figures 4.5-4.6, the performance profiles of TDS and SDYP method are compared with respect to the number of iterates and CPU time (in second) respectively. it is easy to see that the performance of TDS method, in most cases, is better than that of SDYP method, which show that TDS method grows up faster than SDYP method and so reaches to 1 before it. Thus, from the technique of Dolan and Mor$\acute{e}$ (2002), the performance curves in Figures. 4.5-4.6, for the number of iterates and CPU time (in second), demonstrate that the better results are achieved by TDS method.

4.3 RESULT DISCUSSION

We have tested the performance of our proposed methods namely IDFDD, DSDF and TDS with the respective existing methods i.e, IPRP, DFCG and SDYP methods, by solving twenty (20) benchmark nonlinear systems of equations problems. The numerical results of the methods are reported in Tables 4.1-4.6, where ”NI” is the total number of iterations, ”$\|F(x_k)\|$” is the norm of the residual at the stopping point, x^{*A} is the solution of the proposed methods and x^{*B} is the solution of the compared methods.

In Table 4.1, The numerical results indicate that the proposed method, i.e IDFDD has minimum number of iterations and CPU time, compared to IPRP respectively. we can easily see that our claim is fully justified from the tables, that is, less CPU time and number of iterations for each of the 20 test problems. This is due to the contribution of the added direction in each iteration (i.e double direction). Also, on average, our $\|F(x_k)\|$ is too small which signifies that the solution obtained is the true approximation of the exact solution compared to IPRP. Furthermore, IDFDD methods solved nonlinear equations problems where the compered method (IPRP) failed. (see problems 4, 8, 10 and 12 in Tables 4.1).

However, in Table 4.2, when comparing DSDF with DFCG subject to CPU time in seconds, we have seen that DSDF was better than DFCG. Moreover Comparing DSDF with DFCG subject to number of iterations, one can easily see that DSDF was better in 18 problems (i.e it achieved the minimum number of iterations). Therefore, in comparison DSDF appears to generate the best search direction due to the contribution of two step lengths in the method. Additionally, DSDF methods solved nonlinear equations problems where the DSDF failed. (see problems 4, 6, 8, 10, 15 and 16 in Tables 4.2).

The numerical results in Table 4.3, demonstrate clearly that the proposed method (TDS) shows better improvement, compared to SDYP. One can observe from the tables that the proposed method (i.e TDS) outperformed SDYP method in almost all the 20 test problems. However, TDS methods solved nonlinear equations problems where the compered method (SDYP) failed. (see problems 6,10, 12 and 14 in Tables 4.3).

Finally, in Table 4.4-4.6 we also report the behavior of the IDFDD algorithm,

DSDF algorithm and TDS algorithm for Functions 1, 6, 9 and 15 with some different initial points, in order to illustrate the global convergence.
Figures (4.1-4.6) show the performance of these methods relative to the number of iterations and CPU time, which were evaluated using the profiles of Dolan and Mor*é* (2002). That is, for each method, we plot the fraction $P(\tau)$ of the problems for which the method is within a factor τ of the best time. The top curve is the method that solved the most problems in a time that was within a factor τ of the best time. Clearly the proposed methods are more efficient in all aspect i.e number of iterations and CPU time.

CHAPTER FIVE

SUMMARY, CONCLUSION AND RECOMMENDATIONS

5.1 INTRODUCTION

This chapter gives the summary and conclusion of the entire work, together with suggestions and recommendations for further research.

5.2 SUMMARY

In this work, we were able to achieve the derivative-free, matrix-free and singularity-free double direction and step length methods. This is made possible by approximating the Jacobian matrix in the Newton method by sufficiently constructed diagonal matrix via acceleration parameter.
Chapter one of this dissertation, introduces the concepts of the Newton Method, Quasi-Newton for solving system of nonlinear equations. We however introduced the concept of double direction and step length methods for solving unconstrained optimization problem. Moreover, it contains, aim and objectives, scope and limitations and basic definitions of some terms.

Literature review was given in chapter two. In chapter three, the three proposed methods were presented. Also details of the derivations of the respective algorithms as well as convergence analysis of the proposed methods were included in the chapter.
Chapter four constitutes the numerical results. The methods are tested based on the performance profile developed by Dolan and More (2002) in terms of number

of iterations and CPU time in seconds. The results show that the three proposed methods are good alternative for solving nonlinear system of equations. Moreover, suggestion recommendation and conclusion were presented in the last chapter.

5.3 CONCLUSION

In this research, we presented three methods for solving large-scale system of nonlinear equations and compared there performances with that of an inexact PRP conjugate gradient method for symmetric nonlinear equations (IPRP) (Zhou & Shen, 2014), a derivative-free CG method and its global convergence for solving symmetric nonlinear equations (DFCG) (Waziri & Sabiu, 2015) and spectral DY-Type projection method for nonlinear monotone systems of equations (SDYP) (Liu & Li, 2015), by doing some intensive numerical experiments through solving different benchmark test problems. We therefore proved that the proposed methods i.e IDFDD, DSDF and TDS are faster and more efficient in terms of number of iterations and CPU time, compared to IPRP, DFCG and SDYP methods respectively. We however proved the global convergence of our proposed method by using a backtracking type line search, and the numerical results show that our method is very efficient.

5.4 FUTURE RESEARCH

- To apply our proposed methods to solve Real-life problems.
- Extending this work to solve nonsmooth nonlinear system of equations.

APPENDIX A

Test Functions

We present here the benchmark problems used to test the proposed methods in this research.

Problem 1: (Musa, 2015)

$$\begin{aligned} F_i(x) &= x_i^2 - 1, \\ i &= 1,2,3,...,n, \\ x_0 &= (0,0,...,0)^T, \\ x^* &= (1,1,,...,1)^T. \end{aligned}$$

Problem 2: (Musa, 2015)

$$\begin{aligned} F_i(x) &= x_i^2 + x_i - 2, \\ i &= 1,2,3,...,n, \\ x_0 &= (-0.5,-0.5,...,-0.5)^T, \\ x^* &= (1,1,,...,1)^T. \end{aligned}$$

Problem 3: (Musa, 2015)

$$\begin{aligned}F_i(x) = x_i - 1/n^2(\sum_{i=1}^{n} x_i)^2) + (\sum_{i=1}^{n} x_i) - n,\\ i = 1, 2, 3, ..., n,\\ x_0 = (5, 5, ..., 5)^T,\\ x^* = (1, 1, , ..., 1)^T.\end{aligned}$$

Problem 4: (Musa, 2015)

$$\begin{aligned}F_i(x) = (\sum_{i=1}^{n} x_i + i)(x_i - 1) + e^{x_i} - 1,\\ i = 1, 2, 3, ..., n,\\ x_0 = (0.3, 0.3, ..., 0.3)^T,\\ x^* = (1, 1, , ..., 1)^T.\end{aligned}$$

Problem 5: (Waziri & Sabiu, 2015)

$$\begin{aligned}F_i(x) = (1 - x_i^2) + x_i(1 + x_i x_{n-2} x_{n-1} x_n) - 2.\\ i = 1, 2, ..., n,\\ x_0 = (0.7, 0.7, ..., 0.7)^T,\\ x^* = (1, 1, , ..., 1)^T.\end{aligned}$$

Problem 6: (Waziri & Sabiu, 2015)

$$\begin{aligned}F_1(x) = x_1^2 - 3x_1 + 1 + cos(x_1 - x_2),\\ F_i(x) = x_1^2 - 3x_i + 1 + cos(x_i - x_{i-1}).\\ i = 2, 3, ..., n.\\ x_0 = (0.4, 0.4, ..., 0.4)^T\\ x^* = (1, 1, , ..., 1)^T.\end{aligned}$$

Problem 7: (Musa, 2015)

$$\begin{aligned} F_i(x) &= 5x_i^2 - 2x_i - 3, \\ i &= 1,2,...,n, \\ x_0 &= (0.5, 0.5, ..., 0.5)^T \\ x^* &= (1,1,,...,1)^T. \end{aligned}$$

Problem 8: (Musa, 2015)

$$\begin{aligned} F_i(x) &= \sum_{i=1}^{n} x_i(x_i - 2) + cos(x_i - 2) - 1, \\ i &= 1,2,...,n, \\ x_0 &= (1,1,...,1)^T, \\ x^* &= (2,2,,...,2)^T. \end{aligned}$$

Problem 9: (Yana, Penga & Li, 2010)

$$\begin{aligned} F_i(x) &= 2x_i - sin|x_i|, \\ i &= 1,2,...,n, \\ x_0 &= (-0.1, -0.1, ..., -0.1)^T, \\ x^* &= (0,0,...,0)^T. \end{aligned}$$

Problem 10: (Musa, 2015)

$$\begin{aligned} F_i(x) &= \sum_{i=1}^{n} (x_i^2 sinx_i) - x_i^4 + sinx_i^2, \\ i &= 1,2,3,...,n, \\ x_0 &= (-0.5, -0.5, ..., -0.5)^T, \\ x^* &= (0,0,...,0)^T. \end{aligned}$$

Problem 11: (Musa, 2015)

$$
\begin{aligned}
F_i(x) &= e^{x_i} - 1, \\
i &= 1, 2, ..., n, \\
x_0 &= (0.5, 0.5, 0.5, ..., 0.5)^T, \\
x^* &= (0, 0, ..., 0)^T.
\end{aligned}
$$

Problem 12:(Musa, 2015)

$$
\begin{aligned}
F_i(x) &= x_i(sinx_i cosx_i)^2 - x_i(cosx_i - x_i - 1), \\
i &= 1, 2, 3, ..., n, \\
x_0 &= (70, 70, ..., 70)^T, \\
x^* &= (0, 0, ..., 0)^T.
\end{aligned}
$$

Problem 13:(Musa, 2015)

$$
\begin{aligned}
F_1(x) &= \cos x_1 - 9 + 3x_1 + 8e^{x_2}, \\
F_i(x) &= \cos x_i - 9 + 3x_i + 8e^{x_{i-1}}, \\
i &= 2, 3, ..., n, \\
x_0 &= (0.5, 0.5, 0.5, ..., 0.5)^T, \\
x^* &= (0, 0, ..., 0)^T.
\end{aligned}
$$

Problem 14: (Musa, 2015)

$$\begin{aligned} F_i(x) &= e^{x_i^{2-i}} - cos(1-x_i), \\ i &= 1,2,...,n, \\ x_0 &= (0.3,0.3,0.3,...,0.3)^T, \\ x^* &= (1,1,,...,1)^T. \end{aligned}$$

Problem 15: (Musa, 2015)

$$\begin{aligned} F_i(x) &= x_i^2 - 4, \\ i &= 1,2,...,n, \\ x_0 &= (0.1,0.1,0.1,...,0.1)^T, \\ x^* &= (2,2,,...,2)^T. \end{aligned}$$

Problem 16: *(*Musa, 2015)

$$\begin{aligned} F_i(x) &= x_i x_{i+1} - 1, \\ F_n(x) &= x_n x_i - 1, \\ i &= 1,2,...,n-1, \\ x_0 &= (0.05,0.05,0.05,...,0.05)^T, \\ x^* &= (1,1,,...,1)^T. \end{aligned}$$

Problem 17: (Waziri & Sabiu, 2015)

$$F(x) = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2 \end{pmatrix} x + (e_1^x - 1, ..., e_n^x - 1)^T. \quad x_0 = (0.5, 0.5, ..., 0.5)^T,$$

$$x^* = (0, 0, ..., 0)^T.$$

Problem 18: *(*Musa, 2015)

$$\begin{aligned} F_i(x) &= x_1^2 + (x_i - 3)\log(x_{i+3}) - 9 \\ i &= 1, 2, ..., n, \\ x_0 &= (1, 1, ..., 1)^T, \\ x^* &= (3, 3, ..., 3)^T. \end{aligned}$$

Problem 19: (Musa, 2015)

$$\begin{aligned} F_1(x) &= 3x_1^3 + 2x_2 - 5 + \sin(x_1 - x_2)\sin(x_1 + x_2), \\ F_i(x) &= -x_{i-1}e^{x_{i-1} - x_i} + x_i(4 + 3x_i^2) + 2x_{i+1} + \sin(x_i - x_{i+1})\sin(x_i + x_{i+1}) - 8, \\ F_n(x) &= -x_{n-1}e^{x_{n-1} - x_n} + 4x_n - 3, \\ i &= 2, 3, ..., n-1, \\ x_0 &= (0.5, 0.5, ..., 0.5)^T, \\ x^* &= (1, 1, , ..., 1)^T. \end{aligned}$$

Problem 20: (Musa, 2015)

$$\begin{aligned} F_i(x) &= x_i^2 - cos(x_i - 1), \\ i &= 1, 2, ..., n, \\ x_0 &= (0.5, 0.5, ..., 0.5)^T, \\ x_* &= (1, 1, ..., 1)^T. \end{aligned}$$

APPENDIX B

MATLAB Codes for IDFDD, DSDF and TDS Methods

We have implemented the codes of the iterative methods described in Chapter 3 in the computer language MATLAB.

1. **Matlab code for IDFDD Method.**

```
function IDFDD(f,Guess)
% solves the non-linear vector equation  F(x)=0
% solution = The solution to F(x) = 0
tic
t = cputime;
tol=0.0001;
maxit=1000;
x = Guess;
p=0; %number of iterations
G =feval(f,x);
L=0.01;
d0=-(1/L)*G;
mag_of_G=sqrt(sum(G.^2));

while ( p < maxit && mag_of_G >tol);
```

```
    G =feval(f,x);

       r=0.8;
       sg=0.0001;
       n=1/(p+1)^2;
       m=0;
       while(0.5*(norm(feval(f,x+(r^m+(r^m)^2*L)*d0)))^2>=0.5*((norm(G)^2)
               -sg*(norm(r^m*d0))^2-sg*(norm(r^m*G)^2)+n*0.5*(norm(G))^2))

           m=m+1;
       end
      al=r^m;
      x1=x+(al+al^2*L)*d0;
      G1=feval(f,x1);
      y=G1-G;
      s=x1-x;
      L1=(y'*y)/(y'*s);
      d1=-(1/L1)*G1;
      x=x1;
      d0=d1;
      mag_of_G=sqrt(sum(G1.^2));
      p=p+1;
 end
 x
e=(cputime-t)
toc
mag_of_G
p
```

2. **Matlab code for DSDF Method.**

```
function DSDF(f,Guess)
% solves the non-linear vector equation  F(x)=0
% solution = The solution to F(x) = 0
tic
t = cputime;
tol=0.0001;
maxit=1000;
x = Guess;
p=0; %number of iterations
G =feval(f,x);
L=0.01;
d0=-(1/L)*G;
mag_of_G=sqrt(sum(G.^2));
while ( p < maxit && mag_of_G >tol  );
    G =feval(f,x);

       r=0.44;
       q=0.49;
       sg=0.0001;
       n=1/(p+1)^2;
       m=1;
while(0.5*(norm(feval(f,x+(r^m+q^m)*d0)))^2>=0.5*((norm(G)^2)
      -sg*(norm((r^m+q^m)*d0))^2-sg*(norm((r^m+q^m)*G)^2)+n*0.5*(norm(G))^2))

           m=m+1;
end
      ro=(r^m+q^m);
      x1=x+ro*d0;
      G1=feval(f,x1);
      y=G1-G;
      s=x1-x;
      L1=(y'*y)/(y'*s);
      d1=-(1/L1)*G1;
      x=x1;
      d0=d1;
      mag_of_G=sqrt(sum(G1.^2));
      p=p+1;
 end
 x
e=(cputime-t)
toc
 mag_of_G
p
```

3. **Matlab code for TDS Method.**

```
function TDS(f,Guess)
% solves the non-linear vector equation  F(x)=0
% solution = The solution to F(x) = 0
tic
t = cputime;
tol=0.0001;
maxit=2000;
x = Guess;
p=0; %number of iterations
G =feval(f,x);
L=0.01;
d0=-(1/L)*G;
mag_of_G=sqrt(sum(G.^2));
while ( p < maxit && mag_of_G >tol  );
    G =feval(f,x);

       r=0.2;
       sg=0.0001;
       n=1/(p+1)^2;
       m=0;
while(0.5*(norm(feval(f,x+(r^m+(0.5*r^m*L))*d0)))^2>=0.5*((norm(G)^2)
        -sg*(norm(r^m*d0))^2-sg*(norm(r^m*G)^2)+n*0.5*(norm(G))^2))

           m=m+1;
end
      al=r^m;
      x1=x+(al+(0.5*al*L))*d0;
      G1=feval(f,x1);
      y=G1-G;
      s=x1-x;
      L1=(y'*y)/(y'*s);
      d1=-(1/L1)*G1;
      x=x1;
      d0=d1;
      mag_of_G=sqrt(sum(G1.^2));
      p=p+1;
 end
 x
e=(cputime-t)
toc
 mag_of_G
p
```

APPENDIX C

Publications

1. Abubakar S. Halilu and M. Y. Waziri , *An Improved Derivative-Free Method Via Double Direction Approach For For Solving Systems Of Nonlinear Equations*, Journal of Engineering Mathematics. (accepted).

2. Abubakar S. Halilu and M. Y. Waziri ,*A Double Step Length Derivative-Free Method For Solving Systems Of Nonlinear Equations* ,Journal of mathematical and computational science. (accepted).

3. Abubakar S. Halilu and M. Y. Waziri ,*A Transformed Double Step Length Method For Solving Large-Scale Systems Of Nonlinear Equations* , American Journal of computational mathematics. (accepted).

REFERENCES

Broyden, C.G. (1965). A class of methods for solving nonlinear simultaneous equations. *Mathematics of Computation*, 19: 577-593.

Cruz, W. L, Martinez, J.M. and Raydan, M. (2006). Spetral residual method without gradient information for solving large-scale nonlinear systems of equations. *Mathematics of Computation*, 75: 1429-1448.

Dai, Y.H. and Yuan, Y. (1999). A nonlinear conjugate gradient method with a strong global convergence property. *SIAM Journal on Optimization*, 10: 177-182.

Daniel, J. W. (1967). The conjugate gradient method for linear and nonlinear operator equations. *SIAM Journal of Numerical Analysis*, 4: 10-26.

Dbaruranovic, N.I. (2008). A multi step curve search algorithm in nonlinear optimization. *Yugoslav Journal of Operation Research*, 18: 47-52.

Dennis, J.E. and More, J. J. (1974). A characterization of superlinear convergence and its application to quasi-Newton methods. *Mathematics of computation*, 28(126): 549-560.

Deuhard, P. (2012). A short history of Newton's method. *Documenta Mathematica, Optimization stories*, 25-30.

Dolan, E. and Mor*é*, J. (2002). Benchmarking optimization software with performance profiles. *Journal of Mathematical Program*, 91(2): 201-213.

Fletcher, R. and Reeves, C. (1964). Function minimization by conjugate gradients. *The Computer Journal*, 7(2): 149-154.

Griewank, A. (1986). The convergence of Broyden-like methods with a suitable line search. *Journal of the Australian Mathematical Society Series B*, 28: 75-92.

Griewank, A. (2012). Broyden updating, the good and the bad. *Documenta Mathematica, Optimization Stories*, 301-315.

Hestenes, M. R. and Stiefel, E. L. (1952). Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6): 409-436.

Krejic, N. and Luzanin, Z. (2001). Newton-Like Method with modification of right-hand vector. *Journal of Mathematics of Computation*, 237: 237-250.

Liu, Y.F. and Storey, C. (1991). Global convergence result for conjugate gradient methods. *Journal of Optimization Theory and Applications*, 71: 399-405.

Li, D. and Fukushima, M. (1999). A global and superlinear convergent Gauss-Newton based BFGS method for symmetric nonlinear equation. *SIAM Journal on Numerical Analysis*, 37: 152-172.

Liu, J.K. and Li, S.J. (2015). Spectral DY-Type projection method for nonlinear monotone systems of equations. *Journal of Computational Mathematics*, 33: 341-355.

Mascarenhas, W. F. (2004). The BFGS algorithm with exact line searches fails for nonconvex functions. *Journal of mathematical Programming*, 99(1): 49-61.

Musa, Y .B. (2015). *A Family Of BFGS -Methods For Solving Symmetric Nonlinear System Of Equations* (Unpublished master's dissertation). Bayero university, Kano, Nigeria.

Natasa, K. and Zorna, L. (2001). Newton-Like method with modification of rigtht-hand vector. *Mathematics of Computation*, 71: 237-250.

Petrovic, M.J. and Stanimirovic, P.S. (2014). Accelerated double direction method for solving unconstrained optimization problems. *Mathematical problem in Engineering*, 2014: 8.

Petrovic, M.J. (2015). An accelerated double step size model in unconstrained optimization. *Journal of Mathematics and Computation*, 250: 309-319.

Powell, M. J. D. (2000). On the convergence of the DFP algorithm for unconstrained optimization when there are only two variables. *Mathematical Programming*, 87: 281-301.

Raydan, M. (1993). On Barzilai and Borwein choice of step length for the gradient method. *IMA Journal of Numerical Analysis*, 13: 321-326.

Raydan, M. (1997). The Barzilai and Borwein gradient method for the large scale unconstrained minimization problem. *SIAM Journal of Optimization*, 7: 26-33.

Stanimirovic, P. S., Milovanovic, G. V., Petrovic, M. J. and Kontrec, N. Z. (2015). A transformation of accelerated double step size method for unconstrained optimization. *Journal of Mathematical Problems in Engineering*, 2015: 8.

Waziri, M.Y., Leong, W.J., Hassan, M.A. and Monsi, M. (2010). A new Newton's method with diagonal Jacobian approximation for system of nonlinear equations. *Journal of Mathematics and Statistics*, 6: 246-252.

Waziri, M.Y., Leong, W.J., Hassan, M.A. and Monsi, M. (2010). Jacobian computation-free Newton method for systems of Non-Linear equations. *Journal of Numumerical Mathematics and Stochastic*, 2: 54-63.

Waziri, M.Y. and Sabiu, J. (2015). A derivative-free conjugate gradient method and its global convergence for symmetric nonlinear equations. *Journal of Mathematics and Mathematical Sciences*, 8.

Yana, Q., Penga X. and Li, D. (2010). A globally convergent derivative-free method for solving large-scale nonlinear monotone equations. *Journal of Computational and Applied Mathematics*, 234: 649-657.

Yuan*, G. and Lu, X. (2008). A new backtracking inexact BFGS method for symmetric nonlinear equations. *Journal of Computers and Mathematics with Applications*, 55: 116-129.

Yuan, G. and Zhang, M. (2015). A three-terms Polak-Ribiere-Polyak conjugate gradient algorithm for large-scale nonlinear equations. *Journal of Computational and Applied Mathematics*, 286: 186-195.

Zang, L., Zhou, W. and Li, D.H. (2005). Global convergence of modified Fletcher-Reeves conjugate gradient method with Armijo-type line search, *Numerische Mathematik*, 164: 277-289.

Zhou, W. and Shen, D. (2014). An inexact PRP conjugate gradient method for symmetric nonlinear equations. *Numerical Functional Analysis and Optimization* , 35: 370-388.