Solving systems of nonlinear equations using hybrid optimization methods

Bishwesvar Pratap Singh

Licentiate thesis
April 2025

Supervisors:
UGIS Supervisor: Ph.D. Docent Yury V. Nikulin
Research Director and Supervisor: Professor Marko M. Mäkelä

Examiners:
External Reviewer: Ph.D., Docent Jussi Hakanen (University of Jyväskylä)
Internal Reviewer: Ph.D., University Lecturer Kaisa Joki (University of Turku)

Department of Mathematics and Statistics

UNIVERSITY OF TURKU
Faculty of Science
Department of Mathematics and Statistics
Applied Mathematics
SINGH, BISHWESVAR PRATAP: Solving Systems of Nonlinear Equations using Hybrid Optimization Methods
Licentiate thesis, 92 p.
April 2025

ABSTRACT

This licentiate thesis considers various hybrid methods for solving systems of nonlinear equations. Extensive numerical experiments confirm a research hypothesis that one particular form of hybridization of the conjugate gradient algorithm with Newton's method outperforms other hybrid strategies considered in the thesis. Although the findings are preliminary and the computational experiments have certain limitations, this study offers solid insights by highlighting the most promising hybridization techniques for further research and potential advancements in solving nonlinear systems.

KEYWORDS: Newton's method, iterative algorithm, descent direction, conjugate gradient, nonlinear optimization.

TURUN YLIOPISTO
Matemaattis-luonnontieteellinen tiedekunta
Matematiikan ja tilastotieteen laitos
Sovelettu matematiikka
SINGH, BISHWESVAR PRATAP: Epälineaaristen yhtälöryhmien ratkaiseminen hybridioptimointimenetelmillä
Lisensiaatintutkielma, 92 s.
Huhtikuu 2025

TIIVISTELMÄ

Tässä lisensiaatintyössä tarkastellaan erilaisia optimoinnin hybridimenetelmiä epälineaaristen yhtälöryhmien ratkaisemiseksi. Numeeriset testit vahvistavat tutkimushypoteesin, jonka mukaan eräs konjugaattigradienttimenetelmän ja Newtonin algoritmin hybridisaatiomuoto toimii paremmin kuin muut opinnäytetyössä tarkastellut hybridistrategiat. Vaikka tulokset ovatkin vasta alustavia, luovat ne hyvän lähtökohdan epälineaaristen yhtälöryhmien ratkaisemiseen soveltuvien optimoinnin hybridimentelmien jatkokehitystyölle.

AVAINSANAT: Newtonin menetelmä, iteratiivinen algoritmi, laskeva suunta, konjugaattigradientti, epälineaarinen optimointi.

Contents

Symbols and Acronyms

$\mathbb{R}^n$	Real n-dimensional space
$\mathbb{R}$	Set of real numbers
$\mathbb{N}$	Set of natural numbers
$B(\boldsymbol{x}; r)$	Open ball with a radius r and a center point $\boldsymbol{x}$
$\{\boldsymbol{x}_k\}$	Sequence of vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \ldots$
$\lceil \ \rceil$	Ceiling function
$\boldsymbol{x}^T\boldsymbol{y}$	Inner product of $\boldsymbol{x}$ and $\boldsymbol{y}$
$\|\cdot\|$	Vector norm
$(A)_{ij}$	Element of the matrix A in row i of column j
A^T	Transpose of the matrix A
A^{-1}	Inverse of the matrix A
$\det(A)$	Determinant of the matrix A
I	Identity matrix
$\nabla f(\boldsymbol{x})$	Gradient of the function f at $\boldsymbol{x}$
$J(\boldsymbol{x}_k)$	Jacobian matrix at $\boldsymbol{x}_k$
$\frac{\partial}{\partial x_i}\boldsymbol{f}(\boldsymbol{x})$	Partial derivative of function $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$ with respect to x_i
$\partial\boldsymbol{f}(\boldsymbol{x})$	Generalized Jacobian matrix of the function $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$ at $\boldsymbol{x}$
B_k	Approximation of the Jacobian matrix at $\boldsymbol{x}_k$
$T(x)$	Tangent line at $x \in \mathbb{R}$
$H(\boldsymbol{x})$	Hessian at $\boldsymbol{x}$
H_k	Hessian at the k^{th} iteration
H_k^{-1}	Inverse of Hessian at the k^{th} iteration

I_k	Length of interval of uncertainty at the k^{th} iteration
$\mathcal{F}$	Fibonacci series
λ_k	Step length at the k^{th} iteration
$\boldsymbol{d}_k$	Descent direction at the k^{th} iteration
conv S	Convex hull of the set S
int S	Interior of the set S
$f'(\boldsymbol{x};\boldsymbol{d})$	Directional derivative of the function f at $\boldsymbol{x}$ in the direction $\boldsymbol{d}$
$\lvert\lvert\lvert\cdot\rvert\rvert\rvert$	Matrix norm
ϵ	Predefined positive tolerance
N	Maximum number of iterations
$A \succeq 0$	A is positive semidefinite
$A \succ 0$	A is positive definite

Introduction

Solving systems of nonlinear equations is a fundamental problem in applied mathematics, engineering, and scientific computing. Unlike linear systems, where robust and well-established techniques guarantee solutions, nonlinear systems often require more nuanced approaches due to their complexity and potential for multiple solutions, divergence, or non-existence of solutions. This thesis outlines the primary methodologies for tackling systems of nonlinear equations, emphasizing theoretical foundations, numerical techniques, and practical considerations.

For systems of linear equations, all techniques are generally divided into direct methods and iterative methods. Direct methods, such as Gaussian elimination, QR decomposition, and matrix inversion, provide solutions in a finite number of steps, given that the problem is well-posed and numerical stability is ensured. In contrast, iterative methods, including the Jacobi method and the conjugate gradient method, are particularly well suited for large, sparse systems because of their computational efficiency and scalability.

A system of nonlinear equations consists of multiple interdependent equations involving unknown variables. The objective is to find values for these variables that satisfy all equations simultaneously. Nonlinear systems are inherently more complex than linear systems due to the presence of nonlinearity, which can manifest as powers, trigonometric functions, or other non-linear relationships.

While analytical solutions are rare for nonlinear systems, they can be pursued when specific conditions apply. Common approaches include:

- Substitution method: Solving one equation for a variable and substituting it into others, though this quickly becomes impractical for larger systems.
- Graphical methods: Visualizing solutions by plotting functions, feasible only for systems with two or three variables.
- Symbolic computation: Using algebraic manipulation tools like Gröbner bases or software such as Mathematica or Maple to find exact solutions.

Analytical methods are limited to small systems with well-behaved functions. Therefore, in this thesis, we focus on another big class of methods known as iterative numerical methods. For most practical problems, numerical techniques are the preferred approach. These methods iteratively approximate the solution until convergence criteria are met.

For complex systems, iterative numerical methods provide approximate solutions. These methods involve iteratively improving an initial guess until the solution satisfies a predefined accuracy.

- Newton's method: This method refines the solution by approximating the system as a simpler one near the current estimate. It is highly efficient when the initial guess is close to the true solution. However, it requires additional computations and can fail if the guess is too far off or the problem is poorly conditioned.

- Quasi-Newton methods: These methods reduce computational overhead by approximating the relationships between variables rather than recalculating them at every step. They are widely used in optimization problems.

- Fixed-Point iteration: This approach transforms the system into a form in which variables can be successively updated. Convergence depends on the specific properties of the equations and transformation.

- Trust-Region methods: These methods improve stability by limiting the degree to which the solution is adjusted in each step, ensuring that the process remains reliable even for difficult systems.

A system of nonlinear equations can often be solved indirectly by reformulating it as a single objective unconstrained optimization problem [56]. One of the first alternative methods to the classical Newton-based iterative numerical methods was the conjugate gradient method. This approach was first proposed by Hestenes and Stiefel in 1952 [36]. The conjugate gradient algorithm is based on a series of conjugate directions that are generated so that, for example, in the linear cases, they are mutually orthogonal. This enables efficient convergence towards the solution of the linear system. However, this algorithm was not widely used until the early 1960s when Fletcher and Reeves [30] improved it. In this variant, the definition of conjugacy is slightly altered. Instead of seeking exact conjugacy, it allows for a more relaxed criterion for conjugacy. This improves convergence behavior, and it can be especially beneficial when dealing with ill-conditioned or nearly linearly dependent systems [30].

Another important innovation occurred when the idea of hybridization of several methods was met with attention. Hybridization involves combining two or more optimization methods, algorithms, or strategies into a unified framework to leverage the strengths of each component.

In 1970, Powell [47] introduced a hybrid method that integrated the conjugate gradient and Newton's methods. This approach, designed to solve unconstrained optimization problems, demonstrated greater efficiency than either method alone, especially when applied to non-linear optimization problems with a large number of variables. Over the years, researchers have continued to develop and refine hybrid methods, contributing to their growing applicability and effectiveness in solving unconstrained optimization problems [54, 56].

Several factors influence the success of solving nonlinear systems:

- Initial guess: The choice of the starting point is critical for methods like Newton's. A poor guess can lead to divergence or convergence to an unintended solution.

- Scaling: Rescaling the equations or variables ensures that numerical computations remain stable and efficient.
- Stopping criteria: Clearly defined conditions, such as the size of adjustments or the accuracy of the solution, prevent unnecessary computations.
- Software tools: Modern computational tools like MATLAB, Python libraries, and dedicated solvers significantly simplify the process.

This work focuses on solving systems of nonlinear equations by reformulating them as unconstrained optimization problems and employing a hybrid approach that integrates multiple established optimization techniques. Specifically, we investigate the combination of the Gradient (G), Conjugate Gradient (CG), Newton's (N), and Quasi-Newton (QN) methods. These methods are organized into two groups: the first includes the gradient and conjugate gradient methods, while the second encompasses Newton's and quasi-Newton methods. These methods were selected for their compatibility, enabling seamless integration without significant additional computational effort.

The primary objective of this research was to identify the optimal combination of these methods. By selecting one method from each group, we constructed four distinct hybrid algorithms, enabling a comparative analysis based on performance metrics such as iteration count, function evaluations, and computational efficiency. Among these, the hybridization of the Conjugate Gradient and Newton's methods (referred to as CGN) emerged as the most robust optimization algorithm, leveraging the strengths of both approaches. Additionally, the CGN algorithm demonstrated effectiveness in solving small- to medium-scale nonlinear optimization problems. However, for large-scale problems, its computational cost becomes a limitation due to the requirement of computing the Hessian matrix at each iteration.

This thesis is structured as follows: Chapter 1 outlines the foundational definitions and concepts essential to the study. Chapter 2 focuses on nonlinear equations involving one or multiple variables and explores various solution techniques. Chapter 3 provides an overview of methods for unconstrained optimization. Chapter 4 details hybridization strategies for addressing systems of nonlinear equations by reformulating them as unconstrained optimization problems. Finally, Chapter 5 examines two practical applications in which the hybrid conjugate gradient and Newton (CGN) algorithm is expected to demonstrate potential utility based on empirical results obtained in Chapter 4.

Chapter 1

Preliminaries

This chapter presents basic definitions to be used throughout the thesis. For more details on the theory related to the specified definitions, we refer to [1, 7, 11, 44].

Definition 1. A function $F : \mathbb{R}^n \to \mathbb{R}$ is said to be
convex if

$$F(\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y}) \leq \lambda F(\boldsymbol{x}) + (1-\lambda)F(\boldsymbol{y}) \quad \text{for all } \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n \text{ and } \lambda \in [0,1] \tag{1.1}$$

strictly convex if

$$F(\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y}) < \lambda F(\boldsymbol{x}) + (1-\lambda)F(\boldsymbol{y}) \quad \text{for all } \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n, \boldsymbol{x} \neq \boldsymbol{y} \text{ and } \lambda \in (0,1) \tag{1.2}$$

strongly convex if

$$F(\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y}) \leq \lambda F(\boldsymbol{x}) + (1-\lambda)F(\boldsymbol{y}) - \frac{m}{2}\lambda(1-\lambda)\|\boldsymbol{x}-\boldsymbol{y}\|^2 \tag{1.3}$$

for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$, $\boldsymbol{x} \neq \boldsymbol{y}$ and $\lambda \in (0,1)$. In addition, $m > 0$ is called a *strong convexity constant*.

Definition 2. The *inner product* in the n-dimensional real Euclidean space $\mathbb{R}^n$ can be defined as

$$\boldsymbol{x}^T\boldsymbol{y} = \sum_{i=1}^{n} x_i y_i$$

where $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ and $x_i, y_i \in \mathbb{R}$.

Definition 3. The *vector norm* for a column vector $\boldsymbol{x}$ in the n-dimensional real Euclidean space $\mathbb{R}^n$ can be defined as

$$\|\boldsymbol{x}\| = (\boldsymbol{x}^T\boldsymbol{x})^{\frac{1}{2}}.$$

Definition 4. An *open ball* with a center $\boldsymbol{x} \in \mathbb{R}^n$ and a radius $r > 0$ is denoted by $B(\boldsymbol{x}; r)$, that is

$$B(\boldsymbol{x}; r) = \{\boldsymbol{y} \in \mathbb{R}^n \quad | \quad \|\boldsymbol{y} - \boldsymbol{x}\| < r\}.$$

Definition 5. Suppose that $\{\boldsymbol{x}_k\}$ is a sequence in $\mathbb{R}^n$ converging to $\boldsymbol{x}^*$.

1) The sequence $\{\boldsymbol{x}_k\}$ is said to converge *linearly* if there exists $c \in (0,1)$ such that

$$\|\boldsymbol{x}^* - \boldsymbol{x}_{k+1}\| \leq c\|\boldsymbol{x}^* - \boldsymbol{x}_k\| \quad \text{for all } k \text{ sufficiently large.}$$

2) The sequence $\{\boldsymbol{x}_k\}$ is said to converge *superlinearly* if

$$\frac{\|\boldsymbol{x}^* - \boldsymbol{x}_{k+1}\|}{\|\boldsymbol{x}^* - \boldsymbol{x}_k\|} \to 0 \quad \text{as } k \to \infty.$$

3) The sequence $\{\boldsymbol{x}_k\}$ is said to converge *quadratically* if there exists $c \in (0,\infty)$ such that

$$\|\boldsymbol{x}^* - \boldsymbol{x}_{k+1}\| \leq c\|\boldsymbol{x}^* - \boldsymbol{x}_k\|^2 \quad \text{for all } k \text{ sufficiently large.}$$

Definition 6. A function $F : \mathbb{R}^n \to \mathbb{R}$ is said to be *nonlinear* if it does not satisfy the *principle of superposition*, that is

$$F(\boldsymbol{x}_1 + \boldsymbol{x}_2 + \ldots + \boldsymbol{x}_l) \neq F(\boldsymbol{x}_1) + F(\boldsymbol{x}_2) + \ldots + F(\boldsymbol{x}_l)$$

for some combination of $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_l \in \mathbb{R}^n$.

Definition 7. The vector $\nabla F(\boldsymbol{x})$ is called the *gradient vector* of the function $F : \mathbb{R}^n \to \mathbb{R}$ at $\boldsymbol{x} \in \mathbb{R}^n$ and it is defined as:

$$\nabla F(\boldsymbol{x}) = \left[\tfrac{\partial F(\boldsymbol{x})}{\partial x_1}, \ldots, \tfrac{\partial F(\boldsymbol{x})}{\partial x_n}\right]^\top,$$

where the components $\frac{\partial F(\boldsymbol{x})}{\partial x_i}$ for $i = 1, \ldots, n$, are called the *partial derivatives* of the function F.

Definition 8. In vector calculus, the *Jacobian matrix* is the matrix of the first order partial derivatives of a vector-valued function. Suppose $\boldsymbol{F} : \mathbb{R}^n \to \mathbb{R}^m$ is a vector-valued differentiable function. Then at a point $\boldsymbol{x} \in \mathbb{R}^n$ the *Jacobian matrix* $J(\boldsymbol{x})$ of $\boldsymbol{F} = (F_1, \ldots, F_m)$ is a $m \times n$ matrix whose i^{th} row is given by the gradient vector ∇F_i composed of partial derivatives $\frac{\partial F_i}{\partial x_j}(\boldsymbol{x}) = \frac{\partial F_i}{\partial x_j}(x_1, x_2, \ldots, x_n)$, in other words

$$J(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial F_1}{\partial x_1}(\boldsymbol{x}) & \frac{\partial F_1}{\partial x_2}(\boldsymbol{x}) & \cdots & \frac{\partial F_1}{\partial x_n}(\boldsymbol{x}) \\ \frac{\partial F_2}{\partial x_1}(\boldsymbol{x}) & \frac{\partial F_2}{\partial x_2}(\boldsymbol{x}) & \cdots & \frac{\partial F_2}{\partial x_n}(\boldsymbol{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F_m}{\partial x_1}(\boldsymbol{x}) & \frac{\partial F_m}{\partial x_2}(\boldsymbol{x}) & \cdots & \frac{\partial F_m}{\partial x_n}(\boldsymbol{x}) \end{bmatrix}.$$

Definition 9. The matrix $H(\boldsymbol{x})$ is called the *Hessian matrix* of the function $F : \mathbb{R}^n \to \mathbb{R}$ at $\boldsymbol{x}$, and it is defined to consist of the second-order partial derivatives of F. That is,

$$H(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial^2 F(\boldsymbol{x})}{\partial x_1^2} & \cdots & \frac{\partial^2 F(\boldsymbol{x})}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 F(\boldsymbol{x})}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 F(\boldsymbol{x})}{\partial x_n^2} \end{bmatrix}.$$

Definition 10. The system of equations can be written as:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2, \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m, \end{aligned}$$

where a_{ij} represents the coefficients of the variables $x_1, x_2, \ldots, x_n$, and b_i represents the constants on the right-hand side. The augmented matrix for this system is given by:

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} & b_m \end{array}\right].$$

In this matrix, the first n columns form the constraints matrix, and the final column represents the constants from the right-hand side of the equations.

Definition 11. A function $F : \mathbb{R}^n \to \mathbb{R}$ is *locally Lipschitz continuous* at a point $\boldsymbol{x} \in \mathbb{R}^n$ if there exist scalars $L > 0$ and $\epsilon > 0$ such that

$$|F(\boldsymbol{y}) - F(\boldsymbol{z})| \leq L\|\boldsymbol{y} - \boldsymbol{z}\| \quad \text{for all} \quad \boldsymbol{y}, \boldsymbol{z} \in B(\boldsymbol{x}; \epsilon).$$

A vector valued function $\boldsymbol{F} : \mathbb{R}^n \to \mathbb{R}^m$ is *locally Lipschitz continuous* at a point $\boldsymbol{x} \in \mathbb{R}^n$ if F_i is locally Lipschitz continuous at $\boldsymbol{x}$ for all $i = 1, \ldots, m$.

Definition 12. A function $F : \mathbb{R}^n \to \mathbb{R}$ is *Lipschitz continuous* on a set $U \subset \mathbb{R}^n$ if there exists a scalar $L > 0$ such that

$$|F(\boldsymbol{y}) - F(\boldsymbol{z})| \leq L\|\boldsymbol{y} - \boldsymbol{z}\| \quad \text{for all} \quad \boldsymbol{y}, \boldsymbol{z} \in U.$$

A vector valued function $\boldsymbol{F} : \mathbb{R}^n \to \mathbb{R}^m$ is *Lipschitz continuous* on a set $U \subset \mathbb{R}^n$ if F_i is Lipschitz continuous on U for all $i = 1, \ldots, m$.

Definition 13. The *convex hull* of a set $S \subseteq \mathbb{R}^n$ is

$$\text{conv } S = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{x} = \sum_{i=1}^{k} \lambda_i \boldsymbol{x}_i, \quad \sum_{i=1}^{k} \lambda_i = 1, \quad \boldsymbol{x}_i \in S, \quad \lambda_i \geq 0, \quad k > 0\}.$$

Definition 14. Let $\boldsymbol{F} : \mathbb{R}^n \to \mathbb{R}^m$ be locally Lipschitz continuous at a point $\boldsymbol{x} \in \mathbb{R}^n$. Then the *generalised Jacobian* of $\boldsymbol{F}$ at $\boldsymbol{x}$ is the set

$$\partial \boldsymbol{F}(\boldsymbol{x}) := \text{conv}\{A \in \mathbb{R}^{m \times n} \mid \exists \, \{\boldsymbol{x}_i\} \subset \mathbb{R}^n \backslash \Omega_F \text{ such that } \boldsymbol{x}_i \to \boldsymbol{x} \text{ and } J(\boldsymbol{x}_i) \to A\},$$

where $\Omega_{\boldsymbol{F}}$ is a set in $\mathbb{R}^n$ that contains all nondifferentiable points.

Definition 15. Let A be a $m \times n$ matrix, then *matrix norm* of A can be defined as

$$|||A||| := \left(\sum_{i=1}^{m} \|\boldsymbol{a}_i\|^2 \right)^{\frac{1}{2}},$$

where $\boldsymbol{a}_i \in \mathbb{R}^n$ is the i^{th} row of A.

Definition 16. The *classical directional derivative* of $F : \mathbb{R}^n \to \mathbb{R}$ at $\boldsymbol{x} \in \mathbb{R}^n$ in the direction $\boldsymbol{d} \in \mathbb{R}^n$ is defined by

$$F'(\boldsymbol{x}; \boldsymbol{d}) = \lim_{t \downarrow 0} \frac{F(\boldsymbol{x} + t\boldsymbol{d}) - F(\boldsymbol{x})}{t}.$$

If F is differentiable at $\boldsymbol{x}$, then the directional derivative, which exists in every direction $\boldsymbol{d} \in \mathbb{R}^n$, is a linear operator on $\boldsymbol{d}$. In other words, we have the relation

$$F'(\boldsymbol{x}; \boldsymbol{d}) = \nabla F(\boldsymbol{x})^T \boldsymbol{d},$$

where $\nabla F(\boldsymbol{x}) \in \mathbb{R}^n$ is the gradient vector of F at $\boldsymbol{x}$. Shapiro [53] proved that a locally Lipschitz continuous function $F : \mathbb{R}^n \to \mathbb{R}$ is directionally differentiable at $\boldsymbol{x}$ if and only if

$$\lim_{\|\boldsymbol{d}\| \downarrow 0} \frac{F(\boldsymbol{x} + \boldsymbol{d}) - F(\boldsymbol{x}) - F'(\boldsymbol{x}; \boldsymbol{d})}{\|\boldsymbol{d}\|} = 0.$$

Definition 17. In algebra, a matrix $A \in \mathbb{R}^{n \times n}$ is called *positive definite* if

$$\boldsymbol{x}^T A \boldsymbol{x} > \boldsymbol{0} \quad \text{for all } \boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{x} \neq \boldsymbol{0}.$$

A matrix $A \in \mathbb{R}^{n \times n}$ is called *positive semidefinite* if

$$\boldsymbol{x}^T A \boldsymbol{x} \geq \boldsymbol{0} \quad \text{for all } \boldsymbol{x} \in \mathbb{R}^n.$$

In numerical analysis, an iterative method is called *convergent* if the error measure of successive iteration points $\boldsymbol{x}_k$ defined as a distance between computed and true value for some vector-valued function $\boldsymbol{F} : \mathbb{R}^n \to \mathbb{R}^m$ with true solution $\boldsymbol{x}^*$ tends to zero. In other words, we need to have

$$\lim_{\|\boldsymbol{x}_k - \boldsymbol{x}^*\| \downarrow 0} \|\boldsymbol{F}(\boldsymbol{x}_k) - \boldsymbol{F}(\boldsymbol{x}^*)\| = 0.$$

It means that as $\boldsymbol{x}_k$ gets arbitrarily close to $\boldsymbol{x}^*$, the values of $\boldsymbol{F}(\boldsymbol{x}_k)$ get arbitrarily close to true value $\boldsymbol{F}(\boldsymbol{x}^*)$.

Iterative methods are termed *locally convergent* if they guarantee convergence to an optimal solution, provided the initial point is sufficiently close to the solution. Methods used for nonlinear equations and systems, such as Newton-based approaches, typically exhibit only local convergence. If the initial guess is poorly chosen, the method may converge slowly or fail to converge altogether.

Globally convergent iterative methods are those that converge from any initial point. Achieving global convergence often necessitates using conservative step sizes, which can slow down the rate of convergence. Iterative methods for solving systems of linear equations typically exhibit global convergence. A thorough understanding of both the local and global convergence properties of numerical methods is crucial for choosing appropriate algorithms and assessing their reliability and efficiency in addressing specific problems.

Chapter 2

Solving Nonlinear Equations

Solving systems of equations is a fundamental problem in applied mathematics. Numerous robust and efficient techniques are available for addressing systems of linear equations. Significant progress has also been achieved in finding techniques that extend Newton's method to solve systems of nonlinear equations. These techniques are particularly appealing as they retain the fast local convergence properties inherent to Newton-based methods for smooth equations. In this chapter, we provide a short review of the methodology suitable for solving systems of nonlinear equations.

2.1 Nonlinear equations

In mathematics, a system of nonlinear equations is a set of equations where one or more terms have, for example, a variable of degree two or higher and/or there is a product or other more complex forms of combining variables in at least one of the equations. Nonlinearity is a broader concept that includes any equation that cannot be written as a linear combination of variables. Nonlinear equations are solved as part of many simulations of physical processes. Generally, we seek for values of n variables, $(x_1, x_2, \ldots, x_n) = \boldsymbol{x} \in \mathbb{R}^n$ to satisfy m nonlinear equations

$$\boldsymbol{F}(\boldsymbol{x}) = \begin{pmatrix} F_1(\boldsymbol{x}) \\ F_2(\boldsymbol{x}) \\ \vdots \\ F_m(\boldsymbol{x}) \end{pmatrix} = \boldsymbol{0}, \tag{2.1}$$

where $F_1, F_2, \ldots, F_m : \mathbb{R}^n \to \mathbb{R}$ are twice continuously differentiable functions. Many direct and indirect optimization-based methods (see [54, 56]) have been developed for solving (2.1). The solution of a system of nonlinear equations can rarely be given by a closed-form expression, so one must use iterative methods to approximate the solution numerically. The output of such an iterative method is a sequence of approximations to a solution. The value of such methods is measured in terms of the computational cost of obtaining the successive term of the sequence $\{\boldsymbol{x}_k\}$ and the convergence rate. We define linearisation of equation (2.1) at an iteration point $\boldsymbol{x}_k$ by

$$\boldsymbol{F}(\boldsymbol{x}_k) + J(\boldsymbol{x}_k)\boldsymbol{z}_k = \boldsymbol{0} \quad \forall \quad k= 0,1,\ldots \tag{2.2}$$

where $\boldsymbol{z}_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k$ and $J(\boldsymbol{x}_k)$ is the Jacobian matrix of $\boldsymbol{F}$ at $\boldsymbol{x}_k$. If $J(\boldsymbol{x}_k)$ is a nonsingular square matrix, then the determinant is nonzero and $J(\boldsymbol{x}_k)$ has an inverse. The linear approximation (2.2) gives the Newton-Raphson iteration

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - J(\boldsymbol{x}_k)^{-1}\boldsymbol{F}(\boldsymbol{x}_k). \tag{2.3}$$

Equation (2.3) is a generalization of the method proposed in 1669 by Sir Isaac Newton, who first used the iteration

$$x_{k+1} = x_k - \frac{F(x_k)}{F'(x_k)} \quad \forall \quad k{=}0,1,\ldots \tag{2.4}$$

to solve a cubic equation $F(x) = 0$ with a single-valued function $F(x) := x^3 - 2x - 5$ of one variable x. In 1690, Raphson first employed the formula (2.3) to solve a general cubic equation [49]. Then Cauchy (1829), Fourier (1890), and Fine (1916) established convergence theorems of Newton's method for different cases [18, 26, 31]. In 1948, Kantorovich established a convergence theorem [39], referred to as the Newton-Kantorovich theorem, which is the main tool for proving the convergence of various Newton-type methods.

There are various Newton-type methods for solving nonlinear equations. Dembo et al. [23] proposed an inexact Newton's method that solves the linear equations (2.2) approximately. Another efficient approach is approximating the Jacobian or the inverse of the Jacobian that satisfies the secant equation

$$B_{k+1}\boldsymbol{z}_k = \boldsymbol{F}(\boldsymbol{x}_{k+1}) - \boldsymbol{F}(\boldsymbol{x}_k),$$

where B_{k+1} is an approximation of the Jacobian matrix at point $\boldsymbol{x}_{k+1}$. For highly nonlinear problems, these Newton-based schemes tend to fail or require a carefully chosen starting point to produce a solution [9]. The reason is that unless starting points are close enough to a potentially optimal solution, the iterates may converge to another local minimum of the function.

2.2 Solving a single nonlinear equation with one variable

In this section, we first deal with the simplest problem of solving a single equation $F(x) = 0$ with only one variable. Algorithms for one-dimensional problems are the basis of algorithms for higher-dimensional problems. For example, Newton's method has analogs for higher-dimensional problems. Other algorithms, such as the bisection method, are strictly one-dimensional.

2.2.1 Bisection method

The bisection method, or bisection search, is the simplest and most robust way to find the root of a function with one variable. It does not require the considered function to be differentiable but merely continuous on its domain. The method

is based on a simple topological result called the *Intermediate Value Theorem*: if $F : \mathbb{R} \to \mathbb{R}$ is a continuous function of argument x on the interval $a \leq x \leq b$ such that $F(a) < 0 < F(b)$ then there exists at least one $x^* \in (a, b)$ such that $F(x^*) = 0$, and this holds similarly in the case $F(b) < 0 < F(a)$.

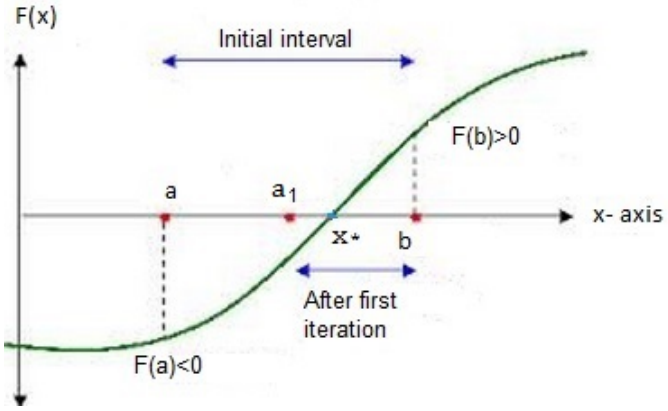

Figure 2.1: First iteration of the Bisection method.

Computational algorithm

The bisection search algorithm involves the iterative division of an interval in which a root is known to exist, as shown in figure 2.1. Initially, the algorithm selects an initial interval $[a, b]$ in which the function F changes its sign. Subsequently, the bisection procedure generates intervals $[a_k, b_k]$ for $k = 1, 2, 3, ...$ that progressively decrease in size, halving with each iteration, following an exponential rate of decrease given by:

$$|b_k - a_k| = \frac{|b - a|}{2^{k-1}}, \quad k = 1, 2, \ldots$$

where a_k and b_k can be calculated from the following computational steps in Algorithm 1:

Algorithm 1: Bisection method [6]

Input: Initial interval $[a, b]$, tolerance $\epsilon > 0$, maximum number of iterations N

Output: Approximate solution x^*

1 Calculate $F(a)$ and $F(b)$
If $F(a)F(b) \geq 0$
OUTPUT: ('No root in $[a, b]$, or the root is trivially at a or b).
Go to step 4.
Else set $k = 1$, $a_k = a$, $b_k = b$, and compute midpoint $x_k = \frac{a_k + b_k}{2}$.

2 While $(k \leq N)$
If $F(a_k)F(x_k) = 0$ then the root is at $x^* = x_k$ and go to step 4.
Else if $F(a_k)F(x_k) < 0$, then the root lies in (a_k, x_k)
If $|F(x_k)| > \epsilon$ then set $(a_{k+1}, b_{k+1}) = (a_k, x_k)$ else $x^* = x_k$ and go to step 4.
Else if $F(a_k)F(x_k) > 0$, then the root lies between (x_k, b_k).
If$|F(x_k)| > \epsilon$ then set $(a_{k+1}, b_{k+1}) = (x_k, b_k)$ else $x^* = x_k$ and go to step 4.
Update iteration counter $k = k + 1$.
Recompute midpoint $x_k = \frac{a_k + b_k}{2}$.

3 End while
OUTPUT: ('Method failed to converge after N iterations')

4 Stop

Example with Matlab

Here we discuss an example. Given the function $F(x) = x^5 - 3x^3 + 2.5x - 0.6$, we see graphically

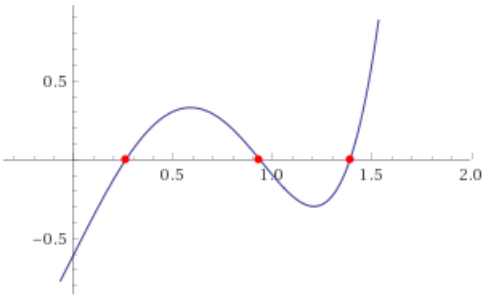

Figure 2.2: Plot of F.

that there are three positive roots for $F(x) = 0$ (Figure 2.2). We solve equation $F(x) = 0$ with initial interval $[1, 2]$, maximum number of iterations $N = 50$ and tolerance $\epsilon = 10^{-6}$. The bisection method took 20 iterations to reach the solution $x^* = 1.38$. Computation results are shown in Table 2.1.

Iter	a_k	b_k	x_k	F(x_k)	(b_k-a_k)
1	1.000000	2.000000	1.500000	0.618750	1.00000000
2	1.000000	1.500000	1.250000	-0.282617	0.50000000
3	1.250000	1.500000	1.375000	-0.046442	0.25000000
4	1.375000	1.500000	1.437500	0.220548	0.12500000
5	1.375000	1.437500	1.406250	0.072249	0.06250000
6	1.375000	1.406250	1.390625	0.009393	0.03125000
7	1.375000	1.390625	1.382812	-0.019379	0.01562500
8	1.382812	1.390625	1.386719	-0.005209	0.00781250
9	1.386719	1.390625	1.388672	0.002038	0.00390625
10	1.386719	1.388672	1.387695	-0.001599	0.00195312
11	1.387695	1.388672	1.388184	0.000216	0.00097656
12	1.387695	1.388184	1.387939	-0.000693	0.00048828
13	1.387939	1.388184	1.388062	-0.000239	0.00024414
14	1.388062	1.388184	1.388123	-0.000012	0.00012207
15	1.388123	1.388184	1.388153	0.000102	0.00006104
16	1.388123	1.388153	1.388138	0.000045	0.00003052
17	1.388123	1.388138	1.388130	0.000017	0.00001526
18	1.388123	1.388130	1.388126	0.000003	0.00000763
19	1.388123	1.388126	1.388124	-0.000004	0.00000381
20	1.388124	1.388126	1.388125	-0.000001	0.00000191

Table 2.1: Solution of $F(x) = x^5 - 3x^3 + 2.5x - 0.6 = 0$ with the initial interval [1,2] using the bisection method.

Convergence

Although the convergence rate of the bisection method may seem good, it converges slowly toward the solution because it halves the intervals. Another disadvantage is that it requires the exact evaluation of $F(x)$, as a computed approximation may not be continuous on x on a fine scale due to the wrong sign. When implementing a code of the bisection method, it is important to consider this potential issue by either verifying consistency among the indicated signs of F and rejecting further bisection beyond a certain point or by utilizing a calculated error estimate for the computed values of F.

Theorem 1. *[15] Suppose that F is a continuous function on $[a, b]$ such that $F(a) \cdot F(b) < 0$. The bisection method produces a sequence $\{x_k\}$ that approximates the root x^* of F with*

$$| x_k - x^* | \leq \frac{b-a}{2^k}, \quad \textit{where} \quad k \geq 1.$$

Proof. Let $[a, b]$ be the initial interval and x^* be a root of F in this interval. For each $k \geq 1$, we define

$$x_k = \frac{b_k + a_k}{2}.$$

Different signs of F at a_k and b_k guarantee that at least one root exists in the interval $[a_k, b_k]$. Now, recursively producing values a_k and b_k as in Algorithm 1 for each $k \geq 1$, we have

$$b_k - a_k = \frac{b-a}{2^{k-1}}.$$

Since the actual solution $F(x^*) = 0$ satisfies $x^* \in [a_k, b_k]$ for all $k \geq 1$, we have

$$| x_k - x^* | \leq \frac{1}{2}(b_k - a_k) = \frac{b-a}{2^k}.$$

Therefore, the sequence $\{x_k\}$ converges to x^* with a rate of convergence $O(1/2^k)$, that is,

$$x_k = x^* + O(1/2^k).$$

□

If we need to know the number of steps for the desired accuracy ϵ we can calculate it from

$$\frac{b-a}{2^k} \leq \epsilon,$$

which simplifies to:

$$k \geq \left\lceil \frac{\log((b-a)/\epsilon)}{\log(2)} \right\rceil. \tag{2.5}$$

Inequality (2.5) gives the minimum number of iterations required to achieve the accuracy ϵ. For example, if $b - a = 1$ and $\epsilon = 0.001$, then from (2.5), we can calculate that $k \geq 10$.

2.2.2 Newton-Raphson method

Newton's method is one of the simplest and widely used iterative methods to solve equations. It is also known as the *Newton-Raphson method*, and it is faster and more efficient than the bisection method.

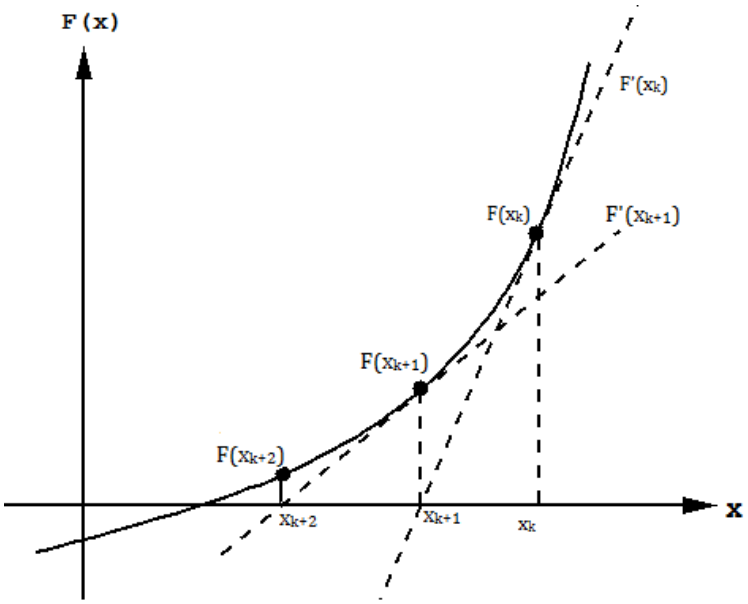

Figure 2.3: Tangent at point x_k to get new iteration point x_{k+1} for Newton-Raphson method.

Newton-Raphson method requires the evaluation of both the function F and its derivative, denoted by F', at arbitrary points. We assume that F has at least one (real) root and start with an initial guess, say x_0, sufficiently close to the exact root x^*. The next iterate x_1 is given by the point at which the tangent line to F at $(x_0, F(x_0))$ crosses the x-axis. We denote by T the tangent line to F at point x_0, such that,

$$T(x) = F(x_0) + F'(x_0)(x - x_0). \tag{2.6}$$

The new iterate x_1 is the point where this tangent line intersects the x-axis. Thus we have

$$0 - F(x_0) = F'(x_0)(x_1 - x_0)$$

which gives

$$x_1 = x_0 - \frac{F(x_0)}{F'(x_0)}, \tag{2.7}$$

as the basic Newton's method. If x_{k+1} represents the next iterate, it corresponds to the point where the tangent line to F at $(x_k, F(x_k))$ intersects the x-axis as shown in figure 2.3. In general, Newton's method for finding a root is given by iterating (2.7) repeatedly, that is,

$$x_{k+1} = x_k - \frac{F(x_k)}{F'(x_k)} \quad \forall \quad k \geq 0 \tag{2.8}$$

provided $F'(x_k)$ is non-zero. A suitable way to terminate the method is using error tolerance $\epsilon > 0$ or the maximum number of iterations N.

Computational algorithm

Algorithm 2: Newton-Raphson method [6]

Input: Starting point x_0, tolerance $\epsilon > 0$, maximum number of iterations N

Output: Approximate solution x^*

1 Set $k = 0$

2 While $(k < N)$

Calculate $F(x_k)$ and $F'(x_k)$.

If $F'(x_k)$ is close to zero then

OUTPUT: ('Method failed to converge')

Go to step 4.

Else set $y_k = -F(x_k)/F'(x_k)$ and update $x_{k+1} = x_k + y_k$

Check,

If $|x_{k+1} - x_k| < \epsilon$ then update $x^* = x_{k+1}$ and go to step 4.

Else update iteration counter $k = k + 1$

3 End while

OUTPUT: ('Method failed to converge after N iterations')

4 Stop

Example with Matlab

Here, we take the same example $F(x) = x^5 - 3x^3 + 2.5x - 0.6 = 0$ that we used in the bisection method. Since the convergence and efficiency of Newton's method are highly dependent on starting points, we solved the equation using different starting points. As Tables 2.2 and 2.3 show, for the tolerance $\epsilon = 10^{-6}$ and maximum number of iterations $N = 50$, Newton's method converges in 7 steps when the starting point is 2, whereas it took 12 iterations when the starting point is 5. In both cases, we reached the same solution.

Iter	x_k	F(x_k)	F'(x_k)	|x_k - x_k-1|
0	2.000000	12.400000	46.500000	---
1	1.733333	3.756479	20.593432	0.266667
2	1.550922	1.058981	9.780522	0.182411
3	1.442647	0.248031	5.426538	0.108274
4	1.396940	0.033931	3.977652	0.045707
5	1.388410	0.001059	3.730647	0.008531
6	1.388126	0.000001	3.722554	0.000284
7	1.388126	0.000000	3.722545	0.000000

Table 2.2: Solution of $F(x) = x^5 - 3x^3 + 2.5x - 0.6 = 0$ using the Newton-Raphson method with the starting point $x_0 = 2$.

Iter	x_k	F(x_k)	F'(x_k)	|x_k - x_k-1|
0	5.000000	2761.900000	2902.500000	---
1	4.048441	897.985338	1198.131064	0.951559
2	3.298953	290.672260	496.760253	0.749488
3	2.713817	93.422598	207.418262	0.585136
4	2.263410	29.676061	87.619671	0.450407
5	1.924718	9.235302	37.777218	0.338692
6	1.680251	2.762131	16.944302	0.244467
7	1.517238	0.755232	8.278187	0.163012
8	1.426007	0.162367	4.874054	0.091232
9	1.392694	0.017305	3.853808	0.033313
10	1.388204	0.000290	3.724769	0.004490
11	1.388126	0.000000	3.722545	0.000078
12	1.388126	0.000000	3.722545	0.000000

Table 2.3: Solution of $F(x) = x^5 - 3x^3 + 2.5x - 0.6 = 0$ using the Newton-Raphson method with the starting point $x_0 = 5$.

Convergence

Sometimes, the Newton-Raphson method does not converge if the initial guess x_0 is not sufficiently close enough to the exact root x^*; in other words, it is not globally convergent. So, while choosing x_0, one should check that $F'(x_0)$ is not close enough

to zero. If $F'(x_0)$ is nearly zero, the tangent line becomes almost parallel to the x-axis, resulting in x_1 being significantly distant from x^*. Due to this, the method may diverge. Before proving the convergence theorem of the Newton-Raphson method, we need to prove the following lemma.

Lemma 1. *[44] For an open interval D, let $F : D \to \mathbb{R}$ be differentiable such that F' is Lipschitz continuous on D. Then for any $x, y \in D$*

$$|F(y) - F(x) - F'(x)(y - x)| \leq \frac{\gamma(y - x)^2}{2},$$

where $\gamma > 0$ is the Lipschitz constant of function F' on the domain D.

Proof. From calculus, we know that $F(y) - F(x) = \int_x^y F'(z)dz$. Equivalently, we can write

$$F(y) - F(x) - F'(x)(y - x) = \int_x^y [F'(z) - F'(x)]dz. \tag{2.9}$$

By the change of variables

$$z = x + t(y - x), \quad dz = dt(y - x),$$

equation (2.9) becomes

$$F(y) - F(x) - F'(x)(y - x) = \int_0^1 [F'(x + t(y - x)) - F'(x)](y - x)dt.$$

Further, by applying the triangle inequality in the above integral part and the Lipschitz continuity of F', we have

$$|F(y) - F(x) - F'(x)(y - x)| \leq |y - x| \int_0^1 \gamma|t(y - x)|dt = \frac{\gamma|y - x|^2}{2}.$$

□

Theorem 2. *[44] For an open interval D, let $F : D \to \mathbb{R}$ be differentiable, and F' be Lipschitz continuous and bounded from below on D. Assume that for some $\rho > 0$, we have $|F'(x)| \geq \rho$ for all $x \in D$. If $F(x) = 0$ has a solution $x^* \in D$ then there exists some $\eta > 0$ such that: if $|x_0 - x^*| < \eta$, then the sequence $\{x_k\}$ generated by*

$$x_{k+1} = x_k - \frac{F(x_k)}{F'(x_k)}$$

exists and converges quadratically to x^*, as

$$|x_{k+1} - x^*| \leq \frac{\gamma}{2\rho}|x_k - x^*|^2 \quad k = 0, 1, \ldots$$

where γ is the Lipschitz constant of function F' on the domain D.

Proof. Let $\tau \in (0, 1)$ and $\hat{\eta}$ be the largest radius of open interval around x^* that is contained in D. Define $\eta = \min\{\hat{\eta}, \tau(2\rho/\gamma)\}$ and select x_0 such that $|x_0 - x^*| \leq \eta$ and thus $x_0 \in D$. The Newton's method calculation (2.7) is of the form

$$x_1 = x_0 - \frac{F(x_0)}{F'(x_0)}.$$

From this, we can obtain

$$x_1 - x^* = x_0 - x^* - \frac{F(x_0)}{F'(x_0)} = x_0 - x^* - \frac{F(x_0) - F(x^*)}{F'(x_0)}$$
$$= \frac{1}{F'(x_0)}[F(x^*) - F(x_0) - F'(x_0)(x^* - x_0)].$$

Then from Lemma 1,

$$|x_1 - x^*| \leq \frac{\gamma}{2|F'(x_0)|}|x_0 - x^*|^2$$

and by the boundness assumption that $|F'(x)| \geq \rho$ for all $x \in D$, the above implies

$$|x_1 - x^*| \leq \frac{\gamma}{2\rho}|x_0 - x^*|^2.$$

Since $|x_0 - x^*| \leq \eta \leq \tau\frac{2\rho}{\gamma}$, we have $|x_1 - x^*| \leq \tau|x_0 - x^*| < \eta$ and $x_1 \in B(x^*; \eta)$. Similarly, we can prove the desired result by iterating on k. Thus, for all $k = 0, 1, ...$, we have $x_k \in D$ and

$$|x_{k+1} - x^*| \leq \frac{\gamma}{2\rho}|x_k - x^*|^2.$$

□

The condition to provide a non-zero lower bound for $F'(x)$ in D means that $F'(x)$ must be non-zero at each $x \in D$ for Newton's method to converge quadratically. If $F'(x) = 0$ for some $x \in D$, then F has multiple roots, and Newton's method converges linearly. Quadratic convergence is very fast, but unfortunately, it is local behaviour. If the initial guess is not close to x^*, there is no guarantee that $x_k \to x^*$ as $k \to \infty$. The local convergence rate of Newton's method is governed by the error in the approximation (2.6). Then (2.7) implies that $|x_1 - x_0| = O(|F(x_0)|)$. If we put x_1 instead of x in equation (2.6), then together with the Taylor series error bound, we have

$$F(x_1) - T(x_1) = O(|x_1 - x_0|^2)$$

and since $T(x_1) = 0$, implies that

$$|F(x_1)| = O(|F(x_0)|^2).$$

This means that there exists a constant $c > 0$ such that

$$|F(x_1)| \leq c|F(x_0)|^2.$$

The same reasoning can be applied to the general x_k. This means that the residual at the next iteration is roughly proportional to the square of the residual at the current iteration.

2.2.3 Secant method

The secant method serves as an alternative to Newton's method when the derivative F' is not explicitly available. This approach is often referred to as a derivative-free

method. It also converges faster than the other derivative-free methods, such as bisection or false position methods [15]. The secant method is based on approximating the function F using a straight line, called a secant, that connects two points on the graph of the function. Thus, the method starts with two initial guesses, x_0 and x_1, which are ideally close to the actual root x^*.

The first iterate x_2 is obtained as the intersection of the secant line passing through $(x_0, F(x_0))$ and $(x_1, F(x_1))$ with the x-axis. In slope–intercept form, the equation of this line is

$$y = \frac{F(x_1) - F(x_0)}{x_1 - x_0}(x - x_1) + F(x_1) \tag{2.10}$$

The root of this linear function, that is the value of x such that $y = 0$ is

$$x = x_1 - F(x_1)\frac{x_1 - x_0}{F(x_1) - F(x_2)}.$$

We then use this new value of x as x_2. Similarly, x_3 is determined to be the point where the secant line passing through $(x_1, F(x_1))$ and $(x_2, F(x_2))$ intersects with the x-axis and so forth. Writing this process in iterative form, with setting $x = x_{k+1}$ and $y = 0$ in the equation (2.10) and using secant line from $(x_{k-1}, F(x_{k-1}))$ and $(x_k, F(x_k))$, we obtain

$$x_{k+1} = x_k - \frac{F(x_k)(x_k - x_{k-1})}{F(x_k) - F(x_{k-1})} \quad \forall \quad k \geq 1, \tag{2.11}$$

where $F(x_k) - F(x_{k-1}) \neq 0$. Equation (2.11) is also obtained if in the formula of the Newton-Raphson method (2.8), the derivative $F'(x_k)$ is replaced with a finite difference approximation based on the two most recent iterates, as follows:

$$F'(x_k) \approx \frac{F(x_k) - F(x_{k-1})}{x_k - x_{k-1}}. \tag{2.12}$$

Computational algorithm

Algorithm 3: Secant Method [6]

Input: Starting points x_0, x_1, tolerance $\epsilon > 0$, maximum number of iterations N.

Output: Approximate solution x^*

1 Set $k = 1$

2 While $(k \leq N)$

Calculate $F(x_{k-1})$ and $F(x_k)$.

If $F(x_{k-1}) = F(x_k)$ then

OUTPUT: ('Method failed to converge')

Go to step 4.

Else calculate $x_{k+1} = x_k - \frac{F(x_k)(x_k - x_{k-1})}{F(x_k) - F(x_{k-1})}$.

Check,

If $|x_{k+1} - x_k| < \epsilon$ then update $x^* = x_{k+1}$ and go to step 4.

Update iteration counter $k = k + 1$.

End while

OUTPUT: ('Method failed to converge after N iterations')

3 Stop

Example with Matlab

We take the same example $F(x) = x^5 - 3x^3 + 2.5x - 0.6 = 0$ with $x_0 = 1$, $x_1 = 2$, the tolerance $\epsilon = 10^{-5}$ and the maximum number of iterations N=50. The secant method took 6 iterations to reach the solution $x^* = 0.93$. Note that this solution is different from the Newton-Raphson method 2.3. Computation results are shown in Table 2.4.

Iter	x_k-1	x_k	F(x_k)	|x_k-x_k-1|
1	1.000000	2.000000	12.400000	1.000000
2	2.000000	1.008000	-0.111932	0.992000
3	1.008000	1.016874	-0.124993	0.008874
4	1.016874	0.931943	0.004616	0.084931
5	0.931943	0.934968	-0.000060	0.003025
6	0.934968	0.934929	0.000000	0.000039
7	0.934929	0.934929	0.000000	0.000000

Table 2.4: Solution of $F(x) = x^5 - 3x^3 + 2.5x - 0.6 = 0$ with $x_0 = 1$ and $x_1 = 2$ using the secant method

Convergence

The secant method has a convergence rate that is similar to Newton's method, even though the secant method is not as fast as Newton's method. Specifically, the secant method exhibits superlinear convergence under some specific assumptions.

Although, same as in Newton's method, there is no universal guarantee that the secant method converges from any initial point, typically, the secant method is popular because secant iteration has less function evaluations than a single Newton iteration. The search for the solution in the secant method is similar to the bisection method, in a sense that after each iteration, one of the previous two interval endpoints from the previous iteration is replaced by the new root estimate.

Theorem 3. *[35] Let D be an open interval, $F : D \to \mathbb{R}$ be differentiable, and x_k be the sequence produced by the secant method. Assume that the sequence converges to a root of $F(x) = 0$. This implies that $x_k \to x^*$ and $F(x^*) = 0$. Moreover, assume the root x^* is regular: $F'(x^*) \neq 0$. Then, the sequence converges to the root x^* superlinearly.*

Proof. See Lemmas 1.2, 2.1, and 2.2 in [35] for detailed proof.

□

2.3 Solving a system of nonlinear equations with several variables

Systems of nonlinear equations can often be approximated by linear systems. When this is not possible or yields unsatisfactory results, the problem must be tackled directly. Computational methods for finding the solution of the system of nonlinear equations are an important part of engineering, physics, chemistry, computer science, and economics [56].

In Subsection 2.3.1, we discuss Newton's method for a system of nonlinear equations as the multi-dimensional extension of the method discussed in Subsection 2.2.2. It is one of the basic methods for solving systems of nonlinear equations. This method converges quadratically to a root if we have a good initial guess. It may fail to converge if the initial guess is far from the root. Two drawbacks of this method are its computational cost and slow convergence in the case of a bad initial guess.

Subsection 2.3.2 presents the inexact Newton method, which improves Newton's method by inducing a special forcing term to control the level of accuracy. In addition, limitations of the inexact Newton method are discussed.

In Subsection 2.3.3, we discuss the quasi-Newton method, which is a more sophisticated variant of Newton's method, trying to improve the computational cost. Calculating and inverting the true Jacobian matrix is computationally expensive, especially for high-dimensional problems. Approximating the Jacobian reduces this burden significantly. In quasi-Newton methods, the Jacobian is approximated by a matrix B_k, which is updated iteratively at each step. The approximation matrix B_k or its inverse B_k^{-1} may become ill-conditioned, reducing numerical stability and potentially slowing convergence. The Sherman-Morrison formula is introduced because it provides an efficient way to update the inverse approximation B_k^{-1} directly without explicitly inverting B_k. This removes the need to compute B_k and its inversion at each iteration. On the other hand, quasi-Newton methods have some disadvantages since they do not converge quadratically like Newton's method. Another drawback of this method is that it does not have the self-correcting property

of Newton's method. Thus, it does not correct itself for rounding errors with consecutive iterations. This might cause a slight inaccuracy in iterations compared to Newton's method.

In the following, we consider the system of m nonlinear equations (2.1) with n variables. If $m < n$, then the system (2.1) has fewer equations than unknown variables. Such a system is called an *underdetermined system* [22]. In the case $m > n$, (2.1) has more equations than unknown variables, and the system is considered *overdetermined system* [22]. Lastly, for the case $m = n$, (2.1) has the same number of equations as the number of unknown variables, and the system is called *determined system*.

In the systems of linear equations, underdetermined systems either have no solution (so-called inconsistent systems) or an infinite number of solutions. Overdetermined systems may sometimes have solutions if one or more equations are linear combinations of others. Determined systems, on the other hand, possess a unique solution if the rank of the system (i.e., the rank of the augmented matrix) equals the rank of the constraint matrix and matches the number of variables.

In this subsection, we focus exclusively on determined systems. Next, to solve the system (2.1), we present the following methods, which extend the techniques discussed in the previous section to higher dimensions.

2.3.1 Newton's method

Newton's method is a powerful numerical technique for solving systems of n nonlinear equations

$$\boldsymbol{F}(\boldsymbol{x}) = \begin{pmatrix} F_1(\boldsymbol{x}) \\ F_2(\boldsymbol{x}) \\ \vdots \\ F_n(\boldsymbol{x}) \end{pmatrix} = \boldsymbol{0}, \tag{2.13}$$

where

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n$$

is the vector of variables and $F_1, F_2, ..., F_n : \mathbb{R}^n \to \mathbb{R}$ are twice continuously differentiable functions. Newton's method iteratively refines an initial guess $\boldsymbol{x}_0$ to approximate the solution $\boldsymbol{x}^*$, which satisfies $\boldsymbol{F}(\boldsymbol{x}^*) = \boldsymbol{0}$.

Newton's method for a system of nonlinear equations is an extension of Newton's method 2.2.2 with one variable. It aims to iteratively approximate (or correct) the solution by linearizing the system at each iteration, using the Jacobian matrix to guide the update steps toward the root. For a small correction $\boldsymbol{z}_k \in \mathbb{R}^n$ at the current iterate $\boldsymbol{x}_k \in \mathbb{R}^n$, we approximate $\boldsymbol{F}$ using a Taylor expansion

$$\boldsymbol{T}(\boldsymbol{x}_k + \boldsymbol{z}_k) \approx \boldsymbol{F}(\boldsymbol{x}_k) + J(\boldsymbol{x}_k)\boldsymbol{z}_k$$

where $J(\boldsymbol{x}_k)$ is the Jacobian matrix of $\boldsymbol{F}$ at $\boldsymbol{x}_k$.

At the root $\boldsymbol{x}^*$ we should have $\boldsymbol{F}(\boldsymbol{x}^*) = \boldsymbol{0}$. Therefore, utilizing this condition in the linear approximation gives

$$\boldsymbol{F}(\boldsymbol{x}_k) + J(\boldsymbol{x}_k)\boldsymbol{z}_k = \boldsymbol{0}. \tag{2.14}$$

If $J(\boldsymbol{x}_k)$ is nonsingular, equation (2.14), yields

$$\boldsymbol{z}_k = -J(\boldsymbol{x}_k)^{-1}\boldsymbol{F}(\boldsymbol{x}_k). \tag{2.15}$$

The next iterate is then given by

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{z}_k \text{ for } k = 0, 1, \ldots \tag{2.16}$$

which is the natural generalization of the one-dimensional formula (2.8). In practical computations, evaluating $J(\boldsymbol{x}_k)^{-1}$ is typically more costly than solving (2.14). When $\|\boldsymbol{x}_k - \boldsymbol{x}^*\|$ is small enough, we can write the Taylor approximation error bound in terms of norms:

$$\|\boldsymbol{F}(\boldsymbol{x}_{k+1}) - \boldsymbol{T}(\boldsymbol{x}_{k+1})\| = \|\boldsymbol{F}(\boldsymbol{x}_k + \boldsymbol{z}_k) - (\boldsymbol{F}(\boldsymbol{x}_k) + J(\boldsymbol{x}_k)\boldsymbol{z}_k)\| = O(\|\boldsymbol{z}_k\|^2). \tag{2.17}$$

The last equality holds since every component of the vector $\boldsymbol{F}(\boldsymbol{x}_{k+1}) - \boldsymbol{T}(\boldsymbol{x}_{k+1})$ has Taylor's approximation error bound equal to $O(\|\boldsymbol{z}_k\|^2)$ (see Lemma 1). We also see from (2.15) that,

$$\|\boldsymbol{z}_k\| = \| - J(\boldsymbol{x}_k)^{-1}\boldsymbol{F}(\boldsymbol{x}_k)\| \leq \left\|\left|-J(\boldsymbol{x}_k)^{-1}\right|\right\| \|\boldsymbol{F}(\boldsymbol{x}_k)\| \leq c_1\|\boldsymbol{F}(\boldsymbol{x}_k)\|, \tag{2.18}$$

where

$$c_1 = \sup_{\boldsymbol{x} \in B(\boldsymbol{x}_k;\delta)} \left\|\left|-J(\boldsymbol{x})^{-1}\right|\right\| \tag{2.19}$$

is a bound relating to Newton's step and some $\delta > 0$ defines the neighbourhood of $\boldsymbol{x}_k$ where Jacobian $J(\boldsymbol{x})$ is nonsingular. Inequalities (2.17) and (2.18) together with $\boldsymbol{T}(\boldsymbol{x}_{k+1}) = 0$ imply that

$$\|\boldsymbol{F}(\boldsymbol{x}_{k+1})\| \leq c\|\boldsymbol{F}(\boldsymbol{x}_k)\|^2,$$

where c combines the bound c_1 relating to Newton's step with a bound on the second-order Taylor remainder. This shows that the residual $\|\boldsymbol{F}(\boldsymbol{x}_k)\|$ converges to 0 at the quadratic rate. Showing quadratic convergence typically requires nonsingularity and continuity of the Jacobian, and a sufficiently close starting point assumption.

Computational algorithm

Algorithm 4: Newton's Method [6]

Input: Starting point $\boldsymbol{x}_0$, tolerance $\epsilon > 0$, maximum number of iterations N

Output: Approximate solution $\boldsymbol{x}^*$

1 Set $k = 0$.

2 While $(k < N)$

Calculate $\boldsymbol{F}(\boldsymbol{x}_k)$ and $J(\boldsymbol{x}_k)$.

If $J(\boldsymbol{x}_k)$ is singular

OUTPUT: ('Method failed because of singularity')

Go to step 4.

Let $\boldsymbol{z}_k = -J(\boldsymbol{x}_k)^{-1}\boldsymbol{F}(\boldsymbol{x}_k)$

Set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{z}_k$

Check,

If $\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\| < \epsilon$ then update $\boldsymbol{x}^* = \boldsymbol{x}_{k+1}$ and go to step 4.

Update iteration counter $k = k + 1$.

3 End while

OUTPUT: ('Method failed to converge after N iterations')

4 Stop

Example with Matlab

The system of equations

$$\boldsymbol{F}(x_1, x_2) = \begin{pmatrix} x_1^3 + 3x_2^2 - 21 \\ x_1^2 + 2x_2 + 2 \end{pmatrix} = \boldsymbol{0} \tag{2.20}$$

is solved using Newton's method with the different starting points, tolerance $\epsilon = 10^{-6}$, and maximum number of iterations $N = 50$. Results are presented in Tables 2.5-2.6. In Table 2.5 starting point is $(1, 1)^T$ and we reached solution $(1.64, -2.34)^T$ in 18 iterations while in Table 2.6 starting point is $(1, -1)^T$ and we reached solution $(1.64, -2.34)^T$ in 6 iterations. In both cases, we reached the same solution but with different numbers of iterations.

Iter	x1_k	x2_k	F1(x1_k,x2_k)	F2(x1_k,x2_k)	||x_{k+1}-x_k||
0	1.000000	1.000000	-17.000000	5.000000	13.433995
1	-9.666667	9.166667	-672.212963	113.777778	12.543874
2	-4.985507	-2.471010	-126.598487	21.913258	290.978482
3	-60.096303	-288.182887	32085.383020	3037.199904	199.542475
4	-31.540660	-90.694218	-6721.741858	815.424782	72.471060
5	-16.367318	-19.829387	-3226.011950	230.230311	35.911223
6	-7.212497	14.895319	269.416681	83.810746	11.449859
7	-2.871367	4.300326	10.804703	18.845403	3.559268
8	-0.520459	1.627945	-13.190365	5.526768	6.859687
9	6.294342	2.411389	245.818358	46.441526	43.761046
10	9.538504	-41.229240	5946.393114	10.524587	21.179721
11	6.785212	-20.229241	1519.051591	7.580620	10.379539
12	4.727239	-10.055767	387.993942	4.235253	5.021989
13	3.263078	-5.251955	96.493300	2.143767	2.295707
14	2.296454	-3.169670	21.251240	0.934361	0.860842
15	1.789924	-2.473627	3.091093	0.256573	0.182183
16	1.651850	-2.354773	0.142117	0.019064	0.010090
17	1.643070	-2.349801	0.000456	0.000077	0.000035
18	1.643038	-2.349787	0.000000	0.000000	0.000000
19	1.643038	-2.349787	0.000000	0.000000	0.000000

Table 2.5: Solution of a system of nonlinear equations using Newton's method with starting point $(1, 1)^T$.

Iter	x1_k	x2_k	F1(x1_k,x2_k)	F2(x1_k,x2_k)	||x_{k+1}-x_k||
0	1.000000	-1.000000	-17.000000	1.000000	2.577802
1	2.555556	-3.055556	23.699246	2.419753	0.885747
2	1.865049	-2.500805	4.249473	0.476799	0.248053
3	1.661337	-2.359271	0.283833	0.041499	0.020464
4	1.643173	-2.349844	0.001905	0.000330	0.000147
5	1.643038	-2.349787	0.000000	0.000000	0.000000
6	1.643038	-2.349787	0.000000	0.000000	0.000000

Table 2.6: Solution of a system of nonlinear equations using Newton's method with starting point $(1, -1)^T$.

Convergence

In practice, Newton's method can be challenging to use because of its lack of robustness, particularly its sensitivity to the choice of the initial guess and potential divergence. The method may demand some ingenuity to find a starting point $\boldsymbol{x}_0$ close enough to $\boldsymbol{x}^*$ to obtain convergence. It is often difficult to know whether a system of nonlinear equations even has a solution.

While Newton's method can suffer from extreme ill-conditioning of the function $\boldsymbol{F}$, it exhibits some robustness due to its affine invariance. In other words, the method is invariant under affine transformations, such as $\boldsymbol{G}(\boldsymbol{y}) = A\boldsymbol{F}(B\boldsymbol{y})$, where A and B are invertible $n \times n$ matrices. This means that Newton's method works

identically with the original equations $\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{0}$ and the transformed equations $\boldsymbol{G}(\boldsymbol{y}) = \boldsymbol{0}$. We now state the following results, which are analogous to the one-dimensional case (see subsection 2.2.2). For the proof of Lemma 2 and the following theorem, we refer to [25].

Lemma 2. *Let $\boldsymbol{F} : \mathbb{R}^n \to \mathbb{R}^n$ be continuously differentiable in an open convex set $D \subset \mathbb{R}^n$, and let J be Lipschitz continuous on D. Then for any $\boldsymbol{x}, \boldsymbol{x} + \boldsymbol{p} \in D$ we have*

$$\|\boldsymbol{F}(\boldsymbol{x} + \boldsymbol{p}) - \boldsymbol{F}(\boldsymbol{x}) - J(\boldsymbol{x})\boldsymbol{p}\| \leq \frac{\gamma}{2}\|\boldsymbol{p}\|^2,$$

where $\gamma > 0$ is the Lipschitz constant of J on D.

The local quadratic convergence of Newton's method for systems of nonlinear equations is proved in the following result. The techniques of the proof are similar to those in the one-dimensional case (see subsection 2.2.2).

Theorem 4. *Let $\boldsymbol{F} : \mathbb{R}^n \to \mathbb{R}^n$ be continuously differentiable in a open convex set $D \subset \mathbb{R}^n$. Assume that there exists an exact solution $\boldsymbol{x}^* \in \mathbb{R}^n$ and $r, \beta > 0$, such that $B(\boldsymbol{x}^*; r) \subset D$, $\boldsymbol{F}(\boldsymbol{x}^*) = \boldsymbol{0}$, $J(\boldsymbol{x}^*)^{-1}$ exist with $\|\|J(\boldsymbol{x}^*)^{-1}\|\| \leq \beta$, and J is Lipschitz continuous with constant $\gamma > 0$ on $B(\boldsymbol{x}^*; r)$. Then there exists $\epsilon > 0$ such that for all $\boldsymbol{x}_0 \in B(\boldsymbol{x}^*; \epsilon)$ the sequence $\{\boldsymbol{x}_k\}$ generated by*

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - J(\boldsymbol{x}_k)^{-1}\boldsymbol{F}(\boldsymbol{x}_k), \quad k = 0, 1, \ldots$$

is well defined and quadratically converges to $\boldsymbol{x}^*$,i.e.

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}^*\| \leq \beta\gamma\|\boldsymbol{x}_k - \boldsymbol{x}^*\|^2, \quad k = 0, 1, \ldots \tag{2.21}$$

Proof. We need to choose $\epsilon > 0$ such that $J(\boldsymbol{x}_0)$ is nonsingular for any $\boldsymbol{x}_0 \in B(\boldsymbol{x}^*; \epsilon)$. Let

$$\epsilon = \min\Big\{r, \frac{1}{2\beta\gamma}\Big\}. \tag{2.22}$$

We first prove that $J(\boldsymbol{x}_0)$ is nonsingular. From $\|\boldsymbol{x}_0 - \boldsymbol{x}^*\| \leq \epsilon$, the Lipschitz continuity of J at $\boldsymbol{x}^*$ and (2.22), it follows that

$$\begin{aligned}\Big\|\Big|J(\boldsymbol{x}^*)^{-1}[J(\boldsymbol{x}_0) - J(\boldsymbol{x}^*)]\Big\|\Big| &\leq \Big\|\Big|J(\boldsymbol{x}^*)^{-1}\Big\|\Big|\,\|\|J(\boldsymbol{x}_0) - J(\boldsymbol{x}^*)\|\| \\ &\leq \beta\gamma\|\boldsymbol{x}_0 - \boldsymbol{x}^*\| \leq \beta\gamma\epsilon \leq \frac{1}{2}.\end{aligned}$$

Thus by Lemma 3.1.20 in [25], $J(\boldsymbol{x}_0)$ is nonsingular and

$$\begin{aligned}\Big\|\Big|J(\boldsymbol{x}_0)^{-1}\Big\|\Big| &\leq \frac{\|\|J(\boldsymbol{x}^*)^{-1}\|\|}{1 - \|\|J(\boldsymbol{x}^*)^{-1}[J(\boldsymbol{x}_0) - J(\boldsymbol{x}^*)]\|\|} \\ &\leq 2\Big\|\Big|J(\boldsymbol{x}^*)^{-1}\Big\|\Big| \leq 2\beta.\end{aligned}$$

Therefore $\boldsymbol{x}_1$ is well defined and

$$\begin{aligned}\boldsymbol{x}_1 - \boldsymbol{x}^* =& \boldsymbol{x}_0 - \boldsymbol{x}^* - J(\boldsymbol{x}_0)^{-1}\boldsymbol{F}(\boldsymbol{x}_0) \\ =& \boldsymbol{x}_0 - \boldsymbol{x}^* - J(\boldsymbol{x}_0)^{-1}[\boldsymbol{F}(\boldsymbol{x}_0) - \boldsymbol{F}(\boldsymbol{x}^*)] \\ =& J(\boldsymbol{x}_0)^{-1}[\boldsymbol{F}(\boldsymbol{x}^*) - \boldsymbol{F}(\boldsymbol{x}_0) - J(\boldsymbol{x}_0)(\boldsymbol{x}^* - \boldsymbol{x}_0)].\end{aligned}$$

Then, using Lemma 2

$$\begin{aligned}\|\boldsymbol{x}_1 - \boldsymbol{x}^*\| &\leq \left\|J(\boldsymbol{x}_0)^{-1}\right\| \|\boldsymbol{F}(\boldsymbol{x}^*) - \boldsymbol{F}(\boldsymbol{x}_0) - J(\boldsymbol{x}_0)(\boldsymbol{x}^* - \boldsymbol{x}_0)\| \\ &\leq 2\beta\frac{\gamma}{2}\|\boldsymbol{x}_0 - \boldsymbol{x}^*\|^2 = \beta\gamma\|\boldsymbol{x}_0 - \boldsymbol{x}^*\|^2\end{aligned}$$

which shows that the above inequality (2.21) holds for $k = 0$. Since $\|\boldsymbol{x}_0 - \boldsymbol{x}^*\| \leq 1/(2\beta\gamma)$, we have

$$\|\boldsymbol{x}_1 - \boldsymbol{x}^*\| \leq \frac{1}{2}\|\boldsymbol{x}_0 - \boldsymbol{x}^*\| < \epsilon.$$

The inequality (2.21) holds similarly for the remaining $k = 1, 2, \ldots$ and induction steps can prove it. □

2.3.2 Inexact Newton methods

One drawback of Newton's method is the necessity to solve the Newton equation (2.14): $J(\boldsymbol{x}_k)\boldsymbol{z}_k = -\boldsymbol{F}(\boldsymbol{x}_k)$ at each iteration. Computing the exact solution using a direct solver can be expensive if the number of unknowns is large [23] and may not be justified when $\boldsymbol{x}_k$ is far from $\boldsymbol{x}^*$. Therefore, it seems reasonable to use an iterative method and to solve (2.14) only approximately. For this purpose, we can consider the class of inexact Newton methods that compute an approximate solution to the Newton equation.

To be precise, an inexact Newton method is any method that, given an initial guess $\boldsymbol{x}_0$, generates a sequence $\{\boldsymbol{x}_k\}$ of approximations to $\boldsymbol{x}^*$ as follows:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{z}_k \tag{2.23}$$

where $\boldsymbol{z}_k$ satisfies

$$J(\boldsymbol{x}_k)\boldsymbol{z}_k = -\boldsymbol{F}(\boldsymbol{x}_k) + \boldsymbol{r}_k.$$

Above $\boldsymbol{r}_k$ is the residual and the approximate solution $\boldsymbol{z}_k$ is determined such a way that

$$||\boldsymbol{r}_k||/||\boldsymbol{F}(\boldsymbol{x}_k)|| \leq \eta_k$$

where the non-negative forcing sequence $\{\eta_k\}$ [2] is used to control the level of accuracy. This means that when we solve the Newton equation with the selected iterative method, we stop as soon as this requirement is fulfilled.

Here, η_k may depend on $\boldsymbol{x}_k$, and $\eta_k = 0$ directly yields the classical Newton's method. The convergence of the inexact Newton method depends on the choice of the forcing terms η_k. The inexact method is locally convergent under the weak assumption that the forcing sequence $\{\eta_k\}$ is uniformly less than one [23]. Almost the same convergence speed as Newton's method can be attained with the inexact Newton method simply by setting the terms η_k small enough [4].

2.3.3 Quasi-Newton methods

A significant drawback of using Newton's method to solve a system of nonlinear equations, as presented in equation (2.1), lies in the substantial computational cost required at each iteration. Specifically, each iteration necessitates the evaluation of the partial derivatives of $\boldsymbol{F}$ at $\boldsymbol{x}$, as well as solving a system of linear equations involving the resulting Jacobian matrix. In many cases, obtaining exact evaluations of the partial derivatives can be inconvenient or, in some situations, impossible. While software packages such as Matlab and Mathematica provide predefined solvers and functions, the challenge of computing the Jacobian matrix remains.

Broyden's method [13] is a quasi-Newton method designed to address this issue and efficiently solve nonlinear equations $\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{0}$. Unlike Newton's method, which requires the computation of the Jacobian matrix and its inverse at each iteration, Broyden's method seeks to reduce the computational burden by calculating the Jacobian only once, at the first iteration. For subsequent iterations, rank-one updates are performed instead of recalculating the Jacobian. The quasi-Newton method can be viewed as a generalization of the secant method, replacing the derivative with the approximate Jacobian B_k at the iteration $k \geq 1$, which is determined using the secant equation (2.11)

$$B_k(\boldsymbol{x}_k - \boldsymbol{x}_{k-1}) \backsimeq \boldsymbol{F}(\boldsymbol{x}_k) - \boldsymbol{F}(\boldsymbol{x}_{k-1}).$$

When the exact evaluation of the Jacobian is not feasible, alternative techniques, such as finite difference approximations, can be used to approximate the partial derivatives at $\boldsymbol{x} \in \mathbb{R}^n$, such as

$$\frac{\partial}{\partial x_k}\boldsymbol{F}(\boldsymbol{x}) \approx \frac{\boldsymbol{F}(\boldsymbol{x} + \boldsymbol{e}_k h) - \boldsymbol{F}(\boldsymbol{x})}{h}$$

where h is a small positive number and the vector $\boldsymbol{e}_k \in \mathbb{R}^n$ is the vector with 1 at the k^{th} coordinate and 0 elsewhere. This strategy has led to the development of quasi-Newton methods. During the first iteration $k = 0$, the method uses Jacobian $J(\boldsymbol{x}_0)$ at the initial solution $\boldsymbol{x}_0$ to provide $\boldsymbol{x}_1$. However, as the method progresses, the Jacobian matrix $J(\boldsymbol{x}_1)$ in Newton's method is replaced with an approximate Jacobian B_1 that satisfies the condition:

$$B_1(\boldsymbol{x}_1 - \boldsymbol{x}_0) = \boldsymbol{F}(\boldsymbol{x}_1) - \boldsymbol{F}(\boldsymbol{x}_0). \tag{2.24}$$

To define B_1 uniquely, we require that:

$$B_1\boldsymbol{s} = J(\boldsymbol{x}_0)\boldsymbol{s} \quad \text{whenever} \quad (\boldsymbol{x}_1 - \boldsymbol{x}_0)^T\boldsymbol{s} = 0. \tag{2.25}$$

This requirement is needed to guarantee that in directions $\boldsymbol{s}$ orthogonal to step $(\boldsymbol{x}_1 - \boldsymbol{x}_0)^T$, the new matrix B_1 should not change from the old matrix $J(\boldsymbol{x}_0)$. In the direction $(\boldsymbol{x}_1 - \boldsymbol{x}_0)^T$, we force B_1 to deviate from $J(\boldsymbol{x}_0)$ to match the new difference in $\boldsymbol{F}$. By (2.24) and (2.25) we can uniquely define

$$B_1 = J(\boldsymbol{x}_0) + \frac{\boldsymbol{F}(\boldsymbol{x}_1) - \boldsymbol{F}(\boldsymbol{x}_0) - J(\boldsymbol{x}_0)(\boldsymbol{x}_1 - \boldsymbol{x}_0)}{\| \boldsymbol{x}_1 - \boldsymbol{x}_0 \|^2}(\boldsymbol{x}_1 - \boldsymbol{x}_0)^T.$$

Now, we can replace $J(\boldsymbol{x}_1)$ in the first Newton's iteration with the approximated matrix B_1. The general update rule (2.15) becomes:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - B_k^{-1}\boldsymbol{F}(\boldsymbol{x}_k) \text{ for all } k \geq 0 \tag{2.26}$$

which is defined as the generalization of (2.2). The approximation of the Jacobian matrix B_k can be calculated at each iteration by the general formula

$$B_k = B_{k-1} + \frac{\boldsymbol{y}_{k-1} - B_{k-1}\boldsymbol{z}_{k-1}}{\| \boldsymbol{z}_{k-1} \|^2}\boldsymbol{z}_{k-1}^T, \quad \text{for all } k \geq 1 \tag{2.27}$$

where $\boldsymbol{y}_{k-1} = \boldsymbol{F}(\boldsymbol{x}_k) - \boldsymbol{F}(\boldsymbol{x}_{k-1})$ and $\boldsymbol{z}_{k-1} = \boldsymbol{x}_k - \boldsymbol{x}_{k-1}$.

As already seen, quasi-Newton methods use matrices B_k to approximate Jacobian matrices J_k. In many of these methods, the calculation of equation (2.26) does not require the computation of derivatives. Additionally, many of these methods use a straightforward formula to derive B_k^{-1} from B_{k-1}^{-1}, resulting in a relatively low computational cost. Broyden also suggested the use of the Sherman-Morrison formula to update the inverse of the Jacobian approximation (2.27) directly. The Sherman-Morrison formula is specifically suitable for methods that involve rank-one updates, like Broyden's method. A rank-one update refers to modifying a matrix by adding or subtracting a matrix of rank one.

Broyden referred to equation (2.26) as "the fundamental equations of quasi-Newton methods" in [14]. Following Dennis and Schnabel [25], most authors have adopted the term "quasi-Newton method" to encompass all methods of the form given by equation (2.26).

Sherman-Morrison Formula

Let B be an invertible $n \times n$ matrix and let $\boldsymbol{x}$ and $\boldsymbol{y}$ be n-dimensional column vectors. If B^{-1} is the inverse of B, the Sherman-Morrison formula gives that the inverse of $B + \boldsymbol{x}\boldsymbol{y}^T$ exists, provided that $1 + \boldsymbol{y}^T B^{-1}\boldsymbol{x} \neq 0$ and

$$(B + \boldsymbol{x}\boldsymbol{y}^T)^{-1} = B^{-1} - \frac{B^{-1}\boldsymbol{x}\boldsymbol{y}^T B^{-1}}{1 + \boldsymbol{y}^T B^{-1}\boldsymbol{x}}.$$

The Sherman-Morrison formula allows for to calculation of B_k^{-1} directly from B_{k-1}^{-1}, eliminating the need for a matrix inversion with each iteration. Taking the inverse of equation (2.27) on both sides, we get

$$\begin{aligned}
B_k^{-1} =& \left(B_{k-1} + \frac{\boldsymbol{y}_{k-1} - B_{k-1}\boldsymbol{z}_{k-1}}{\| \boldsymbol{z}_{k-1} \|^2}\boldsymbol{z}_{k-1}^T \right)^{-1} \\
=& B_{k-1}^{-1} - \frac{B_{k-1}^{-1}\left(\frac{\boldsymbol{y}_{k-1} - B_{k-1}\boldsymbol{z}_{k-1}}{\|\boldsymbol{z}_{k-1}\|^2}\boldsymbol{z}_{k-1}^T \right) B_{k-1}^{-1}}{1 + \boldsymbol{z}_{k-1}^T B_{k-1}^{-1}\left(\frac{\boldsymbol{y}_{k-1} - B_{k-1}\boldsymbol{z}_{k-1}}{\|\boldsymbol{z}_{k-1}\|^2} \right)} \\
=& B_{k-1}^{-1} - \frac{(B_{k-1}^{-1}\boldsymbol{y}_{k-1} - \boldsymbol{z}_{k-1})\boldsymbol{z}_{k-1}^T B_{k-1}^{-1}}{\| \boldsymbol{z}_{k-1} \|^2 + \boldsymbol{z}_{k-1}^T B_{k-1}^{-1}\boldsymbol{y}_{k-1} - \| \boldsymbol{z}_{k-1} \|^2} \\
=& B_{k-1}^{-1} + \frac{(\boldsymbol{z}_{k-1} - B_{k-1}^{-1}\boldsymbol{y}_{k-1})\boldsymbol{z}_{k-1}^T B_{k-1}^{-1}}{\boldsymbol{z}_{k-1}^T B_{k-1}^{-1}\boldsymbol{y}_{k-1}}.
\end{aligned}$$

This computation involves only matrix-vector multiplications at each step and therefore requires only $O(n^2)$ arithmetic calculations [15]. The calculation of B_k is bypassed, as is the necessity of solving $\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{0}$.

Computational algorithm

Algorithm 5: The Broyden method [6]

Input: Starting point $\boldsymbol{x}_0$, tolerance $\epsilon > 0$, maximum number of iterations N.

Output: Approximate solution $\boldsymbol{x}^*$.

1 Set $k = 0$.
Calculate $\boldsymbol{u}_0 = \boldsymbol{F}(\boldsymbol{x}_0)$ and $B_0 = J(\boldsymbol{x}_0)$.
Set $\boldsymbol{z}_0 = B_0^{-1}\boldsymbol{u}_0$ and update $\boldsymbol{x}_1 = \boldsymbol{x}_0 + \boldsymbol{z}_0$ and $k = 1$.

2 While $(k < N)$
Calculate $\boldsymbol{u}_k = \boldsymbol{F}(\boldsymbol{x}_k)$
Set $\boldsymbol{y}_{k-1} = \boldsymbol{u}_k - \boldsymbol{u}_{k-1}$
Calculate $p_k = \boldsymbol{z}_{k-1}^T B_{k-1}^{-1}\boldsymbol{y}_{k-1}$
Update $B_k = B_{k-1} + \frac{1}{p_k}[(\boldsymbol{z}_{k-1} - B_{k-1}^{-1}\boldsymbol{y}_{k-1})\boldsymbol{z}_{k-1}^T B_{k-1}^{-1}]$
Calculate $\boldsymbol{z}_k = -B_k^{-1}\boldsymbol{u}_k$ and update $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{z}_k$.
If $\| \boldsymbol{z}_k \| < \epsilon$ then $\boldsymbol{x}^* = \boldsymbol{x}_{k+1}$ and go to step 4.
Update iteration counter $k = k + 1$.

3 End while
OUTPUT: ('Method failed to converge after N iterations')

4 Stop

Example with Matlab

The system of equations

$$\boldsymbol{F}(\boldsymbol{x}) = \begin{pmatrix} x_1^3 + 3x_2^2 - 21 \\ x_1^2 + 2x_2 + 2 \end{pmatrix} = \boldsymbol{0}, \tag{2.28}$$

is solved using the Broyden method with the starting point $(1, 1)^T$, tolerance $\epsilon = 10^{-6}$ and maximum number of iterations $N = 50$. The results are presented in the following Table 2.7. Broyden method reached the solution $(1.64, -2.34)^T$, which is the same one as obtained in Tables 2.5 and 2.6 with the Newton's method, but the number of iterations is higher than in the Newton's method.

Iter	x1_k	x2_k	F1(x1_k,x2_k)	F2(x1_k,x2_k)	‖x_k-x_k-1‖
0	1.00000	1.00000	-17.00000	5.00000	0.0000000
1	18.00000	-4.00000	5859.00000	318.00000	17.7200451
2	-0.71028	-5.01551	54.10767	-7.52652	18.7378214
3	1.76906	3.92917	30.85144	12.98791	9.2819372
4	0.35837	-0.67168	-19.60050	0.78506	4.8122604
5	0.16670	-1.60316	-13.28499	-1.17853	0.9509939
6	0.24676	-1.43320	-14.82275	-0.80552	0.1878685
7	0.29089	-1.44830	-14.68266	-0.81199	0.0466424
8	-5.92004	-0.13953	-228.42072	36.76783	6.3473261
9	0.05709	-1.85745	-10.64948	-1.71163	6.2191130
10	-0.31531	-2.24357	-5.93059	-2.38771	0.5364402
11	7.29615	3.16034	397.36510	61.55449	9.3346912
12	-0.16249	-2.00383	-8.95825	-1.98126	9.0719357
13	0.01735	-1.82668	-10.98970	-1.65306	0.2524425
14	2.99667	0.14783	5.97562	11.27565	3.5742096
15	0.61614	-1.59617	-13.12277	-0.81272	2.9510046
16	1.03740	-1.51444	-13.00296	0.04733	0.4291170
17	1.76906	-1.63513	-7.44267	1.85931	0.7415424
18	1.47235	-1.79216	-8.17271	0.58348	0.3357033
19	1.57632	-1.95894	-5.57085	0.56692	0.1965331
20	1.68258	-2.37955	0.75035	0.07198	0.4338291
21	1.64056	-2.34460	-0.09317	0.00224	0.0546645
22	1.64301	-2.34969	-0.00165	0.00010	0.0056508
23	1.64304	-2.34979	-0.00000	0.00000	0.0001040
Small denominator in Sherman-Morrison update, skipping B update.					
24	1.64304	-2.34979	-0.00000	0.00000	0.0000004

Table 2.7: Solution of a system of nonlinear equations using quasi-Newton method with starting point $(1, 1)^T$.

Convergence

Previously described Broyden's method has local superlinear convergence [14, 15, 24] to a solution under specific conditions, such as the smoothness and differentiability of the function $\boldsymbol{F}$, the non-singularity of the Jacobian at the solution $\boldsymbol{x}^*$, and an initial approximation $\boldsymbol{x}_0$ close to $\boldsymbol{x}^*$. The convergence is characterized by:

$$\lim_{k\to\infty} \frac{\|\boldsymbol{x}_{k+1} - \boldsymbol{x}^*\|}{\|\boldsymbol{x}_k - \boldsymbol{x}^*\|} = 0$$

where $\boldsymbol{x}_{k+1}$ and $\boldsymbol{x}_k$ are successive approximations of $\boldsymbol{x}^*$. However, superlinear convergence is slower than the quadratic convergence of Newton's method, potentially requiring more iterations to reach a solution. This trade-off underscores the importance of assumptions like a good initial guess and accurate Jacobian approximations for the method's success.

Chapter 3

Methods of Unconstrained Optimization

In an indirect optimization approach, the system of nonlinear equations (2.1) can be reformulated as the following unconstrained optimization problem

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}) := \frac{1}{2}\|\boldsymbol{F}(\boldsymbol{x})\|^2, \tag{3.1}$$

where $\|\cdot\|$ means the Euclidean norm. If the minimum $\boldsymbol{x}^*$ of (3.1) exists with $f(\boldsymbol{x}^*) = 0$, then it corresponds to the solution of (2.1). The necessary condition for $\boldsymbol{x}^*$ to be a minimum is that $\nabla f(\boldsymbol{x}^*) = \mathbf{0}$. Thus, the problem of finding the minimum of f can now be converted into solving a system of nonlinear equations

$$\nabla f(\boldsymbol{x}) = \mathbf{0}. \tag{3.2}$$

While the formulation (3.2) is conceptually appealing, solving the system directly is not always practical due to the nonlinearity, potential ill-conditioning, and complexity of the problem. Although (3.2) gives a necessary condition for a minimum, it does not guarantee that the solution is a global minimum. The unconstrained optimization problem may have multiple local minima, and local methods can be guaranteed to converge to a local minimum rather than a global one.

Suppose the objective function is

$$f(\boldsymbol{x}) = x_1^3 + x_2^2 + 5x_1^2x_2 \tag{3.3}$$

and the gradient of f is

$$\nabla f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial f(\boldsymbol{x})}{\partial x_1} \\ \frac{\partial f(\boldsymbol{x})}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 3x_1^2 + 10x_1x_2 \\ 2x_2 + 5x_1^2 \end{bmatrix}.$$

Setting this gradient to zero gives the system of nonlinear equations:

$$\begin{cases} 3x_1^2 + 10x_1x_2 = 0 \\ 2x_2 + 5x_1^2 = 0. \end{cases} \tag{3.4}$$

The condition (3.4) is necessary for identifying potential extremum points. However, it is not sufficient to determine whether a point is a minimum, maximum,

or something else, such as a saddle point. Additional tests, such as examining the second-order derivatives or using other optimization methods, are necessary to classify extremum points properly. Solving the system (3.4) would yield the unique real-valued optimal solution $\boldsymbol{x}^* = \mathbf{0}$ with $f(\boldsymbol{x}^*) = 0$ that minimizes the original objective function (3.3). Thus, we have obtained a solution for the original system of nonlinear equations.

Unconstrained optimization refers to the process of minimization or maximization of an objective function without any constraints. The objective function can have either one variable or several variables. In the first half of this chapter, we focus on one-dimensional problems and the methods used to solve them. The multidimensional case is discussed later in the chapter.

3.1 Basic definitions

We start with basic definitions and results (see [7, 9, 11]).

Definition 18. Let Q be an $n \times n$ symmetric matrix. The vectors $\boldsymbol{d}_1, ..., \boldsymbol{d}_n \in \mathbb{R}^n$ are called *conjugate directions* if they are linearly independent and if $\boldsymbol{d}_i^T Q \boldsymbol{d}_j = 0$ for all $i \neq j$.

Definition 19. A vector $\boldsymbol{x}^* \in \mathbb{R}^n$ is a *local minimum* of a function $f : \mathbb{R}^n \to \mathbb{R}$ if there exist $\delta > 0$ such that for all $\boldsymbol{x} \in \mathbb{R}^n$ satisfying $\|\boldsymbol{x} - \boldsymbol{x}^*\| \leq \delta$ we have

$$f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$$

and $\boldsymbol{x}^* \in \mathbb{R}^n$ is a *global minimum* if

$$f(\boldsymbol{x}^*) \leq f(\boldsymbol{x}) \quad \text{for all } \boldsymbol{x} \in \mathbb{R}^n.$$

Definition 20. A vector $\boldsymbol{x}^* \in \mathbb{R}^n$ is a *local maximum* of a function $f : \mathbb{R}^n \to \mathbb{R}$ if there exist $\delta > 0$ such that for all $\boldsymbol{x} \in \mathbb{R}^n$ satisfying $\|\boldsymbol{x} - \boldsymbol{x}^*\| \leq \delta$ we have

$$f(\boldsymbol{x}) \leq f(\boldsymbol{x}^*)$$

and $\boldsymbol{x}^* \in \mathbb{R}^n$ is a *global maximum* if

$$f(\boldsymbol{x}) \leq f(\boldsymbol{x}^*) \quad \text{for all } \boldsymbol{x} \in \mathbb{R}^n.$$

Next, we present some optimality conditions. These conditions are crucial for understanding and solving optimization problems because they provide the necessary and sometimes sufficient conditions under which a solution can be considered optimal. The first condition presented is known as the *first order necessary optimality condition* (see [9, 11]).

Theorem 5. *If $\boldsymbol{x}^* \in \mathbb{R}^n$ is a local minimum of a function $f : \mathbb{R}^n \to \mathbb{R}$ and f is continuously differentiable in an open neighbourhood of $\boldsymbol{x}^*$, then*

$$\nabla f(\boldsymbol{x}^*) = \mathbf{0}.$$

The previous result means that the gradient of f at $\boldsymbol{x}^*$ needs to be zero. This indicates a *stationary point*, meaning that there is no first order change in f at $\boldsymbol{x}^*$. In other words, $\boldsymbol{x}^*$ is a point where the gradient vanishes.

This next condition is known as the *second order necessary optimality condition* (see [9, 11]).

Theorem 6. *Assume that a function $f : \mathbb{R}^n \to \mathbb{R}$ is twice continuously differentiable in an open neighbourhood of $\boldsymbol{x}^*$. If $\boldsymbol{x}^* \in \mathbb{R}^n$ is a local minimizer of a function f, then*

$$\nabla f(\boldsymbol{x}^*) = \mathbf{0},$$

and

$$H(\boldsymbol{x}^*) \succeq 0,$$

where $\succeq 0$ means that the matrix $H(\boldsymbol{x}^*)$ is positive semidefinite.

Theorem 6 means that the Hessian matrix of f at $\boldsymbol{x}^*$, denoted here by $H(\boldsymbol{x}^*)$, should be positive semidefinite (see Definition 17, Chapter 1). To verify positive semidifiniteness, one can check if all the leading principal minors of the matrix are non-negative [38], which is equivalent to verifying that the eigenvalues of the matrix are non-negative. If the Hessian is positive semidefinite, the function does not curve downwards in any direction around $\boldsymbol{x}^*$. If the Hessian is negative in some (any) direction, $\boldsymbol{x}^*$ could be a saddle point (or local maximum) rather than a minimum.

Theorem 7. *[9] Suppose that a function $f : \mathbb{R}^n \to \mathbb{R}$ is twice continuously differentiable in an open neighborhood of $\boldsymbol{x}^* \in \mathbb{R}^n$. If*

$$\nabla f(\boldsymbol{x}^*) = \mathbf{0},$$

and

$$H(\boldsymbol{x}^*) \succ 0,$$

then $\boldsymbol{x}^*$ is a local minimum of f. Here $\succ 0$ means that the matrix $H(\boldsymbol{x}^*)$ is positive definite.

Theorem 7 is known as the *second order sufficient optimality condition*. While the second order necessary condition requires the Hessian to be positive semidefinite, the second order sufficient condition requires the Hessian to be positive definite. This implies that the function f curves upward in all directions around $\boldsymbol{x}^*$. This guarantees that $\boldsymbol{x}^*$ is indeed a local minimum.

3.2 Direct search methods in one-dimensional domain

Here we start by discussing direct search methods to solve unconstrained optimization problems in the one-dimensional domain. We are presenting these methods here for the sake of completeness in the methodology section, but not all of them will be used in the hybrid methods, which will be discussed in Chapter 4.

The term direct search was first suggested by Hooke and Jeeves in 1961 [37]. Direct search methods do not need any information about function derivatives, and they rely solely on sequential evaluations of f at the iteration points x_k. The function values $f(x_k)$ are then used to guide the search towards an optimum.

Direct search methods are mostly applied when either the objective function is not differentiable or the cost of the derivative computation is too high in practice. That is why these direct search methods are sometimes referred to as *derivative-free methods*.

We first define some properties of functions needed in this section (see [9, 10]).

Definition 21. Let S be a non-empty convex set in $\mathbb{R}$. A function $f : S \to \mathbb{R}$ is said to be *strictly quasiconvex function* if for each $x_1, x_2 \in S$, $x_1 \neq x_2$ with $f(x_1) \neq f(x_2)$, we have

$$f(\lambda x_1 + (1-\lambda)x_2) < \max\{f(x_1), f(x_2)\},$$

for all $\lambda \in (0, 1)$.

Definition 22. A function $f : S \to \mathbb{R}$ is said to be *unimodal* if it has only one local minimum or maximum in the given domain $S \subseteq \mathbb{R}$. Unimodal functions may be nondifferentiable or even discontinuous.

Definition 23. *Interval of uncertainty* can be defined as the closed interval $[a, b] \in \mathbb{R}$ where $a < b$ within which we expect a local optimum of the function f to lie.

3.2.1 Dichotomous search

Let us consider a strictly quasiconvex, unimodal, and continuous function $f : \mathbb{R} \to \mathbb{R}$. It needs to be minimized over the initial interval of uncertainty $[a, b]$. In the dichotomous search method, at each iteration, portions of the interval of uncertainty that do not contain the minimum are eliminated. In addition, we need at least two function evaluations to reduce the interval of uncertainty, unlike in the bisection method, where we determine a root of a function (see subsection 2.2.1). At the iteration k, two points λ_k and μ_k are placed close to the middle of the interval of uncertainty $[a_k, b_k]$ such that

$$\lambda_k = \frac{a_k + b_k}{2} - \epsilon \quad \text{and} \quad \mu_k = \frac{a_k + b_k}{2} + \epsilon$$

where $\epsilon > 0$ is a small tolerance. After this we evaluate function values at λ_k and μ_k and check whether $f(\lambda_k) > f(\mu_k)$ in which case the new interval is $[a_{k+1}, b_{k+1}] = [\lambda_k, b_k]$. If $f(\lambda_k) < f(\mu_k)$ then the new interval is $[a_{k+1}, b_{k+1}] = [a_k, \mu_k]$. The global optimum belongs to the new interval due to the property of strict quasiconvexity for unimodal functions (see Theorem 8.1.1 [9]). If $f(\lambda_k) = f(\mu_k)$, then the new interval of uncertainty is reduced directly to $[\lambda_k, \mu_k]$, However, in practice, such exact equality is unlikely to occur due to limitations in numerical precision and the nature of the function evaluation.

As we have seen, the dichotomous method requires two function evaluations at each iteration. The length I_k of the final interval of uncertainty $[a_k, b_k]$ at iteration k can be calculated approximately by the equation

$$I_k = \frac{I_0}{2^k},$$

where $I_0 = b - a$ is the length of the initial interval of uncertainty. The method is named for the fact that, at each iteration, the length of the interval containing the minimum is approximately halved. The exact length of the interval of uncertainty at iteration k is given by

$$I_k = \frac{1}{2^k} I_0 + 2\epsilon\Big(1 - \frac{1}{2^k}\Big)$$

where $\epsilon > 0$ is the small distinguishable tolerance used in the method. The distinguishable tolerance represents the smallest interval length within which two function values can be numerically distinguished from each other. In dichotomous search, as already seen, the reduction ratio is approximately $(0.5)^k$, reflecting the exponential reduction in the interval length and demonstrating the halving characteristic of the method [9].

Computational algorithm

Algorithm 6: Dichotomous search [9]

Input: Initial interval of uncertainty $[a, b]$, tolerance $\epsilon > 0$, maximum number of iterations N

Output: Approximate solution x^*

1 Initialize $k = 0$ and set $a_k = a$, $b_k = b$

2 While $(k < N)$

If $|a_k - b_k| < \epsilon$ then

Set $x^* = \frac{a_k + b_k}{2}$ and go to step 4

Else

calculate $\lambda_k = \frac{a_k + b_k}{2} - \epsilon$ and $\mu_k = \frac{a_k + b_k}{2} + \epsilon$

If $f(\lambda_k) > f(\mu_k)$ then

Update $a_{k+1} = \lambda_k,\ b_{k+1} = b_k$

Else

Update $a_{k+1} = a_k,\ b_{k+1} = \mu_k$

Update iteration counter $k = k + 1$

3 End while

OUTPUT: ('Method failed to converge after N iterations')

4 Stop

Example with Matlab

A strictly quasiconvex, unimodal, and continuous function $f(x) = x^2 - 7x + 12$ is minimized using the dichotomous search with starting interval $[2, 4]$, tolerance 10^{-6}, and the maximum number of iterations $N = 50$. The function attained the minimum value -0.25 at point 3.5. The results are presented in Table 3.1.

Iter	a_k	b_k	f(a_k)	f(b_k)	|a_k - b_k|
0	2.000000	4.000000	2.000000	0.000000	2.00000000
1	3.000000	4.000000	0.000000	0.000000	1.00000000
2	3.000000	3.500000	0.000000	-0.250000	0.50000000
3	3.250000	3.500000	-0.187500	-0.250000	0.25000000
4	3.375000	3.500000	-0.234375	-0.250000	0.12500000
5	3.437500	3.500000	-0.246094	-0.250000	0.06250000
6	3.468750	3.500000	-0.249023	-0.250000	0.03125000
7	3.484375	3.500000	-0.249756	-0.250000	0.01562500
8	3.492188	3.500000	-0.249939	-0.250000	0.00781250
9	3.496094	3.500000	-0.249985	-0.250000	0.00390625
10	3.498047	3.500000	-0.249996	-0.250000	0.00195312
11	3.499023	3.500000	-0.249999	-0.250000	0.00097656
12	3.499512	3.500000	-0.250000	-0.250000	0.00048828
13	3.499756	3.500000	-0.250000	-0.250000	0.00024414
14	3.499878	3.500000	-0.250000	-0.250000	0.00012207
15	3.499939	3.500000	-0.250000	-0.250000	0.00006104
16	3.499969	3.500000	-0.250000	-0.250000	0.00003052
17	3.499985	3.500000	-0.250000	-0.250000	0.00001526
18	3.499992	3.500000	-0.250000	-0.250000	0.00000763
19	3.499996	3.500000	-0.250000	-0.250000	0.00000381
20	3.499998	3.500000	-0.250000	-0.250000	0.00000191
21	3.499999	3.500000	-0.250000	-0.250000	0.00000095

Table 3.1: Minimization of the function $f(x) = x^2 - 7x + 12$ with the initial interval $[2, 4]$ using the dichotomous search.

3.2.2 Golden section method

The golden section search method is one of the most efficient methods for finding the global minimum of unimodal continuous functions. This method was developed by the famous statistician Kiefer [40] in 1953. He also invented the Fibonacci search method, which we discuss in the next subsection.

The golden section search method reduces the interval of uncertainty enclosing the optimum by a constant factor at each step, requiring two function evaluations in the first iteration and only one additional function evaluation in subsequent iterations. Let us consider an unimodal continuous function $f : \mathbb{R} \to \mathbb{R}$ where we need to find the global minimum in $[a, b]$. At iteration k the calculation of the two points λ_k and μ_k within the interval of uncertainity $[a_k, b_k]$ can be defined as follows:

$$\lambda_k = a_k + (1 - \alpha)(b_k - a_k) \quad \text{and} \quad \mu_k = a_k + \alpha(b_k - a_k), \tag{3.5}$$

where $\alpha \in (0, 1)$. The new reduced interval of uncertainty $[a_{k+1}, b_{k+1}]$ depends on $f(\lambda_k)$ and $f(\mu_k)$. If $f(\lambda_k) < f(\mu_k)$ the new interval is $[a_k, \mu_k]$ and otherwise it is $[\lambda_k, b_k]$. We can easily prove that at each iteration, only one new point needs to be calculated, as the second point coincides with one of the previously computed points. Specifically, λ_{k+1} aligns with μ_k, or μ_{k+1} aligns with λ_k. This characteristic

optimizes the computational process by reducing the number of new function evaluations required per iteration while efficiently updating the interval of uncertainty.

The main point to observe in the golden section search method is that the ratio of two adjacent intervals of uncertainty is a constant α. In simple words, this means

$$\frac{I_k}{I_{k+1}} = \frac{I_{k+1}}{I_{k+2}} = \cdots = \alpha,$$

where $I_k = b_k - a_k$ for all $k \geq 0$. To calculate this constant, we use equation (3.5) and

$$\mu_{k+1} = a_{k+1} + \alpha(b_{k+1} - a_{k+1})$$

and get (see the proof of Theorem 8) the equation

$$\alpha^2 + \alpha - 1 = 0$$

which has solutions $\alpha = 0.618$ and $\alpha = -1.618$. Since $\alpha \in (0, 1)$, the positive α is selected and it is called *golden ratio*.

Computational algorithm

Algorithm 7: Golden section search [9]

Input: Initial interval of uncertainty $[a, b]$, tolerance $\epsilon > 0$, $\alpha = 0.618$, maximum number of iterations N

Output: Approximate solution x^*

1 Initialize $k = 0,\ a_k = a,\ b_k = b$

2 While $(k < N)$

  Calculate $\lambda_k = a_k + (1 - \alpha)(b_k - a_k)$ and $\mu_k = a_k + \alpha(b_k - a_k)$

  If $|a_k - b_k| > \epsilon$ then

    If $f(\lambda_k) > f(\mu_k)$ then

      Update $[a_{k+1}, b_{k+1}] = [\lambda_k, b_k]$

      Set $\lambda_{k+1} = \mu_k$ and $\mu_{k+1} = a_{k+1} + \alpha(b_{k+1} - a_{k+1})$

      Calculate $f(\mu_{k+1})$

    Else

      Update $[a_{k+1}, b_{k+1}] = [a_k, \mu_k]$

      Set $\mu_{k+1} = \lambda_k$ and $\lambda_{k+1} = a_{k+1} + (1 - \alpha)(b_{k+1} - a_{k+1})$

      Calculate $f(\lambda_{k+1})$

    Update iteration counter $k = k + 1$

  Else

    Set $x^* = \frac{a_k + b_k}{2}$ and go to step 4

3 End while

  OUTPUT ('Method failed to converge after N iterations')

4 Stop

Example with Matlab

A continuous unimodal function $f(x) = x^2 - 7x + 12$ is solved using golden section method with starting interval $[2, 4]$, $\alpha = 0.618$, tolerance 10^{-6} and the maximum

number of iterations $N = 50$. The function attained the minimum value -0.25 at point 3.5, which is the same as obtained with the dichotomous search 3.1. The results are presented in Table 3.2.

Iter	a_k	b_k	f(a_k)	f(b_k)	|a_k - b_k|
0	2.000000	4.000000	2.000000	0.000000	2.0000000
1	2.764000	4.000000	0.291696	0.000000	1.2360000
2	3.236152	4.000000	-0.180384	0.000000	0.7638480
3	3.236152	3.708210	-0.180384	-0.206649	0.4720581
4	3.416478	3.708210	-0.243024	-0.206649	0.2917319
5	3.416478	3.596768	-0.243024	-0.240636	0.1802903
6	3.416478	3.527898	-0.243024	-0.249222	0.1114194
7	3.459040	3.527898	-0.248322	-0.249222	0.0688572
8	3.485344	3.527898	-0.249785	-0.249222	0.0425537
9	3.485344	3.511642	-0.249785	-0.249864	0.0262982
10	3.495390	3.511642	-0.249979	-0.249864	0.0162523
11	3.495390	3.505434	-0.249979	-0.249970	0.0100439
12	3.495390	3.501597	-0.249979	-0.249997	0.0062071
13	3.497761	3.501597	-0.249995	-0.249997	0.0038360
14	3.499226	3.501597	-0.249999	-0.249997	0.0023707
15	3.499226	3.500691	-0.249999	-0.250000	0.0014651
16	3.499786	3.500691	-0.250000	-0.250000	0.0009054
17	3.499786	3.500345	-0.250000	-0.250000	0.0005595
18	3.499786	3.500132	-0.250000	-0.250000	0.0003458
19	3.499918	3.500132	-0.250000	-0.250000	0.0002137
20	3.499918	3.500050	-0.250000	-0.250000	0.0001321
21	3.499968	3.500050	-0.250000	-0.250000	0.0000816
22	3.499968	3.500019	-0.250000	-0.250000	0.0000504
23	3.499988	3.500019	-0.250000	-0.250000	0.0000312
24	3.499988	3.500007	-0.250000	-0.250000	0.0000193
25	3.499995	3.500007	-0.250000	-0.250000	0.0000119
26	3.499995	3.500002	-0.250000	-0.250000	0.0000074
27	3.499998	3.500002	-0.250000	-0.250000	0.0000045
28	3.499998	3.500001	-0.250000	-0.250000	0.0000028
29	3.499999	3.500001	-0.250000	-0.250000	0.0000017
30	3.500000	3.500001	-0.250000	-0.250000	0.0000011
31	3.500000	3.500000	-0.250000	-0.250000	0.0000007

Table 3.2: Minimization of function $f(x) = x^2 - 7x + 12$ over interval $[2, 4]$ using golden section method.

Convergence

Next, we state the convergence result for the golden section search.

Theorem 8. *[48] Let S be a closed interval [a,b] and $f : S \to \mathbb{R}$ be a continuous and unimodal function. Then the golden section method converges to the unique minimum x^*, and the convergence rate is linear.*

Proof. Since f is continuous and unimodal on the closed interval $[a, b]$, there exists a unique minimum of that function by definition. As the number of iterations of the golden section method approaches infinity, the sequence of intervals $[a_k, b_k]$ generated by the method converges to a single point x^*, so we have

$$\lim_{k\to\infty} |b_k - a_k| = 0.$$

Since the interval size is consistently reduced and the minimum is always retained within the interval as k tends to infinity, the continuous function values at the endpoints of these intervals converge to the function value at x^*, and the point x^* is the minimum. This means that

$$\lim_{k\to\infty} f(a_k) = \lim_{k\to\infty} f(b_k) = f(x^*).$$

Thus, the golden section method is guaranteed to converge to the optimal solution within the closed interval $[a, b]$ when the considered function is unimodal and continuous on the interval. The golden section search method does not require strict quasiconvexity.

For the rate of convergence, let $I_0 = b - a$ represent the length of the initial interval of uncertainty, and let λ_0 and μ_0 be two interior points within this interval. The current interval is updated based on the function evaluations at λ_0 and μ_0, progressively reducing the interval length in a ratio α such that $I_1 = \alpha I_0$.

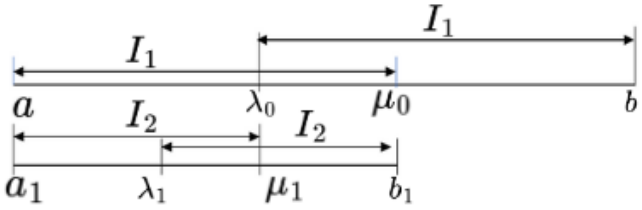

Figure 3.1: First two intervals of uncertainty for the golden section method.

Thus from figure 3.1, we can write

$$I_1 = \mu_0 - a = b - \lambda_0, \tag{3.6}$$

which can also be written as

$$\mu_0 - a = \alpha(b - a), \tag{3.7}$$

$$b - \lambda_0 = \alpha(b - a). \tag{3.8}$$

In other words, α specifies the accuracy ratio and the convergence rate. Now let us assume that $f(\lambda_0) < f(\mu_0)$, then the new interval of uncertainty changes to $[a_1, b_1] = [a, \mu_0]$. New point $\lambda_1 \in [a, \mu_0]$ is located at distance I_2 from μ_0. Similarly $\mu_1 = \lambda_0$ is located at distance I_2 from a. Thus we have

$$I_2 = \lambda_0 - a = \mu_0 - \lambda_1.$$

Since the decrease ratio α remains constant for each new interval, we obtain

$$\lambda_0 - a = \alpha(\mu_0 - a), \tag{3.9}$$

$$\mu_0 - \lambda_1 = \alpha(\mu_1 - a).$$

Thus, if we put values λ_0 and μ_0 from (3.7) and (3.8) into (3.9), we get

$$b - \alpha(b-a) - a = \alpha[a + \alpha(b-a) - a], \tag{3.10}$$

which yields

$$(b-a)(1-\alpha-\alpha^2) = 0.$$

Since $(b-a)$ is the length of the initial interval and it cannot be zero, we need to have

$$1 - \alpha - \alpha^2 = 0.$$

Solutions for this equation are $\alpha = 0.618$ and $\alpha = -1.618$. We can not accept a negative value for α as we want to reduce the interval of uncertainty. We accept only value

$$\alpha = \frac{-1+\sqrt{5}}{2} \approx 0.618.$$

In general form we have

$$I_{k+1} = \alpha I_k.$$

To reconcile with the definition of linear convergence for $\{x_k\}$, we can simply bound $|x_k - x^*|$ with I_k. So the same linear factor α serves as a linear convergence rate for the sequence $\{x_k\}$.

□

3.2.3 Fibonacci method

The Fibonacci search method is also a direct search method. It is more efficient than the dichotomous method 3.2.1 and the golden section method 3.2.2. This method is also used to find the global minimum of a continuous and unimodal function over the provided initial interval of uncertainty $[a, b]$. Similarly to the golden section method, the Fibonacci method does not require strict quasiconvexity. The Fibonacci search method is a refinement of the golden section search algorithm, originally developed by Kiefer, designed to locate the maximum or minimum of an unimodal function within a specified interval.[40]

We consider the problem where $f : \mathbb{R} \to \mathbb{R}$ is continuous and unimodal in the interval $[a, b]$. The Fibonacci method also reduces the interval of uncertainty at each iteration to locate the global minimum. As well as the golden section search method, the Fibonacci method also needs two functional evaluations at the first iteration and only one functional evaluation for the successive iterations. But the main difference arises when we calculate new points λ_k and μ_k at the iteration k. In the golden section method, the value of α is constant at every iteration, but in the case of the Fibonacci search method, α is the ratio of two consecutive Fibonacci numbers, so it

varies at every iteration. Let n be the number of function evaluations done in the method. For example at iteration $k = 1, 2, ..., n-1$ value of α_k is defined as follows:

$$\alpha_k = \frac{\mathcal{F}_{n-(k+1)}}{\mathcal{F}_{n-k}}$$

where $\mathcal{F}_k$ for all $k = 1, 2, 3, \ldots$ is from the Fibonacci series. In general $\mathcal{F}_k$ is defined as follows

$$\mathcal{F}_0 = \mathcal{F}_1 = 1 \quad \text{and} \quad \mathcal{F}_{k+1} = \mathcal{F}_k + \mathcal{F}_{k-1} \quad \forall \quad k = 1, 2, 3, \ldots$$

so we get the sequence 1,1,2,3,5,8,13,21,....

At the beginning of the iteration k, the original interval of uncertainty $[a, b]$ is reduced to $[a_k, b_k]$. So, like in the golden section method, we need to calculate two points λ_k and μ_k such that $a_k \leq \lambda_k \leq \mu_k \leq b_k$. These points are obtained with the formulas

$$\lambda_k = a_k + \frac{\mathcal{F}_{n-(k+1)}}{\mathcal{F}_{n-(k-1)}}(b_k - a_k) \tag{3.11}$$

and

$$\mu_k = a_k + \frac{\mathcal{F}_{n-k}}{\mathcal{F}_{n-(k-1)}}(b_k - a_k) \quad \forall \quad k = 1, 2, \ldots, n-1 \tag{3.12}$$

where n is the maximum number of iterations. Note that always $\lambda_{n-1} = \mu_{n-1}$ and thus we need to decrease or increase during the iteration $n-1$ either λ_{n-1} or μ_{n-1} with ϵ to decrease the current interval $[a_{n-1}, b_{n-1}]$. The new interval $[a_{k+1}, b_{k+1}]$ depends on $f(\lambda_k)$ and $f(\mu_k)$. If $f(\lambda_k) < f(\mu_k)$, the new interval of uncertainty is $[a_k, \mu_k]$. Otherwise it is $[\lambda_k, b_k]$. Suppose that $[a_{k+1}, b_{k+1}] = [\lambda_k, b_k]$. In this case, when we move on to execute the iteration round $k+1$, only one function evaluation is required as λ_{k+1} coincides with μ_k. The point μ_{k+1} can be calculated using equation (3.12) when we replace k with $k+1$ and this gives

$$\mu_{k+1} = a_{k+1} + \frac{\mathcal{F}_{n-(k+1)}}{\mathcal{F}_{n-k}}(b_{k+1} - a_{k+1}) \quad \forall \quad k = 1, 2, 3, \ldots, n-2.$$

Similarly, if the new interval of uncertainty is $[a_{k+1}, b_{k+1}] = [a_k, \mu_k]$, then $\mu_{k+1} = \lambda_k$ and λ_{k+1} can be obtained by updating k with $k+1$ in (3.11).

One key point for the Fibonacci search method is the total number of iterations n required for the desired length of the interval of uncertainty. From equation (3.12) we can see that

$$\mu_k - a_k = \frac{\mathcal{F}_{n-k}}{\mathcal{F}_{n-(k-1)}}(b_k - a_k).$$

If $f(\lambda_k) < f(\mu_k)$ the new interval of uncertainty is $[a_{k+1}, b_{k+1}] = [a_k, \mu_k]$, so we get

$$b_{k+1} - a_{k+1} = \mu_k - a_k = \frac{\mathcal{F}_{n-k}}{\mathcal{F}_{n-(k-1)}}(b_k - a_k).$$

On other hand if $f(\lambda_k) > f(\mu_k)$ the new interval of uncertainty is $[a_{k+1}, b_{k+1}] = [\lambda_k, b_k]$, so we get

$$b_{k+1} - a_{k+1} = b_k - \lambda_k = \frac{\mathcal{F}_{n-k}}{\mathcal{F}_{n-(k-1)}}(b_k - a_k).$$

Both cases reduce the interval length proportionally. It shows that the length of the interval at the iteration $k+1$ is proportional to the length of the interval at the iteration k. Therefore, during the iteration n, we obtain

$$b_n - a_n = \left(\prod_{k=1}^{n-1} \frac{\mathcal{F}_{n-k)}}{\mathcal{F}_{n-(k-1)}} \right) (b_1 - a_1). \tag{3.13}$$

Since

$$\prod_{k=1}^{n-1} \frac{\mathcal{F}_{n-k}}{\mathcal{F}_{n-(k-1)}} = \frac{\mathcal{F}_{n-1} \cdot \mathcal{F}_{n-2} \cdot \cdots \cdot \mathcal{F}_1}{\mathcal{F}_n \cdot \mathcal{F}_{n-1} \cdot \cdots \cdot \mathcal{F}_0} = \frac{1}{\mathcal{F}_n}$$

so we can write (3.13) in more simpler way as

$$b_n - a_n = \frac{1}{\mathcal{F}_n}(b_1 - a_1).$$

Thus, the number of iterations must be chosen in such a way that $\frac{b_1 - a_1}{\mathcal{F}_n} \approx \epsilon$.

Computational algorithm

Algorithm 8: Fibonacci search method [9]

Input: Initial interval of uncertainty $[a, b]$, tolerance $\epsilon > 0$, maximum number of iterations N, $\mathcal{F}_0 = \mathcal{F}_1 = 1$

Output: Approximate solution x^*

1 Initialize $k = 1,\ a_k = a,\ b_k = b$

2 Compute the smallest integer $n \geq 2$ such that

$$\mathcal{F}_n := \mathcal{F}_{n-1} + \mathcal{F}_{n-2} \geq \frac{b-a}{\epsilon}$$

3 Calculate $\lambda_k = a_k + \frac{\mathcal{F}_{n-2}}{\mathcal{F}_n}(b_k - a_k)$ and $\mu_k = a + \frac{\mathcal{F}_{n-1}}{\mathcal{F}_n}(b_k - a_k)$

4 While $(k \leq N)$

If $|a_k - b_k| > \epsilon$ then

If $f(\lambda_k) > f(\mu_k)$

Update $[a_{k+1}, b_{k+1}] = [\lambda_k, b_k]$, $\lambda_{k+1} = \mu_k$,

$\mu_{k+1} = a_{k+1} + \frac{\mathcal{F}_{n-(k+1)}}{\mathcal{F}_{n-k}}(b_{k+1} - a_{k+1})$

If $\lambda_{k+1} = \mu_{k+1}$, set $\mu_{k+1} = \mu_{k+1} + \epsilon$.

Calculate $f(\mu_{k+1})$

Else

Update $[a_{k+1}, b_{k+1}] = [a_k, \mu_k]$ and $\mu_{k+1} = \lambda_k$,

$\lambda_{k+1} = a_{k+1} + \frac{\mathcal{F}_{n-(k+2)}}{\mathcal{F}_{n-k}}(b_{k+1} - a_{k+1})$

If $\mu_{k+1} = \lambda_{k+1}$, set $\lambda_{k+1} = \lambda_{k+1} - \epsilon$.

Calculate $f(\lambda_{k+1})$

Update iteration counter $k = k + 1$

Else

Set $x^* = \frac{a_k + b_k}{2}$ and go to step 6

5 End while

OUTPUT: ('Method failed to converge after N iterations').

6 Stop

Example with Matlab

A continuous unimodal function $f(x) = x^2 - 7x + 12$ is solved using the Fibonacci method with starting interval $[2, 4]$, tolerance 10^{-6}, and the maximum number of iterations $N = 50$. The function attained the minimum value -0.25 at point 3.5, which is the same as obtained with the dichotomous search 3.1 and golden section method 3.2. The results are presented in Table 3.3.

Iter	a_k	b_k	f(a_k)	f(b_k)	|a_k - b_k|
1	2.000000	4.000000	2.000000	0.000000	2.0000000
2	2.763932	4.000000	0.291796	0.000000	1.2360680
3	3.236068	4.000000	-0.180340	0.000000	0.7639320
4	3.236068	3.708204	-0.180340	-0.206651	0.4721360
5	3.416408	3.708204	-0.243012	-0.206651	0.2917961
6	3.416408	3.596748	-0.243012	-0.240640	0.1803399
7	3.416408	3.527864	-0.243012	-0.249224	0.1114562
8	3.458980	3.527864	-0.248317	-0.249224	0.0688837
9	3.485292	3.527864	-0.249784	-0.249224	0.0425725
10	3.485292	3.511603	-0.249784	-0.249865	0.0263112
11	3.495342	3.511603	-0.249978	-0.249865	0.0162612
12	3.495342	3.505392	-0.249978	-0.249971	0.0100500
13	3.495342	3.501553	-0.249978	-0.249998	0.0062112
14	3.497714	3.501553	-0.249995	-0.249998	0.0038388
15	3.499180	3.501553	-0.249999	-0.249998	0.0023725
16	3.499180	3.500647	-0.249999	-0.250000	0.0014663
17	3.499740	3.500647	-0.250000	-0.250000	0.0009062
18	3.499740	3.500300	-0.250000	-0.250000	0.0005601
19	3.499740	3.500087	-0.250000	-0.250000	0.0003461
20	3.499873	3.500087	-0.250000	-0.250000	0.0002139
21	3.499954	3.500087	-0.250000	-0.250000	0.0001322
22	3.499954	3.500036	-0.250000	-0.250000	0.0000817
23	3.499986	3.500036	-0.250000	-0.250000	0.0000505
24	3.499986	3.500017	-0.250000	-0.250000	0.0000312
25	3.499986	3.500005	-0.250000	-0.250000	0.0000193
26	3.499993	3.500005	-0.250000	-0.250000	0.0000119
27	3.499997	3.500005	-0.250000	-0.250000	0.0000074
28	3.499997	3.500002	-0.250000	-0.250000	0.0000046
29	3.499999	3.500002	-0.250000	-0.250000	0.0000028
30	3.499999	3.500001	-0.250000	-0.250000	0.0000017
31	3.499999	3.500000	-0.250000	-0.250000	0.0000011
32	3.500000	3.500000	-0.250000	-0.250000	0.0000007

Table 3.3: Minimization of function $f(x) = x^2 - 7x + 12$ over interval $[2, 4]$ using Fibonacci method.

Convergence

The next theorem gives the convergence result for the Fibonacci search.

Theorem 9. *[48] Let S be a closed interval $[a, b]$ and $f : S \to \mathbb{R}$ be a continuous and unimodal function. Then the Fibonacci search method converges to the unique minimum x^*, and the convergence rate is linear.*

Proof. Since f is continuous and unimodal on the closed interval $[a, b]$, it follows from the definition that f has a unique minimum in this interval. The Fibonacci search method systematically reduces the interval of uncertainty $[a_k, b_k]$ at each iteration, ensuring that the minimum x^* is retained within the reduced interval.

As the number of iterations $k \to \infty$ the length of the intervals $[a_k, b_k]$ converges to zero:

$$\lim_{k\to\infty} |b_k - a_k| = 0.$$

Since function f is continuous, the function values at the endpoints of the intervals $[a_k, b_k]$ approach the function value at the unique minimumizer x^*. During each iteration, the Fibonacci search method computes two new points λ_k and μ_k, which are used to reduce the interval based on the function values at these points. This process ensures that the sequence of intervals shrinks around x^*.

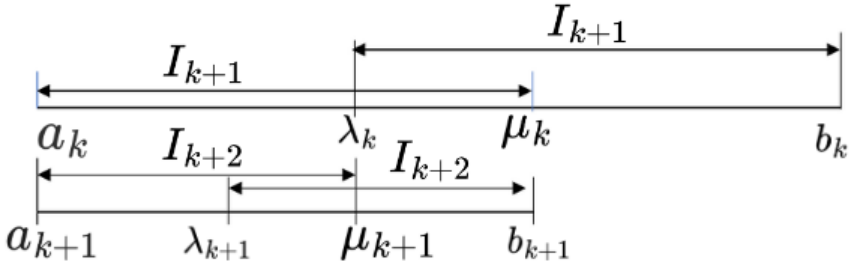

Figure 3.2: Fibonacci method new interval of uncertainty.

Assume, that a new search interval is $[a_{k+1} = a_k, b_{k+1} = \mu_k]$. At each iteration, the interval of uncertainty is reduced by the factor of r_k. So, I_{k+1} can be calculated as

$$I_{k+1} = r_k(b_k - a_k) = r_k I_k \quad \text{for} \quad k = 1, 2, 3, \ldots \tag{3.14}$$

where original interval $I_1 = (b - a)$. Similarly, we can get

$$I_{k+2} = r_{k+1}(b_{k+1} - a_{k+1}) = r_{k+1} I_{k+1}. \tag{3.15}$$

Now if we look Figure 3.2 we can get the relation

$$\mu_k - \lambda_k = b_{k+1} - \mu_{k+1}. \tag{3.16}$$

In addition, from Figure 3.2, we see that we can write the left side of equation (3.16) as

$$\begin{aligned} \mu_k - \lambda_k =& (b_k - a_k) - (\lambda_k - a_k) - (b_k - \mu_k) \\ =& (b_k - a_k) - (1 - r_k)(b_k - a_k) - (1 - r_k)(b_k - a_k) \\ =& (b_k - a_k)(2r_k - 1) \end{aligned} \tag{3.17}$$

and the right-hand side of (3.16) can be written as

$$\begin{aligned} b_{k+1} - \mu_{k+1} =& (1 - r_{k+1})(b_{k+1} - a_{k+1}) \\ =& (1 - r_{k+1}) r_k (b_k - a_k). \end{aligned} \tag{3.18}$$

By combining equations (3.17) and (3.18) we get

$$2r_k - 1 = (1 - r_{k+1}) r_k \tag{3.19}$$

which simplifies to

$$r_{k+1} = \frac{1 - r_k}{r_k}. \tag{3.20}$$

This can also be written as Fibonacci numbers

$$r_{k+1} = \frac{1 - \frac{\mathcal{F}_{n-k}}{\mathcal{F}_{n-k+1}}}{\frac{\mathcal{F}_{n-k}}{\mathcal{F}_{n-k+1}}} = \frac{\mathcal{F}_{n-k-1}}{\mathcal{F}_{n-k}} \tag{3.21}$$

So when $k = 0$ the value of r_1 is

$$r_1 = \frac{\mathcal{F}_{n-1}}{\mathcal{F}_n}.$$

We can write the length of the interval of uncertainty at the last iteration n as

$$\begin{aligned} I_n =& r_{n-1} I_{n-1} \\ =& \frac{\mathcal{F}_{n-1}\mathcal{F}_{n-2}...\mathcal{F}_1}{\mathcal{F}_n\mathcal{F}_{n-1}...\mathcal{F}_2} I_1 \\ =& \frac{1}{\mathcal{F}_n} I_1. \end{aligned} \tag{3.22}$$

To reconcile with the definition of linear convergence for $\{x_k\}$, we can simply bound $|x_k - x^*|$ with I_k. So the same linear factor $\frac{1}{\mathcal{F}_n}$ serves as a linear convergence rate for the sequence $\{x_k\}$. □

Both the Fibonacci search and golden section search methods exhibit linear convergence with similar rates, as the Fibonacci sequence ratio asymptotically approaches the golden ratio.

3.2.4 Comparison of search methods in one-dimensional domain

The reduction ratio measures the efficiency of direct search methods by quantifying the rate of reduction in the interval of uncertainty per iteration. Analyzing the reduction ratio reveals the efficiency of methods in narrowing the interval of uncertainty and approaching the optimal solution. The ratio is defined as follows:

$$\frac{\text{length of the interval of uncertainty after } n \text{ function evaluations are taken}}{\text{length of the interval of uncertainty at the beginning}}$$

Assuming that a function $f : \mathbb{R} \to \mathbb{R}$ is continuous, unimodal, and strictly quasiconvex on the interval $[a, b]$, each method discussed in this section will, within a finite number of steps, produce a point x such that the final interval of uncertainty satisfies $|x - x^*| < I_f$, where I_f is the length of the final interval of uncertainty and x^* is the minimum point over the interval. Specifically, given the desired accuracy represented by I_f, the required number of function evaluations n can be determined as the smallest positive integer that satisfies the following relationships [9]:

$$\text{Dichotomous search method:} \quad (1/2)^{(n/2)} \leq \frac{I_f}{b-a}$$

$$\text{Golden section method:} \quad (0.618)^{(n-1)} \leq \frac{I_f}{b-a}$$

$$\text{Fibonacci search method:} \quad \mathcal{F}_n \geq \frac{(b-a)}{I_f}.$$

Using the above expressions, we can calculate the required number of function evaluations based on the ratio $(b-a)/I_f$. For a fixed ratio $(b-a)/I_f$, the fewer evaluations that are needed, the more efficient the algorithm. It is clear that the Fibonacci method is the most efficient, followed by the golden section method and then the dichotomous search method. Additionally, note that for large n, $1/\mathcal{F}_n$ approaches $(0.618)^{(n-1)}$, making the Fibonacci search method and the golden section method nearly identical in performance. Among derivative-free methods for minimizing continuous, unimodal, and strictly quasiconvex functions over a closed bounded interval, the Fibonacci search method is the most efficient, requiring the fewest function evaluations [9].

3.3 Gradient-based search methods in multi-dimensional domain

Next, we discuss methods in the multi-dimensional domain. We consider a class of methods known as gradient-based methods, in which the calculation of gradients or Hessian matrices is the key to solving optimization problems. For example, the well-known Newton-Raphson algorithm is a gradient-based method since it uses the first and second derivatives of the function (i.e., the gradient and the Hessian matrix). However, if the objective function contains discontinuities (i.e., it is typically not differentiable at its minimizers or maximizers), these methods do not perform well.

Gradient-based optimization methods iteratively refine a solution to minimize a differentiable objective function f. Starting from an initial guess $\boldsymbol{x}_0$, each iteration typically computes the gradient $\nabla f(\boldsymbol{x}_k)$ at the current iteration point $\boldsymbol{x}_k$, which indicates the direction of steepest ascent. The negative gradient or another computed direction $\boldsymbol{d}_k$ is used to update the solution, with the step length λ_k determining the extent of movement. The updated iteration point is given by

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \lambda_k \boldsymbol{d}_k.$$

Generally, the search direction $\boldsymbol{d}_k$ needs to satisfy

$$\nabla f(\boldsymbol{x}_k)^T \boldsymbol{d}_k < 0, \quad \text{for } k = 0, 1, ...,$$

since this ensures that $\boldsymbol{d}_k$ is a descent direction at $\boldsymbol{x}_k$ [56]. Exact line search chooses λ_k to minimize the function f exactly in the direction of $\boldsymbol{d}_k$. That is

$$\lambda_k = \arg\min_{\lambda>0} f(\boldsymbol{x}_k + \lambda \boldsymbol{d}_k). \tag{3.23}$$

To solve the problem (3.23), one can use any of the methods discussed previously in this chapter. This iterative process is repeated until convergence criteria, such as a small norm of the gradient or negligible change in function value, are met. Gradient-based methods, such as gradient descent and Newton's method, are widely applied due to their efficiency in solving unconstrained optimization problems, particularly for smooth and convex functions. All the descent methods, either directly or indirectly, use the gradient vector to find the best search directions. Additionally, it is essential to note that the efficacy of a gradient-based method hinges significantly on the starting point.

3.3.1 Steepest descent method

The steepest descent method is an iterative gradient-based method for finding the minimum or maximum of a differentiable function. It was first proposed by Cauchy in 1847 [19], so sometimes it is also referred to as *Cauchy method*. The method seeks to minimize a function $f : \mathbb{R}^n \to \mathbb{R}$ by iteratively updating the current guess $\boldsymbol{x}_k \in \mathbb{R}^n$ in the direction of the negative gradient $-\nabla f(\boldsymbol{x}_k)$, which represents the steepest descent direction of the function at each point. The steps are adjusted by a step size (i.e., learning rate) to ensure progress toward the minimum. In general form, the new iterate can be written as

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \lambda_k \nabla f(\boldsymbol{x}_k). \tag{3.24}$$

The optimum step length λ_k can be calculated by solving

$$\min_{\lambda>0} f(\boldsymbol{x}_k + \lambda \boldsymbol{d}_k)$$

where $\boldsymbol{d}_k = -\nabla f(\boldsymbol{x}_k)$.

The computational load of the method is low because it requires only one gradient at each iteration, so it belongs to the category of first-order methods. When we use the search direction $-\nabla f(\boldsymbol{x}_k)$ (antigradient), the method is called the steepest descent, as it aims to minimize the function by moving in the direction of the steepest decrease. Conversely, if the search direction is selected as $\nabla f(\boldsymbol{x}_k)$, the method approaches the local maximum, and it is known as the steepest ascent.

The steepest descent method is typically effective during the initial stages of optimization, with its performance heavily influenced by the choice of the starting point. Unfortunately, when approaching a stationary point, the method often exhibits poor performance by taking small, nearly orthogonal steps. In [9], the authors have discussed some ways to overcome the difficulties of zigzagging by deflecting the gradient. Rather than moving along $\boldsymbol{d} = -\nabla f(\boldsymbol{x})$, we can move along $\boldsymbol{d} = -\boldsymbol{D}\nabla f(\boldsymbol{x})$ or $\boldsymbol{d} = -\nabla f(\boldsymbol{x}) + \boldsymbol{g}$, where $\boldsymbol{D}$ is an appropriate matrix and $\boldsymbol{g}$ is an appropriate vector (see Subsections 3.3.3 and 3.3.4).

Computational algorithm

Algorithm 9: Steepest descent method [9]

Input: Starting point $\boldsymbol{x}_0$, tolerance $\epsilon > 0$, maximum number of iterations N

Output: Approximate solution $\boldsymbol{x}^*$

1 Set $k = 0$

2 While $(k < N)$

Calculate $\nabla f(\boldsymbol{x}_k)$ and $\boldsymbol{d}_k = -\nabla f(\boldsymbol{x}_k)$.

If $\|\nabla f(\boldsymbol{x}_k)\| < \epsilon$

Set $\boldsymbol{x}^* = \boldsymbol{x}_k$ and go to step 4

Else

Find optimal λ_k by solving

$$\min_{\lambda > 0} f(\boldsymbol{x}_k + \lambda \boldsymbol{d}_k)$$

Set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \lambda_k \boldsymbol{d}_k$

Update iteration counter $k = k + 1$

3 End while

OUTPUT: ('Method failed to converge after N iterations')

4 Stop

Example with Matlab

For the purpose of diversifying a set of test problems, we selected a different test problem from Subsection 2.3.1-2.3.3. A function $f(\boldsymbol{x}) = x_1^2 + \frac{1}{2}x_2^2 + 3$ is minimized using the steepest descent method with starting point $(1, 1)^T$, tolerance 10^{-6} and the maximum number of iterations $N = 50$. The function attained the minimum value 3 at point $(0, 0)$. The results are presented in Table 3.4.

Iter	x_1	x_2	f(x_1, x_2)	‖grad_f‖
0	1.000000	1.000000	4.500000	2.23606798
1	0.000000	0.500000	3.125000	0.50000000
2	0.000000	0.250000	3.031250	0.25000000
3	0.000000	0.125000	3.007812	0.12500000
4	0.000000	0.062500	3.001953	0.06250000
5	0.000000	0.031250	3.000488	0.03125000
6	0.000000	0.015625	3.000122	0.01562500
7	0.000000	0.007812	3.000031	0.00781250
8	0.000000	0.003906	3.000008	0.00390625
9	0.000000	0.001953	3.000002	0.00195312
10	0.000000	0.000977	3.000000	0.00097656
11	0.000000	0.000488	3.000000	0.00048828
12	0.000000	0.000244	3.000000	0.00024414
13	0.000000	0.000122	3.000000	0.00012207
14	0.000000	0.000061	3.000000	0.00006104
15	0.000000	0.000031	3.000000	0.00003052
16	0.000000	0.000015	3.000000	0.00001526
17	0.000000	0.000008	3.000000	0.00000763
18	0.000000	0.000004	3.000000	0.00000381
19	0.000000	0.000002	3.000000	0.00000191
20	0.000000	0.000001	3.000000	0.00000095

Table 3.4: Minimization of the function $f(\boldsymbol{x}) = x_1^2 + \frac{1}{2}x_2^2 + 3$ with the starting point $(1, 1)$ using the steepest descent method.

Convergence

Next, we discuss the convergence of the steepest descent method. Recall that a *cluster point* of a sequence $\{\boldsymbol{x}_k\}$ in the Euclidean space $\mathbb{R}^n$ is a point $\bar{\boldsymbol{x}}$ such that, for every neighbourhood V of $\bar{\boldsymbol{x}}$ there are infinitely many natural numbers k such that $\boldsymbol{x}_k \in V$.

Theorem 10. *[27] Let $f : \mathbb{R}^n \to \mathbb{R}$ be continuously differentiable on the set $S = \{\boldsymbol{x} \in \mathbb{R}^n \mid f(\boldsymbol{x}) \leq f(\boldsymbol{x}_0)\}$, where $\boldsymbol{x}_0$ is the starting point and suppose that S is a closed and bounded subset of $\mathbb{R}^n$. Then every cluster point $\bar{\boldsymbol{x}}$ of the sequence $\{\boldsymbol{x}_k\}$ constructed by Algorithm 9 satisfies $\nabla f(\bar{\boldsymbol{x}}) = \boldsymbol{0}$.*

Proof. The proof of this theorem is done by contradiction. By the Bolzano-Weierstrass theorem (also known as the sequential compactness theorem) [8], at least one cluster point of the sequence $\{\boldsymbol{x}_k\}$ must exist. Let $\bar{\boldsymbol{x}}$ be any such cluster point. Without loss of generality, assume that

$$\lim_{k\to\infty} \boldsymbol{x}_k = \bar{\boldsymbol{x}}$$

but that $\nabla f(\bar{\boldsymbol{x}}) \neq \boldsymbol{0}$. This being the case implies that there exists a value $\hat{\lambda} > 0$ such that $(\bar{\boldsymbol{x}} + \hat{\lambda}\boldsymbol{d}) \in \text{int } S$ and $\delta = f(\bar{\boldsymbol{x}}) - f(\bar{\boldsymbol{x}} + \hat{\lambda}\boldsymbol{d}) > 0$ when $\boldsymbol{d} = -\nabla f(\bar{\boldsymbol{x}})$.

Now $\lim_{k\to\infty} \boldsymbol{d}_k = \boldsymbol{d}$. Then since $(\bar{\boldsymbol{x}} + \hat{\lambda}\boldsymbol{d}) \in \text{int } S$ and $(\boldsymbol{x}_k + \hat{\lambda}\boldsymbol{d}_k) \to (\bar{\boldsymbol{x}} + \hat{\lambda}\boldsymbol{d})$, for k sufficiently large we have

$$f(\boldsymbol{x}_k + \hat{\lambda}\boldsymbol{d}_k) \leq f(\bar{\boldsymbol{x}} + \hat{\lambda}\boldsymbol{d}) + \frac{\delta}{2} = f(\bar{\boldsymbol{x}}) - \delta + \frac{\delta}{2}$$

Since $\boldsymbol{d}_k = -\nabla f(\boldsymbol{x}_k)$ is a descent direction at $\boldsymbol{x}_k$, it follows that $f(\boldsymbol{x}_{k+1}) < f(\boldsymbol{x}_k) < \cdots < f(\boldsymbol{x}_0)$. Thus, we have

$$f(\bar{\boldsymbol{x}}) < f(\boldsymbol{x}_k + \lambda_k\boldsymbol{d}_k) \leq f(\boldsymbol{x}_k + \hat{\lambda}\boldsymbol{d}_k) \leq f(\bar{\boldsymbol{x}}) - \delta + \frac{\delta}{2},$$

which is, of course, a contradiction, since δ is positive. Thus $\nabla f(\bar{\boldsymbol{x}}) = \boldsymbol{0}$. □

Under the assumptions of the theorem, the steepest descent method converges to a stationary point, and this is a necessary optimality condition for a local minimum (see Theorem 5).

3.3.2 Newton's method

Newton's method (also known as the Newton-Raphson method) is one of the most fundamental methods in numerical unconstrained optimization. Over time, many variations and modifications of this method have been developed. Normally, this method is used to find the root of a real-valued function using a gradient (see Subsection 2.3.1), but we can also use it to solve an unconstrained optimization problem under certain conditions. The method requires gradient and Hessian matrix information at each iteration. However, for large-scale problems, this can pose challenges due to computational complexity or the infeasibility of exact calculations. Another drawback of Newton's method is that the starting point $\boldsymbol{x}_0$ must be sufficiently close to the local minimum for convergence, so the method doesn't guarantee global convergence.

Let function $f : \mathbb{R}^n \to \mathbb{R}$ be twice differentiable, and assume that the Hessian H of f is continuous. Let us consider the approximation function

$$g(\boldsymbol{x}) = f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^T(\boldsymbol{x} - \boldsymbol{x}_k) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_k)^T H(\boldsymbol{x}_k)(\boldsymbol{x} - \boldsymbol{x}_k),$$

where $H(\boldsymbol{x}_k)$ is the Hessian matrix of f at the iteration point $\boldsymbol{x}_k$. A necessary condition for minimization of g is $\nabla g(\boldsymbol{x}) = \boldsymbol{0}$ and we get

$$\nabla g(\boldsymbol{x}) = \nabla f(\boldsymbol{x}_k) + H(\boldsymbol{x}_k)(\boldsymbol{x} - \boldsymbol{x}_k) = \boldsymbol{0}$$

implying that

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - H(\boldsymbol{x}_k)^{-1}\nabla f(\boldsymbol{x}_k).$$

Note that if $H(\boldsymbol{x}_k)$ is positive definite then $-H(\boldsymbol{x}_k)^{-1}\nabla f(\boldsymbol{x}_k)$ gives us a descent direction $\boldsymbol{d}_k$. The line search (3.23) is not used in Algorithm 10, meaning that the step length is always one.

Computational algorithm

Algorithm 10: Newton's method [9]

Input: Starting point $\boldsymbol{x}_0$, tolerance $\epsilon > 0$, maximum number of iterations N

Output: Approximate solution $\boldsymbol{x}^*$

1 Set $k = 0$

2 While $(k < N)$

Calculate $\nabla f(\boldsymbol{x}_k)$ and $H(\boldsymbol{x}_k)$

If $\|\nabla f(\boldsymbol{x}_k)\| < \epsilon$ then set $\boldsymbol{x}^* = \boldsymbol{x}_k$ and go to step 4

Else

calculate $\boldsymbol{d}_k = -H(\boldsymbol{x}_k)^{-1}\nabla f(\boldsymbol{x}_k)$ and $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{d}_k$

Update iteration counter $k = k + 1$

3 End while

OUTPUT: ('Method failed to converge after N iterations')

4 Stop

Example with Matlab

The system of equations

$$\boldsymbol{F}(\boldsymbol{x}) = \begin{pmatrix} x_1^3 + 3x_2^2 - 21 \\ x_1^2 + 2x_2 + 2 \end{pmatrix} = \boldsymbol{0},$$

can be presented as an unconstrained minimization problem with the objective function $f(\boldsymbol{x}) = \frac{1}{2}(f_1(\boldsymbol{x})^2 + f_2(\boldsymbol{x})^2)$ where $f_1(\boldsymbol{x}) = x_1^3 + 3x_2^2 - 21$ and $f_2(\boldsymbol{x}) = x_1^2 + 2x_2 + 2$. This optimization problem is solved using Newton's method with the starting point $(1.5, -1.5)^T$, tolerance $\epsilon = 10^{-6}$, and maximum number of iterations $N = 50$. The function attained minimum 0 at point $(1.64, -2.34)$. Results are presented in Table 3.5.

Iter	x_1	x_2	f(x_1, x_2)	‖grad_f‖
0	1.500000	-1.500000	59.914062	122.176650
1	2.583134	-3.579665	602.433433	1021.300451
2	2.243742	-2.689551	73.332661	267.929271
3	2.019748	-2.280196	5.178504	54.308851
4	1.854514	-2.234818	0.535320	7.879385
5	1.699214	-2.328533	0.041368	2.995456
6	1.649951	-2.346907	0.000529	0.275409
7	1.643138	-2.349750	0.000000	0.004860
8	1.643038	-2.349787	0.000000	0.000001
9	1.643038	-2.349787	0.000000	0.000000

Table 3.5: Minimization of function $f(\boldsymbol{x}) = \frac{1}{2}(f_1(\boldsymbol{x})^2 + f_2(\boldsymbol{x})^2)$ where $f_1(\boldsymbol{x}) = x_1^3 + 3x_2^2 - 21$ and $f_2(\boldsymbol{x}) = x_1^2 + 2x_2 + 2$ with starting point $(1.5, -1.5)^T$ using Newton's method.

The problem previously addressed using the Newton-Raphson method in Subsection 2.3.1 is revisited here with a different starting point, as the starting point used in Subsection 2.3.1 does not lead to convergence. Consequently, solving the problem as a system of equations or as an optimization problem may yield different solutions, emphasizing the sensitivity of convergence to the choice of initial starting points.

Convergence

Newton's method is a natural choice for solving systems of nonlinear equations when these problems are reformulated as (3.1) because it generalizes well to the multi-dimensional case. For more details regarding the convergence of Newton's method, please refer to [45].

Theorem 11. *[45] Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is twice differentiable and that the Hessian H is Lipschitz continuous in a neighborhood of a solution $\boldsymbol{x}^* \in \mathbb{R}^n$ at which the sufficient conditions (Theorem 2.4 [45]) are satisfied. Consider the iteration*

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - H(\boldsymbol{x}_k)^{-1}\nabla f(\boldsymbol{x}_k).$$

Then,

- *if the starting point $\boldsymbol{x}_0 \in \mathbb{R}^n$ is sufficiently close to $\boldsymbol{x}^*$, the sequence of iterates converges to $\boldsymbol{x}^*$;*
- *the rate of convergence of $\boldsymbol{x}_k$ is quadratic; and*
- *the sequence of gradient norms $\|\nabla f(\boldsymbol{x}_k)\|$ converges quadratically to zero.*

Proof. Please refer to Theorem 3.5 [45] for a detailed proof. □

3.3.3 Quasi-Newton methods

Davidon presented the first quasi-Newton method in the mid-1950s [21]. He used the quasi-Newton method in place of the coordinate descent method to solve long optimization calculations. Long optimization calculations refer to optimization problems that are difficult and time-consuming to solve due to their size and complexity, requiring many iterations or sophisticated methods to reach a solution. Quasi-Newton methods are directly inspired by Newton's method, which uses second-order information. These methods are widely used in optimization, particularly when dealing with nonlinear problems where the exact calculation of the Hessian matrix (second-order derivatives) is computationally expensive or infeasible. Quasi-Newton methods build an approximation to the Hessian matrix using only gradient (first-order derivative) information, which is significantly less computationally demanding. Quasi-Newton methods are also known as *variable metric methods* because they iteratively update an approximation of the Hessian matrix or its inverse, which defines the metric used to measure distances and gradients in the optimization process.

The general idea of quasi-Newton methods can be summarised in the following steps:

1. Optimization Goal:

- The objective is to find $\boldsymbol{x}^*$ that minimizes a nonlinear function $f : \mathbb{R}^n \to \mathbb{R}$. The optimal point satisfies the condition:

$$\nabla f(\boldsymbol{x}^*) = \mathbf{0}.$$

2. New iteration point with Hessian Approximation:

- Quasi-Newton methods avoid directly computing the Hessian matrix $H(\boldsymbol{x}_k)$ at the current iterate $\boldsymbol{x}_k$. Instead, the method iteratively approximates the inverse Hessian matrix H_k^{-1} and updates the solution with formula

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - B_k^{-1}\nabla f(\boldsymbol{x}_k),$$

  where B_k^{-1} is the current approximation of $H(\boldsymbol{x}_k)^{-1}$.

3. Iterative Update Formula for Hessian Approximation:

- As already mentioned, we do not compute the inverse Hessian directly in the quasi-Newton methods. Instead, we update it iteratively based on the observed changes in gradients and the position of new iteration points. The inverse Hessian matrix may be updated using, for example, the BFGS or DFP formulas.

Broyden-Fletcher-Goldfarb-Shanno method

The basic idea of the BFGS (Broyden-Fletcher-Goldfarb-Shanno) method was given by Broyden in 1965 [13] and later updated by Fletcher, Goldfarb, and Shanno in 1970 [28, 32, 52]. The BFGS method is one of the most efficient versions of the quasi-Newton method for solving unconstrained optimization problems. It is used in many solvers because of its efficiency and convergence speed.

The BFGS method begins with a starting point $\boldsymbol{x}_0$ and iteratively updates the estimate by refining the approximation of the Hessian matrix, aiming to improve the solution at each step. The ordinary BFGS method generates a sequence $\{\boldsymbol{x}_k\}$ by the iteration scheme

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \lambda_k \boldsymbol{d}_k,$$

where $\boldsymbol{d}_k$ is a search direction and λ_k is a step-size used to control how far along the search direction $\boldsymbol{d}_k$ the algorithm moves from the current point $\boldsymbol{x}_k$. The value of λ_k is typically obtained through line search, a procedure that seeks to minimize the function f along the direction $\boldsymbol{d}_k$. Formally, λ_k is chosen to satisfy:

$$\lambda_k = \arg\min_{\lambda>0} f\left(\boldsymbol{x}_k + \lambda \boldsymbol{d}_k\right).$$

The search direction $\boldsymbol{d}_k$ can be obtained by solving $B_k \boldsymbol{d}_k = -\nabla f(\boldsymbol{x}_k)$. In it, B_k is an approximation of the Hessian matrix, updated using the following formula

$$B_{k+1} = B_k + \frac{\boldsymbol{y}_k \boldsymbol{y}_k^T}{\boldsymbol{y}_k^T \boldsymbol{s}_k} - \frac{B_k \boldsymbol{s}_k \boldsymbol{s}_k^T B_k}{\boldsymbol{s}_k^T B_k \boldsymbol{s}_k}, \quad \text{for } k = 0, 1, \ldots,$$

where $\boldsymbol{y}_k = \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k)$ and $\boldsymbol{s}_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k$. In addition, B_0 is an initial positive definite matrix, often chosen as the identity matrix. Alternatively, if available and computationally practical, the exact Hessian matrix can be used.

Computational algorithm

Algorithm 11: Broyden-Fletcher-Goldfarb-Shanno method [9]

Input: Starting point $\boldsymbol{x}_0$, tolerance $\epsilon > 0$, initial positive definite matrix B_0, maximum number of iterations N

Output: Approximate solution $\boldsymbol{x}^*$

1 Calculate $\nabla f(\boldsymbol{x}_0)$ and set $k = 0$

2 While $(k < N)$

If $\|\nabla f(\boldsymbol{x}_k)\| < \epsilon$ then

Set $\boldsymbol{x}^* = \boldsymbol{x}_k$ and go to step 4

Else

Calculate $\boldsymbol{d}_k$ by solving $B_k \boldsymbol{d}_k = -\nabla f(\boldsymbol{x}_k)$

Find optimal λ_k by solving

$$\min_{\lambda > 0} f(\boldsymbol{x}_k + \lambda \boldsymbol{d}_k)$$

Set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \lambda_k \boldsymbol{d}_k$ and calculate $\nabla f(\boldsymbol{x}_{k+1})$

Update $\boldsymbol{y}_k = \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k)$

and $\boldsymbol{s}_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k$

Update $B_{k+1} = B_k + \frac{\boldsymbol{y}_k \boldsymbol{y}_k^T}{\boldsymbol{y}_k^T \boldsymbol{s}_k} - \frac{B_k \boldsymbol{s}_k \boldsymbol{s}_k^T B_k}{\boldsymbol{s}_k^T B_k \boldsymbol{s}_k}$

Update iteration counter $k = k + 1$

3 End while

OUTPUT: ('Method failed to converge after N iterations')

4 Stop

Example with Matlab

The system of equations

$$\boldsymbol{F}(\boldsymbol{x}) = \begin{pmatrix} x_1^3 + 3x_2^2 - 21 \\ x_1^2 + 2x_2 + 2 \end{pmatrix} = \boldsymbol{0},$$

can be presented as an unconstrained minimization problem with the objective function $f(\boldsymbol{x}) = \frac{1}{2}(f_1(\boldsymbol{x})^2 + f_2(\boldsymbol{x})^2)$ where $f_1(\boldsymbol{x}) = x_1^3 + 3x_2^2 - 21$ and $f_2(\boldsymbol{x}) = x_1^2 + 2x_2 + 2$. This optimization problem is solved using the BFGS method with the starting point $(1.5, -1.5)^T$, tolerance $\epsilon = 10^{-6}$, and maximum number of iterations $N = 50$. The function attained the minimum value 0 at point $(1.64, -2.34)$, which is the same as obtained with the Newton's method 3.5. The results are presented in Table 3.6.

iter	x_1	x_2	f(x_1,x_2)	‖grad_f‖
0	1.500000	-1.500000	59.914062	10.946603
1	1.939345	-2.133099	1.118857	6.358860
2	1.662219	-2.347640	0.010344	2.088218
3	1.655115	-2.342404	0.001505	0.229881
4	1.644386	-2.349065	0.000018	0.025405
5	1.643054	-2.349786	0.000000	0.001928
6	1.643038	-2.349787	0.000000	0.000019
7	1.643038	-2.349787	0.000000	0.000000

Table 3.6: Minimization of function $f(\boldsymbol{x}) = \frac{1}{2}(f_1(\boldsymbol{x})^2 + f_2(\boldsymbol{x})^2)$ where $f_1(\boldsymbol{x}) = x_1^3 + 3x_2^2 - 21$ and $f_2(\boldsymbol{x}) = x_1^2 + 2x_2 + 2$ with starting point $(1.5, -1.5)^T$ BFGS method.

Davidon-Fletcher-Powell method

The earliest and one of the best methods to approximate the inverse of the Hessian was first proposed by Davidon (1959) [21] and later developed by Fletcher and Powell (1963) [29]. The main difference between the BFGS and DFP(Davidon–Fletcher–Powell) methods is that in DFP, we try to approximate the inverse of the Hessian, while in BFGS, we approximate the Hessian itself.

The DFP method being described as a "general class of quasi-Newton methods" means that it represents a specific strategy within the broader quasi-Newton framework. Like the BFGS method, the DFP method also generates a sequence $\{\boldsymbol{x}_k\}$ by the iteration scheme

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \lambda_k \boldsymbol{d}_k,$$

where the search direction is obtained by $\boldsymbol{d}_k = -D_k \nabla f(\boldsymbol{x}_k)$ instead of solving $B_k \boldsymbol{d}_k = -\nabla f(\boldsymbol{x}_k)$ as in the BFGS method and optimal λ_k by solving

$$\min_{\lambda > 0} f(\boldsymbol{x}_k + \lambda \boldsymbol{d}_k).$$

Essentially, the negative gradient direction is modified using D_k, which is $n \times n$ symmetric positive definite matrix that approximates the inverse Hessian. This approach is also referred to as *rank two correction procedure* because the approximated inverse of the Hessian matrix D_k is updated by adding two symmetric matrices of rank one each. In some sources, the DFP method is sometimes referred to as a "variable metric method" because it can be interpreted as choosing the steepest descent step in a transformed space, where the transformation is based on the Cholesky factorization of the matrix D_k [34].

Computational algorithm

Algorithm 12: Davidon-Fletcher-Powell method [9]

Input: Starting point $\boldsymbol{x}_0$, tolerance $\epsilon > 0$, initial positive definite matrix D_0, maximum number of iterations N

Output: Approximate solution $\boldsymbol{x}^*$

1 Calculate $\nabla f(\boldsymbol{x}_0)$ and set $k = 0$

2 While $(k < N)$

If $\|\nabla f(\boldsymbol{x}_k)\| < \epsilon$ then

Set $\boldsymbol{x}^* = \boldsymbol{x}_k$ and go to step 4

Else

Calculate $\boldsymbol{d}_k = -D_k \nabla f(\boldsymbol{x}_k)$

Find optimal λ_k by solving

$$\min_{\lambda > 0} f(\boldsymbol{x}_k + \lambda \boldsymbol{d}_k)$$

Set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \lambda_k \boldsymbol{d}_k$ and calculate $\nabla f(\boldsymbol{x}_{k+1})$

Update $\boldsymbol{y}_k = \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k)$

and $\boldsymbol{s}_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k$

Update $D_{k+1} = D_k + \frac{\boldsymbol{s}_k \boldsymbol{s}_k^T}{\boldsymbol{y}_k^T \boldsymbol{s}_k} - \frac{D_k \boldsymbol{y}_k \boldsymbol{y}_k^T D_k}{\boldsymbol{y}_k^T D_k \boldsymbol{y}_k}$

Update iteration counter $k = k + 1$

3 End while

OUTPUT: ('Method failed to converge after N iterations')

4 Stop

Example with Matlab

The system of equations

$$\boldsymbol{F}(\boldsymbol{x}) = \begin{pmatrix} x_1^3 + 3x_2^2 - 21 \\ x_1^2 + 2x_2 + 2 \end{pmatrix} = \boldsymbol{0},$$

can be presented as an unconstrained minimization problem with the objective function $f(\boldsymbol{x}) = \frac{1}{2}(f_1(\boldsymbol{x})^2 + f_2(\boldsymbol{x})^2)$ where $f_1(\boldsymbol{x}) = x_1^3 + 3x_2^2 - 21$ and $f_2(\boldsymbol{x}) = x_1^2 + 2x_2 + 2$. This optimization problem is solved using the DFP method with the starting point $(1.5, -1.5)^T$, tolerance $\epsilon = 10^{-6}$, and maximum number of iterations $N = 50$. The function attained the minimum value 0 at point $(1.64, -2.34)$, which is the same as obtained in the Newton's method (3.5) and the BFGS method (3.6). The results are presented in Table 3.7.

Iter	x_1	x_2	f(x_1,x_2)	‖grad_f‖
0	1.500000	-1.500000	59.914062	10.946603
1	1.939345	-2.133099	1.118857	6.358860
2	1.662218	-2.347641	0.010344	2.088270
3	1.655115	-2.342404	0.001505	0.229870
4	1.645347	-2.348468	0.000052	0.039405
5	1.643060	-2.349788	0.000000	0.003100
6	1.643038	-2.349787	0.000000	0.000049
7	1.643038	-2.349787	0.000000	0.000000

Table 3.7: Minimization of function $f(\boldsymbol{x}) = \frac{1}{2}(f_1(\boldsymbol{x})^2 + f_2(\boldsymbol{x})^2)$ where $f_1(\boldsymbol{x}) = x_1^3 + 3x_2^2 - 21$ and $f_2(\boldsymbol{x}) = x_1^2 + 2x_2 + 2$ with starting point $(1.5, -1.5)^T$ DFP method.

Convergence

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex and twice differentiable function satisfying the following three conditions:

1. ∇f is Lipschitz continuous on $\mathbb{R}^n$ with parameter $L > 0$.
   This condition implies that the Hessian is bounded above, ensuring that f is sufficiently smooth so that the error in a quadratic approximation is controlled.

2. f is strongly convex (see equation (1.3)) with parameter $m > 0$.
   This ensures that the curvature is not too flat and provides a uniform rate of descent in a neighborhood of the minimizer.

3. H is Lipschitz continuous on $\mathbb{R}^n$ with parameter $M > 0$.
   The condition further controls the change in curvature. This smoothness property is crucial for establishing that the quasi-Newton approximations (whether in the BFGS or DFP update) remain close to the true Hessian, at least asymptotically.

Then, both BFGS and DFP converge globally [16]. Furthermore, for all $k \geq k_0$, we have

$$\|\boldsymbol{x}_k - \boldsymbol{x}^*\| \leq c_k \|\boldsymbol{x}_{k-1} - \boldsymbol{x}^*\|,$$

where $c_k \to 0$ as $k \to \infty$. Here k_0 and c_k depend on L, m and M. This means that quasi-Newton methods have a superlinear convergence rate [46] (see Definition 5).

3.3.4 Conjugate gradient method

The conjugate gradient method was originally proposed by Hestenes and Stiefel in 1952 for solving a system of linear equations [36]. It was originally suggested as a direct search method for solving a system of linear equations, but later, due to the preconditioning technique, it is now considered an iterative method. An extension of the conjugate gradient method for solving a system of nonlinear equations and

unconstrained optimization problems was suggested in [29, 30]. The conjugate gradient method is not as efficient as the quasi-Newton methods. However, it is quite effective when the problem is sufficiently large.

The basic idea of the conjugate gradient method is to start with a starting point $\boldsymbol{x}_0 \in \mathbb{R}^n$ and generate a sequence $\{\boldsymbol{x}_k\}$ by setting

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \lambda_k \boldsymbol{d}_k \quad \text{for } k = 0, 1, \ldots$$

where $\boldsymbol{d}_k$ is search direction and $\lambda_k > 0$ is step length. The initial search direction $\boldsymbol{d}_0$ can be defined as

$$\boldsymbol{d}_0 = -\nabla f(\boldsymbol{x}_0)$$

and for subsequent search directions, we use the formula

$$\boldsymbol{d}_{k+1} = -\nabla f(\boldsymbol{x}_{k+1}) + \alpha_k \boldsymbol{d}_k,$$

where $\alpha_k > 0$ is a deflection parameter. For a quadratic objective function directions $\boldsymbol{d}_k$, $k = 0, 1, 2, \ldots$ are conjugate. The deflection parameter can be calculated by one of the three formulas named Fletcher Reeves (FR), Polak-Ribiere (PR), or Hestenes-Stiefel (HS) as follows:

$$\alpha_k(FR) = \frac{\nabla f(\boldsymbol{x}_k)^T \nabla f(\boldsymbol{x}_k)}{\nabla f(\boldsymbol{x}_{k-1})^T \nabla f(\boldsymbol{x}_{k-1})} \tag{3.25}$$

$$\alpha_k(PR) = \frac{\nabla f(\boldsymbol{x}_k)^T (\nabla f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}_{k-1}))}{\nabla f(\boldsymbol{x}_{k-1})^T \nabla f(\boldsymbol{x}_{k-1})} \tag{3.26}$$

$$\alpha_k(HS) = \frac{\nabla f(\boldsymbol{x}_k)^T (\nabla f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}_{k-1}))}{\boldsymbol{d}_{k-1}^T (\nabla f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}_{k-1}))}. \tag{3.27}$$

Unlike in the previous methods, the step length is now searched with an inexact line search. In an unconstrained minimization problem, the Wolfe-Powell conditions are a set of inequalities for performing the inexact line search. The Wolfe-Powell rule states that during each iteration k the step length λ_k along the direction $\boldsymbol{d}_k$, must satisfy

$$f(\boldsymbol{x}_k + \lambda_k \boldsymbol{d}_k) \leq f(\boldsymbol{x}_k) + \rho \lambda_k \nabla f(\boldsymbol{x}_k)^T \boldsymbol{d}_k \tag{3.28}$$

and

$$\nabla f(\boldsymbol{x}_k + \lambda_k \boldsymbol{d}_k)^T \boldsymbol{d}_k \geq \sigma \nabla f(\boldsymbol{x}_k)^T \boldsymbol{d}_k \tag{3.29}$$

for some parameters $\rho \in (0, 1)$ and $\sigma \in (\rho, 1)$, which are fixed in advance. Each value λ_k can be determined through an iterative search to satisfy Wolfe conditions (3.28) and (3.29). Condition (3.28) ensures that if the algorithm is taking a larger step length, then it has a larger decrease in function value. In simple words, this condition keeps an upper limit on step length. On the other hand, (3.29) makes sure that the algorithm does not take an excessively short step, effectively setting a lower bound on the step length. Both conditions ensure an optimal step length, as a small step length may not guarantee convergence, while a larger step length could lead to faster convergence. If

$$|\nabla f(\boldsymbol{x}_k + \lambda_k \boldsymbol{d}_k)^T \boldsymbol{d}_k| \leq \sigma |\nabla f(\boldsymbol{x}_k)^T \boldsymbol{d}_k| \quad \text{for fixed } \sigma \in (\rho, 1), \tag{3.30}$$

then (3.28) and (3.30) together form the so-called strong Wolfe conditions [45]. The only difference with the Wolfe-Powell rule is that we no longer allow the derivative $\phi'(\lambda_k)$ to be too positive where $\phi(\lambda) = f(\boldsymbol{x}_k + \lambda \boldsymbol{d}_k)$ [45]. Hence, we exclude points that are far from the stationary points of ϕ.

Computational algorithm

Algorithm 13: Conjugate gradient method [9]

Input: Starting point $\boldsymbol{x}_0$, tolerance $\epsilon > 0$, maximum number of iterations N, parameters $\rho \in (0, 1)$ and $\sigma \in (\rho, 1)$

Output: Approximate solution $\boldsymbol{x}^*$

1 Set $k = 0$

2 While $(k < N)$

Calculate $\nabla f(\boldsymbol{x}_k)$

If $\|\nabla f(\boldsymbol{x}_k)\| < \epsilon$ then

Set $\boldsymbol{x}^* = \boldsymbol{x}_k$ and go to step 4

Else

Calculate

$$\boldsymbol{d}_k = \begin{cases} -\nabla f(\boldsymbol{x}_k), & k = 0 \\ -\nabla f(\boldsymbol{x}_k) + \alpha_k \boldsymbol{d}_{k-1}, & k \geq 1 \end{cases}$$

where α_k is fixed to be one of the variants (3.25), (3.26) or (3.27)

Find $\lambda_k > 0$ satisfying Wolfe conditions (3.28) and (3.29)

Update $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \lambda_k \boldsymbol{d}_k$ and iteration counter $k = k + 1$

3 End while

OUTPUT: ('Method failed to converge after N iterations')

4 Stop

Example with Matlab

For the purpose of diversifying the sample problem, we have selected a different test problem. A function $f(\boldsymbol{x}) = x_1^2 + \frac{1}{2}x_2^2 + 3$ is minimized using the conjugate gradient method with starting point $(1, 1)^T$, tolerance 10^{-6} and the maximum number of iterations $N = 50$. The deflection parameter α_k, is determined using the Fletcher-Reeves (FR) formula as given in equation (3.25), with the parameters $\rho = 0.001$ and $sigma = 0.9$. Function attained the minimum value 3 at point $(0, 0)$, which is the same as in the steepest descent method 3.4. The results are presented in Table 3.8.

Iter	x_1	x_2	f(x_1, x_2)	||grad_f||
0	1.000000	1.000000	4.500000	2.236068
1	-0.111111	0.444444	3.111111	0.496904
2	0.000000	0.000000	3.000000	0.000000

Table 3.8: Minimization of function $f(\boldsymbol{x}) = x_1^2 + \frac{1}{2}x_2^2 + 3$ with starting point $\boldsymbol{x}_0 = (1, 1)^T$ using conjugate gradient method.

Convergence

The following lemma and theorem present theoretical justification for the convergence of the conjugate gradient method.

Lemma 3. *[34] Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable. Let $\boldsymbol{d}_k \in \mathbb{R}^n$ be a descent direction at the current iteration point $\boldsymbol{x}_k \in \mathbb{R}^n$, and assume that f is bounded below along the ray $\boldsymbol{x}_k + \lambda \boldsymbol{d}_k$ where $\lambda > 0$. If $0 < \rho < \sigma < 1$, then there exists an interval of step lengths satisfying Wolfe condition (3.28) and (3.29), and the strong Wolfe conditions (3.28) and (3.30).*

Proof. Please refer to Lemma 3.1 [34]. □

Theorem 12. *[34] Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable, Lipschitz continuous and bounded on $\mathbb{R}^n$ and each λ_k satisfies (3.28) and (3.30) with $0 < \rho < \sigma < \frac{1}{2}$. Then*

$$\liminf_{k \to \infty} \|\nabla f(\boldsymbol{x}_k)\| = 0, \tag{3.31}$$

where $\{\boldsymbol{x}_k\}$ is a sequence of approximate solution generated by Algorithm 13.

Chapter 4

Hybrid Methods

All the iterative numerical methods discussed in the preceding chapters to solve a system of equations yield approximate solutions, as in most cases, exact solutions cannot be determined. These methods involve successive approximations of the solution that progressively converge towards the true solution of one equation or a system of equations.

It is well known that Newton's method is one of the most traditional methods for solving either one equation or a system of equations, but its convergence is very sensitive to the starting point. To overcome the starting point problem and achieve faster convergence, many adaptive hybrid methods have been proposed [12, 56]. Some of these algorithms are complex and computationally expensive (requiring high processing time and significant computing power), but they are still widely used as modern computing systems continue to become more affordable.

In this chapter, we consider a system of nonlinear equations (2.1) formulated in Section 2.1. As mentioned at the beginning of Chapter 3, in an indirect optimization-based approach, the system of nonlinear equations (2.1) is first reformulated as the optimization problem (3.1) before it can be solved.

In this chapter, we propose a hybrid algorithm to solve the system of nonlinear equations (2.1) as the optimization problem (3.1). We generally follow the idea of hybriziation as a combination of two search algorithms [56]. The suggested algorithm is a combination of Newton's method (see Subsection 3.3.2) and the conjugate gradient method (see Subsection 3.3.4). We use Newton's method to increase the convergence rate and the conjugate gradient method to satisfy global convergence and low memory requirements. During each iteration of the hybrid method, the step length is determined in the direction that is the combination of both Newton's and conjugate gradient directions.

4.1 Preliminaries

To make this chapter self-contained, we start with a brief reminder of the general idea of the iterative numerical method. It is well known that, with a given starting point $\boldsymbol{x}_0$, optimization methods generate a sequence of estimates $\{\boldsymbol{x}_k\}$, progressively approaching a solution within a specified tolerance. The main behavior of the algorithm is that at every iteration k, the iteration point $\boldsymbol{x}_k$ moves steadily towards

the neighborhood of a local minimizer. Execution of the algorithm can be stopped when $\boldsymbol{x}_k$ satisfies the stopping criterion

$$\|\nabla f(\boldsymbol{x}_k)\| < \epsilon,$$

where ϵ is a predefined positive tolerance parameter. In this case, $\boldsymbol{x}_k$ is the approximate solution. Let $\boldsymbol{x}_k$ be the iteration point, $\boldsymbol{d}_k$ be the search direction, and λ_k be the step length at k^{th} iteration. Then the next iteration point $\boldsymbol{x}_{k+1}$ can be obtained by

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \lambda_k \boldsymbol{d}_k.$$

Multiple strategies have been employed to find a suitable step length from the current iterate to the next one, with two commonly used approaches being the trust region method and line search method. This thesis focuses solely on the line search strategy, leaving other methods for future exploration.

4.1.1 Search direction

In Subsection 3.3, we have already seen different ways to select the search direction in gradient-based methods. For example, in the steepest descent method (see Subsection 3.3.1) the search direction is given by $\boldsymbol{d}_k = -\nabla f(\boldsymbol{x}_k)$, but for general gradient-based methods, the search direction can be defined as

$$\boldsymbol{d}_k = -B_k^{-1} \nabla f(\boldsymbol{x}_k),$$

where B_k is a suitable nonsingular and symmetric matrix. For Newton's method (see Subsection 3.3.2)

$$\boldsymbol{d}_k = -H_k^{-1} \nabla f(\boldsymbol{x}_k) \tag{4.1}$$

where H_k is the exact Hessian of f at $\boldsymbol{x}_k$. For the quasi-Newton method (see Subsection 3.3.3), B_k is an approximation of the Hessian H_k updated at every iteration. In the conjugate gradient method (see Subsection 3.3.4)

$$\boldsymbol{d}_k = \begin{cases} -\nabla f(\boldsymbol{x}_k), & k = 0 \\ -\nabla f(\boldsymbol{x}_k) + \alpha_{k-1} \boldsymbol{d}_{k-1}, & k \geq 1, \end{cases} \tag{4.2}$$

where α_k is the deflection parameter. In this chapter, we are using the Fletcher-Reeves formula (3.25), leading to

$$\alpha_k = \frac{\nabla f(\boldsymbol{x}_k)^T \nabla f(\boldsymbol{x}_k)}{\nabla f(\boldsymbol{x}_{k-1})^T \nabla f(\boldsymbol{x}_{k-1})}$$

although it would be possible to use also (3.26) or (3.27).

4.1.2 Line search

Line search is an umbrella term for a class of one of the most effective conventional routines incorporated into optimization methods. The line search tries to find the

optimal step length λ_k along the search direction $\boldsymbol{d}_k$ after it is specified and fixed. The accuracy and speed of optimization methods depend on the search direction $\boldsymbol{d}_k$ and step length λ_k. The direction vector $\boldsymbol{d}_k$ decides in which direction we move at the current iteration point $\boldsymbol{x}_k$, and λ_k decides how far we move along $\boldsymbol{d}_k$. There are two types of line search methods to find the step length: exact and inexact ones.

To find λ_k using the exact line search we need to solve

$$\min_{\lambda>0} f(\boldsymbol{x}_k + \lambda \boldsymbol{d}_k). \tag{4.3}$$

To solve (4.3), the methods presented in Subsection 3.2 can be used. On the other hand, choosing any step length λ_k which decreases the function value at the next iterate, such that

$$f(\boldsymbol{x}_k + \lambda_k \boldsymbol{d}_k) < f(\boldsymbol{x}_k) \tag{4.4}$$

is called an inexact line search. In multi-dimensional real-world problems, finding the exact step length is difficult, so the inexact line search is quite popular. It is also computationally less expensive to perform.

Note that the decreasing property of the function value, in other words (4.4), is not sufficient for the optimization method to converge to the solution, as the decrease in the function value may sometimes be insufficient. Hence, inexact line searches typically use some extra conditions to move sufficiently in the direction $\boldsymbol{d}_k$. There exists a variety of inexact methods to estimate a suitable step length λ_k, such as the Armijo rule, the Goldstein rule, and the Wolfe-Powell rule.

In this work, we are using the Wolfe-Powell rules (3.28)-(3.29) during each iteration k, where we start with the initial step length $\lambda_k = 1$, for fixed values $\rho \in (0, 1)$ and $\sigma \in (\rho, 1)$. After this, we decrease λ_k by some small quantity until we find the largest value that satisfies the conditions (3.28)-(3.29).

4.2 Conjugate Gradient and Newton (CGN) algorithm

In this section, we present a new hybrid method that combines the Conjugate Gradient (CG) method and Newton's (N) method to solve (3.1). We denote the combination as CGN. This hybrid method is based on the method presented in [56], but instead of the gradient and Newton's method, it combines the conjugate gradient and Newton's methods. The main idea of Algorithm 14 is to use as a search direction a linear combination of the Newton's direction (4.1) $\boldsymbol{d}_{1,k}$ and the conjugate gradient direction $\boldsymbol{d}_{2,k}$ (4.2) at $\boldsymbol{x}_k$.

In Algorithm 14, δ_0, η, ρ, σ are tuning parameters such that $0 < \delta_0 < 1$, $0 < \eta < 1$, $0 < \rho < \frac{1}{2}$ and $\rho < \sigma < 1$. Furthermore, $\gamma_1 > 1$, $\gamma_2 > 1$, $\Lambda_0 \geq 1$ and $b_1 \in (0, 1)$, $1 < b_2 < \frac{1}{\delta_0}$, $b_3 > 1$ are positive constants. In numerical tests, their values have been chosen to be similar to those used in [56].

Algorithm 14: Conjugate Gradient and Newton algorithm (CGN)

Input: Starting point $\boldsymbol{x}_0$, maximum number of iterations N, stopping tolerance $\epsilon > 0$, initial values $\delta_0 \in (0,1)$ and $\Lambda_0 \geq 1$ of modifiable parameters δ and Λ, fixed value parameters $\gamma_1 > 1$, $\gamma_2 > 1$, $0 < \eta < 1$, $0 < \rho < \frac{1}{2}$, $\sigma \in (\rho, 1)$, $b_1 \in (0,1)$, $1 < b_2 < \frac{1}{\delta}$, $b_3 > 1$.

Output: The approximate solution $\boldsymbol{x}^*$

1 Set $k = 0$

2 While $(k < N)$

3 If $\|\nabla f(\boldsymbol{x}_k)\| < \epsilon$ then $\boldsymbol{x}^* = \boldsymbol{x}_k$ is the solution and go to step 17.

4 If $\det(H_k) = 0$ compute the conjugate gradient direction

$$\boldsymbol{d}_{2,k} = \begin{cases} -\nabla f(\boldsymbol{x}_k), & k = 0 \\ -\nabla f(\boldsymbol{x}_k) + \alpha_{k-1}\boldsymbol{d}_{2,k-1}, & k \geq 1. \end{cases} \tag{4.5}$$

where α_{k-1} is a deflection parameter obtained with (3.25) and go to step 10.

5 Compute the Newton's direction $\boldsymbol{d}_{1,k}$ that satisfies $H_k\boldsymbol{d}_{1,k} = -\nabla f(\boldsymbol{x}_k)$ and the conjugate gradient direction $\boldsymbol{d}_{2,k}$ as (4.5)

6 Initialize $\delta = \delta_0$ and $\Lambda = \Lambda_0$. If $|f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k-1})| > \gamma_1$ for $k \geq 1$ and $\|\nabla f(\boldsymbol{x}_k)\| > \gamma_2$ then set $\delta = b_2\delta_0$ go to step 9.

7 If $k = 0$ or $\|\nabla f(\boldsymbol{x}_k)\| \leq \|\nabla f(\boldsymbol{x}_{k-1})\|$, set $\bar{\boldsymbol{x}} = \boldsymbol{x}_k + \boldsymbol{d}_{1,k}$ and go to step 8. Otherwise go to step 9.

8 If $f(\bar{\boldsymbol{x}}) < f(\boldsymbol{x}_k)$ and $\nabla f(\bar{\boldsymbol{x}}) < \eta\nabla f(\boldsymbol{x}_k)$, then set $\delta = b_1\delta_0$.

9 If $\boldsymbol{d}_{1,k}^T\boldsymbol{d}_{2,k} \geq 0$ go to step 11.

10 Use the Wolfe-Powell rules (3.28)-(3.29) to calculate a step length $\lambda_k > 0$ along the direction $\boldsymbol{d}_k = \boldsymbol{d}_{2,k}$. Set $\boldsymbol{s}_k = \lambda_k\boldsymbol{d}_k$ and go to step 14.

11 Calculate ξ such that

$$\xi = \begin{cases} \frac{1}{\Lambda + \|\nabla f(\boldsymbol{x}_k)\|} & k = 0 \\ \frac{1}{\Lambda + |f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k-1})|} & k \geq 1. \end{cases}$$

and set $\boldsymbol{d}_k(\xi) = (1-\xi)\boldsymbol{d}_{2,k} + \xi\boldsymbol{d}_{1,k}$.

12 If $\boldsymbol{d}_k(\xi)^T\boldsymbol{d}_{2,k} < \delta\|\boldsymbol{d}_k(\xi)\| \cdot \|\boldsymbol{d}_{2,k}\|$ set $\Lambda = b_3\Lambda$ and go to step 11.

13 (A) Use the Wolfe-Powell rules (3.28)-(3.29) to get $\lambda_k > 0$ along the direction $\boldsymbol{d}_k = \boldsymbol{d}_{2,k}$ and set $\bar{\boldsymbol{s}}_k = \lambda_k(1-\xi)\boldsymbol{d}_{2,k} + \xi\boldsymbol{d}_{1,k}$. If $f(\boldsymbol{x}_k + \bar{\boldsymbol{s}}_k) \leq f(\boldsymbol{x}_k) - \tau\|\bar{\boldsymbol{s}}_k\|$ and $\lambda_k\|\boldsymbol{d}_{2,k}\| \leq T\|\boldsymbol{d}_{1,k}\|$ set $\boldsymbol{s}_k = \bar{\boldsymbol{s}}_k$. Otherwise $\boldsymbol{s}_k = \lambda_k\boldsymbol{d}_{2,k}$.
(B) Use the Wolfe-Powell rules (3.28)-(3.29) to calculate $\lambda_k > 0$ along the direction $\boldsymbol{d}_k = \boldsymbol{d}_k(\xi)$. Set $\boldsymbol{s}_k = \lambda_k\boldsymbol{d}_k$.

14 Update $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{s}_k$ and $k = k + 1$

15 End while

16 OUTPUT ('Method failed to converge after N iterations')

17 Stop

If at any iteration point $\boldsymbol{x}_k$ in step 4 of the CGN algorithm $\det(H_k) = 0$, then the Newton's direction is not computable. In this case, we proceed only with the conjugate gradient direction $\boldsymbol{d}_{2,k}$, and calculate the step length λ_k at step 10, after which the next iteration point $\boldsymbol{x}_{k+1}$ is calculated. In step 7, we follow Newton's direction in the case of the first iteration or the iteratively decreasing norm of the gradient.

When both the Newton's direction (4.1) and the conjugate gradient direction (4.2) are available, we compute both and verify whether both directions belong to the same orthant (that is, they are close enough) in step 9. If the condition is satisfied, the hybridization process starts; otherwise, we continue using the conjugate gradient direction. In steps 11-12 (both directions are available), we first calculate a parameter ξ (regulated by Λ), which is the coefficient of the convex combination of $\boldsymbol{d}_{1,k}$ and $\boldsymbol{d}_{2,k}$. After this, we check that the hybrid direction deviates enough (regulated by δ) from the conjugate gradient direction and if the answer is no, then we decrease the preference of $\boldsymbol{d}_{1,k}$ in the convex combination in step 11 where the hybrid direction is calculated. By doing that, we increase the importance of $\boldsymbol{d}_{2,k}$ in the scalarised hybrid direction $\boldsymbol{d}_k(\xi)$. In step 13, two different ways of calculating the next iteration point have been discussed. The choice between A and B depends on the optimizer, although both options have been tested numerically. Step 13A is responsible for determining the step length λ_k along the direction of the conjugate gradient. The satisfaction of the conditions in step 13A means that the hybrid direction is a locally descent direction. In step 13B, the step length λ_k is determined along the hybrid direction. In what follows CGN(A) and CGN(B) are versions of Algorithm 14. In CGN(A), the step length λ_k is calculated along $\boldsymbol{d}_{2,k}$, and in the case of CGN(B), the step length λ_k is calculated along the hybrid direction.

The convergence of Algorithm 14 should naturally follow from the convergence property of individual methods used in the hybridization. However, it is left for future research to provide a more detailed explanation.

4.3 Various hybridizations

In addition to the CGN method described in the previous section, we propose other hybrids of different methods. Expanding on the idea of combining two methods originally outlined in [56], we have extended the algorithm discussed in [56] to incorporate various hybrids of the Gradient method (G), Conjugate Gradient method (CG), Newton's method (N) and quasi-Newton method (QN) (see Table 4.1). The combined methods are categorized into two groups based on their similarities. To ensure a comprehensive evaluation, we consider combinations that pair a G or CG method with an N or QN method, resulting in four possible combinations. Furthermore, each method is analyzed using two distinct hybridization strategies, labeled A and B, to account for variations in approach.

Please note that [56] has considered only a combination of the gradient and Newton's methods (GN). Anyway, for our experiment, we implemented GN(A) and GN(B) on our own. In the case of other hybrids considered in Table 4.1, Algorithm 14 was modified according to the requirements of separate methods used in it.

Table 4.1: Hybridizations

Method Group 1	Method Group 2	Stepping direction	Method name
Conjugate Gradient (CG)	Newton's (N)	Conjugate gradient direction (A)	CGN(A)
Conjugate Gradient (CG)	Newton's (N)	Hybrid direction (B)	CGN(B)
Conjugate Gradient (CG)	quasi-Newton (QN)	Conjugate Gradient direction (A)	CGQN(A)
Conjugate Gradient (CG)	quasi-Newton (QN)	Hybrid direction (B)	CGQN(B)
Gradient (G)	Newton's (N)	Gradient direction (A)	GN(A)
Gradient (G)	Newton's (N)	Hybrid direction (B)	GN(B)
Gradient (G)	quasi-Newton (QN)	Gradient direction (A)	GQN(A)
Gradient (G)	quasi-Newton (QN)	Hybrid direction (B)	GQN(B)

In the algorithm version A, a step length λ_k is calculated only in the direction $\boldsymbol{d}_{2,k}$ while in version B, a step length λ_k is calculated in the direction $\boldsymbol{d}_k(\xi)$ which is a convex combination of direction $\boldsymbol{d}_{1,k}$ and $\boldsymbol{d}_{2,k}$.

As outlined in Appendix D, the GN method follows steps similar to those of the CGN Algorithm 14 described earlier. However, in step 4 of the GN method, the direction vector $\boldsymbol{d}_{2,k} = -\nabla f(\boldsymbol{x}_k)$ for all values of k. In other words, the hybrid direction $\boldsymbol{d}_k(\xi)$ is derived as a convex combination of the gradient and Newton's directions.

Conversely, in the GQN method given in appendix C, the hybrid direction $\boldsymbol{d}_k(\xi)$ is also a convex combination of the gradient and Newton's directions. However, since the quasi-Newton method is used instead of the Hessian matrix H_k, the method employs an approximated Hessian B_k, which is iteratively updated using the BFGS method (see subsection 3.3.3).

In the case of the CGQN method given in appendix B, both directions $\boldsymbol{d}_{1,k}$ and $\boldsymbol{d}_{2,k}$ are involved at every iteration. The hybrid direction $\boldsymbol{d}_k(\xi)$ is computed as a convex combination of the conjugate gradient and quasi-Newton directions.

4.4 Numerical experiments and results

In this section, we show the computational performance of the hybrid algorithms on the selected test problems discussed in [3, 56]. In order to support the purpose of this preliminary research experiment, 2 linear and 12 nonlinear problems were carefully chosen to provide a representative variety of different functions. As noted in [3], the purpose of that test problem collection is to provide a sufficiently large set of general test functions for testing an unconstrained optimization algorithm and conducting comparison studies. In this work, only artificial unconstrained optimization test problems are considered as they are relatively easy to manipulate and use in algorithmic design and testing.

Table 4.2 gives the basic description of test problems where n is the dimension of the problem and m is the number of functions. Problems P00 and P01 have the same linear functions but use different starting points. In problems P1-P6, the dimension n is fixed, while in problems P7-P12, it can vary, so the user can choose different numerical values for n and m. In our experiments, we have set $n = 20, 50$ and 100. Each problem with a different value of n is considered a separate test problem. The calculations are carried out in Matlab. For termination criteria, we are using the

stopping tolerance $\epsilon = 10^{-6}$ in condition $\|\nabla f(\boldsymbol{x}_k)\| < \epsilon$ and the maximum number of iterations $N = 500$.

The user-defined parameters (the values are similar to those used in [56] for the GN algorithm and carefully tested with trial and error technique) for the experiment are the following: $\delta_0 = 0.001$, $\Lambda_0 = 1$, $\eta = 0.99$, $\rho = 0.001$, $\sigma = 0.9$, $b_1 = 0.9$, $b_2 = \frac{1}{b_1}$, $b_3 = 1.1$, $\gamma_1 = 2$, $\gamma_2 = 2$, $\tau = 10^{-10}$ and $T = 10^{10}$. However, it is the user's choice to take any values within the defined limits.

Table 4.2: Test problems. [3, 56]

Problem	Function name	Dimension n	Number of functions m
P00	Linear functions	21	21
P01	Linear functions	21	21
P1	Helical Valley function	3	3
P2	Powell Singular function	4	4
P3	Wood function	4	6
P4	Watson function	6	31
P5	Variably dimensional function	n	n+2
P6	Discrete Boundary Value function	n	n
P7	Extended Rosenbrock function	n	n
P8	Trigonometric function	n	n
P9	Axis Parallel Hyper-Ellipsoid function	n	n
P10	Griewangk's function	n	n
P11	Sum of Different Power functions	n	n
P12	Ackley's function	n	n

In Table 4.3, we compare the number of iterations required by GN, CGQN, CGN, and GQN. The number of iterations in bold font shows the minimum number of iterations for each test problem. In most traditional optimization methods, there is typically a consistent relationship between the number of iterations and function evaluations. However, this relationship becomes less predictable in hybrid approaches. For example, in the CGN algorithm (Algorithm 14), the computational behavior varies depending on the availability of the Newton direction $\boldsymbol{d}_{2,k}$ at each iteration. When this direction is unavailable, the algorithm skips certain computations and advances solely along the conjugate gradient direction $\boldsymbol{d}_{1,k}$. As a result, the ratio between iterations and function evaluations is not maintained uniformly throughout the optimization process, highlighting the adaptive nature of hybrid methods.

Table 4.3: Number of iterations for test problems.

Prob.	n	CGN(A)	CGN(B)	CGQN(A)	CGQN(B)	GN(A)	GN(B)	GQN(A)	GQN(B)
P00	21	14	**13**	*	*	15	22	*	41
P01	21	**3**	**3**	*	*	**3**	**3**	*	33
P1	3	**17**	55	*	*	19	28	31	41
P2	4	**20**	**20**	27	*	**20**	26	46	54
P3	4	29	47	*	*	**10**	63	*	88
P4	6	**11**	12	*	*	12	22	27	36
P5	20	8	10	12	10	**7**	9	9	9
	50	**9**	14	15	25	**9**	15	12	15
	100	11	18	12	26	**9**	11	**9**	11
P6	20	**2**	**2**	*	43	**2**	**2**	95	40
	50	**1**	**1**	*	*	**1**	**1**	*	106
	100	**1**	**1**	*	*	**1**	2	2	187
P7	20	27	68	*	*	**17**	48	*	140
	50	**20**	448	*	*	*	67	*	153
	100	**22**	35	*	*	*	65	*	172
P8	20	11	21	185	252	**10**	12	17	33
	50	22	39	*	*	*	**15**	19	44
	100	*	33	*	*	**20**	*	28	*
P9	20	18	19	47	55	**14**	**14**	116	81
	50	21	23	79	122	**15**	**15**	167	96
	100	125	**93**	178	166	*	*	136	350
P10	20	**9**	12	*	45	13	14	137	137
	50	17	34	*	115	**16**	**16**	195	211
	100	*	*	*	*	*	*	**239**	263
P11	20	20	*	**5**	8	14	*	86	138
	50	*	*	**5**	**5**	*	*	44	47
	100	*	*	**5**	**5**	*	*	35	83
P12	20	**4**	7	53	47	5	7	5	7
	50	**3**	5	34	43	**3**	5	5	5
	100	**4**	5	28	33	**4**	5	5	5

* means the method failed to converge within the maximum number of iterations.

There exist quite many failures (marked by *). In Table 4.3, cases CGQN(A) and CGQN(B) had the most failures. In general, algorithmic failures in optimization and equation-solving methods can often be attributed to three key issues: slow convergence rate, suboptimal choice of starting point, and potential computational singularities.

1. **Slow convergence rate:** Some algorithms may progress toward the solution at a very gradual pace, particularly when the problem involves ill-conditioned functions or a landscape with shallow gradients. This slow rate can significantly increase computational time and make the method impractical for large-scale problems.

2. **Suboptimal choice of starting point:** The starting point plays a crucial role in iterative methods. In most of our test problems, a predefined starting point was provided. A poor choice can lead the algorithm to converge to a local extremum instead of the global solution or, worse, prevent it from converging at all. Sensitivity to the starting point is particularly pronounced in non-convex problems or when dealing with systems that have multiple solutions.

3. **Potential computational singularities:** These arise from numerical issues such as division by values approaching zero, poorly conditioned matrices, or

inaccuracies in calculating derivatives or Jacobian matrices. Such singularities can cause the algorithm to break down, fail to update correctly, or produce unreliable results.

The number of iterations can depend on the design of the code, so we have also included the total number of function evaluations reported in Table 4.4 to analyze the efficiency of each algorithm more precisely. The total number of function evaluations includes those conducted during the iteration process of the algorithm as well as additional evaluations required for determining the step length. The number of function evaluations in bold font in Table 4.4 shows the minimum number of function evaluations for each test problem.

Table 4.4: Total number of function evaluations.

Prob.	n	CGN(A)	CGN(B)	CGQN(A)	CGQN(B)	GN(A)	GN(B)	GQN(A)	GQN(B)
P00	21	73	63	*	*	**56**	57	*	104
P01	21	12	**5**	*	*	9	**5**	*	73
P1	3	**84**	336	*	*	93	103	150	143
P2	4	78	**77**	107	*	77	79	205	142
P3	4	204	311	*	*	**51**	239	*	334
P4	6	54	59	*	*	**52**	73	132	100
P5	20	66	61	96	61	**60**	**60**	70	**60**
	50	95	115	153	218	**94**	99	132	99
	100	135	161	142	247	113	110	114	**107**
P6	20	5	4	*	103	**3**	4	318	94
	50	3	**2**	*	*	3	**2**	*	249
	100	3	**2**	*	*	3	**2**	*	449
P7	20	195	404	*	*	**89**	169	*	419
	50	**116**	9244	*	*	*	256	*	518
	100	**126**	182	*	*	*	250	*	648
P8	20	61	119	1112	1553	50	**45**	85	131
	50	188	206	*	*	*	**71**	114	208
	100	*	214	*	*	**140**	*	196	*
P9	20	54	57	105	157	**38**	**38**	242	172
	50	71	76	191	278	**45**	**45**	349	207
	100	307	**236**	446	388	*	*	320	725
P10	20	**26**	50	*	101	28	30	276	276
	50	96	178	*	427	**36**	**36**	395	427
	100	*	*	*	*	*	*	**491**	538
P11	20	43	*	**19**	22	31	*	175	279
	50	*	*	15	**11**	*	*	91	98
	100	*	*	15	**11**	*	*	74	172
P12	20	**16**	18	109	99	20	17	20	17
	50	**9**	11	71	92	**9**	11	15	11
	100	12	**11**	60	73	12	**11**	15	**11**

* means the method failed to converge within the maximum number of iterations.

4.5 Analysis of the results

In Table 4.3, we have seen that there are some places where the method failed to converge (marked by *). In this situation, performing analysis directly on the number of iterations is not informative. Therefore, we introduce a special ranking system as summarized in Table 4.5.

Table 4.5: Number of iterations with refined ranks for the test problems.

Prob.	n	CGN(A)	CGN(B)	CGQN(A)	CGQN(B)	GN(A)	GN(B)	GQN(A)	GQN(B)
P00	21	2	**1**	11	11	3	4	9	5
P01	21	**2.5**	**2.5**	11	11	**2.5**	**2.5**	10	5
P1	3	**1**	6	11	10	2	3	4	5
P2	4	**2**	**2**	5	10	**2**	4	6	7
P3	4	2	3	11	11	**1**	4	11	5
P4	6	**1**	2.5	10	11	2.5	4	5	6
P5	20	2	6.5	8	6.5	**1**	4	4	4
	50	**1.5**	4	6	8	**1.5**	6	3	6
	100	4	7	6	8	**1.5**	6	3	6
P6	20	**2.5**	**2.5**	11	6	**2.5**	**2.5**	7	5
	50	**2.5**	**2.5**	11	11	**2.5**	**2.5**	10	5
	100	**1.5**	**1.5**	10	10	**1.5**	4	10	5
P7	20	2	4	11	11	**1**	3	11	5
	50	**1**	4	11	10	11	2	11	3
	100	**1**	2	11	11	11	3	11	4
P8	20	2	5	7	8	**1**	3	4	6
	50	3	4	10	10	10	**1**	2	5
	100	10	3	11	10	**1**	10	2	4
P9	20	3	4	5	6	**1.5**	**1.5**	8	7
	50	3	4	5	7	**1.5**	**1.5**	8	6
	100	2	**1**	5	4	10	10	3	6
P10	20	**1**	2	11	5	3	4	6.5	6.5
	50	3	4	11	5	**1.5**	**1.5**	6	7
	100	10	10	11	11	10	10	**1**	2
P11	20	4	9	**1**	2	3	10	5	6
	50	9	9	**1.5**	**1.5**	10	10	3	4
	100	9	9	**1.5**	**1.5**	10	10	3	4
P12	20	**1**	5	8	7	2.5	5	2.5	5
	50	**1.5**	4.5	7	8	**1.5**	4.5	4.5	4.5
	100	**1.5**	4.5	7	8	**1.5**	4.5	4.5	4.5

The algorithm with the least number of iterations has the highest rank, starting with the lowest number 1. In case two or more algorithms have the same number of iterations, they receive the same ranking, which is the mean value of what they would have obtained under ordinal ranking. By ranking in this manner, we ensure that the sum of ranks remains the same as under ordinal ranking. We assign rank 9 to a method when it converges within 750 iterations with the stopping tolerance $\epsilon = 10^{-6}$. We assign rank 10 if a method converges within 750 iterations when the stopping tolerance is reduced to 10^{-3}. If a method is not capable of finding an optimal solution either with an increased number of iterations or with a reduced value of tolerance, then we assign the lowest rank to it, in other words, the largest possible ranking value 11.

Table 4.6: Refined ranking averaged for the number of iterations.

Parameter	CGN(A)	CGN(B)	CGQN(A)	CGQN(B)	GN(A)	GN(B)	GQN(A)	GQN(B)
Average	**3.05**	4.30	8.10	7.88	3.82	4.63	5.88	5.05
Variance	7.33	6.11	10.72	9.05	13.59	8.69	10.68	**1.37**
SD	2.66	2.43	3.22	2.96	3.63	2.90	3.21	**1.15**

SD means Standard Deviation

In Table 4.6, we have calculated the average ranking for the number of iterations based on Table 4.5 and marked the smallest number with bold in the row. We can say that CGN(A) is the most efficient method with the smallest average rank for the

number of iterations. From Table 4.4, we see that GQN(B) is the most consistent hybrid method since it solves almost all the problems without any relaxation except in P8 when $n = 100$. In addition, from Table 4.6, we notice that CQN(B) has the smallest variance and standard deviation of the average ranking.

In evaluating the performance of methods, we assign weighted rankings to emphasize the significance of failures, particularly when a method is unable to solve even relatively simple problems. To compute the Weighted Iterative Ranking (WIR), three Weighted Iteration Parameters (WIPs) are utilized: $\text{wip}_1 = 0.5$, $\text{wip}_2 = 0.3$, $\text{wip}_3 = 0.2$. These weights reflect the relative importance of iteration parameters in assessing the method's performance. For example, considering CGN(A) for problem P5 from Table 4.5 the WIR is calculated as:

$$\text{WIR} = 0.5 \times 2 + 0.3 \times 1.5 + 0.2 \times 4 = 2.25. \tag{4.6}$$

Similarly, the Weighted Function Ranking (WFR) can be calculated for the number of function evaluations, using Weighted Function Parameters (WFP) to emphasize the importance of specific evaluations. For example, considering CGN(A) for problem P5, the WFP values are chosen as $\text{wfp}_1 = 0.5$, $\text{wfp}_2 = 0.3$, $\text{wfp}_3 = 0.2$,. Applying these weights, the WFR is determined as:

$$\text{WFR} = 0.5 \times 6 + 0.3 \times 2 + 0.2 \times 5 = 4.6. \tag{4.7}$$

In the final Grand Rank (GR) calculation, equal weighting of the WIR and the WFR is applied, with $\text{wip} = 0.5$ and $\text{wfp} = 0.5$. These parameters can be adjusted by decision-makers according to their priorities. Using these weights, the GR is calculated as:

$$\text{GR} = 0.5 \times 2.25 + 0.5 \times 4.6 \approx 3.43.$$

This methodology provides a balanced and adaptable framework for evaluating the performance of methods across multiple criteria.

Table 4.7: Equally weighted grand ranks for test problems.

Prob.	CGN(A)	CGN(B)	CGQN(A)	CGQN(B)	GN(A)	GN(B)	GQN(A)	GQN(B)
P00	3	**2**	9.5	9.5	3	**2**	9	5
P01	3	**2**	9.5	9.5	**2**	3	8.5	5
P1	**1**	6	9.5	9	2	3	4.5	4.5
P2	2.5	1.75	5	9	**1.75**	4	6.5	6.5
P3	2	3.5	9.5	9.5	**1**	3.5	9.5	5
P4	**1.5**	2.75	9	9.5	1.75	4	5.5	5.5
P5	3.43	5.5	7.15	6.75	**1.58**	3.53	4.65	3.43
P6	3.03	**2.15**	7.75	6	2.28	2.4	8	5
P7	**1.75**	3.6	9.5	9.35	5.25	2.45	9.5	4.2
P8	3.95	4.3	8.1	8.5	3.65	3.1	**3**	5.3
P9	**2.8**	3.4	5	5.9	3	3	7	6.5
P10	**3.2**	4.5	9.5	6.13	3.5	4	5.25	5.68
P11	6.25	8.5	4.25	**1.63**	6	9	4	5
P12	**1.38**	4.18	7.5	7.5	2.6	3.8	4.63	3.8
Average	**2.77**	3.87	7.91	7.7	2.81	3.63	6.39	5.03

Based on the final GR analysis (Table 4.7), it can be concluded that CGN(A) is the most reliable and efficient method for the test problems considered. The GR

was calculated using equal weight parameters wip = 0.5 and wfp = 0.5. Alternative convex combinations of weights were also examined to assess the stability of the methods. Table 4.8 shows a convex combination of weights in each column where wip is used for iteration rank and wfp for function ranks.

Table 4.8: Possible weightings for grand rank calculation.

Parameter	C1	C2	C3	C4	C5	C6	C7	C8	C9
wip	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
wfp	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9

Let us consider only the test problem P5 and generate a GR table for all weight combinations from Table 4.8. For the CGN(A) method in problem P5, WIR was 2.25 and WFR was 4.6. Thus, with the weights C1, the GR for CGN(A) is

$$0.9 \times 2.25 + 0.1 \times 4.6 \approx 2.48 \tag{4.8}$$

Table 4.9: Grand ranking with all possible weights for P5.

Weights	CGN(A)	CGN(B)	CGQN(A)	CGQN(B)	GN(A)	GN(B)	GQN(A)	GQN(B)
C1	2.48	5.78	7.03	7.15	1.31	4.38	3.49	4.36
C2	2.72	5.71	7.06	7.05	1.38	4.17	3.78	4.13
C3	2.95	5.64	7.09	6.95	1.44	3.95	4.07	3.89
C4	3.19	5.57	7.12	6.85	1.51	3.74	4.36	3.66
C5	3.43	5.5	7.15	6.75	1.58	3.53	4.65	3.43
C6	3.66	5.43	7.18	6.65	1.64	3.31	4.94	3.19
C7	3.89	5.36	7.21	6.55	1.70	3.09	5.23	2.95
C8	4.13	5.29	7.24	6.45	1.77	2.88	5.52	2.72
C9	4.36	5.22	7.27	6.35	1.83	2.66	5.81	2.48

Next, we plotted the GR for each algorithm with all possible weight combinations from Table 4.9 to analyze how the weights affect the GR.

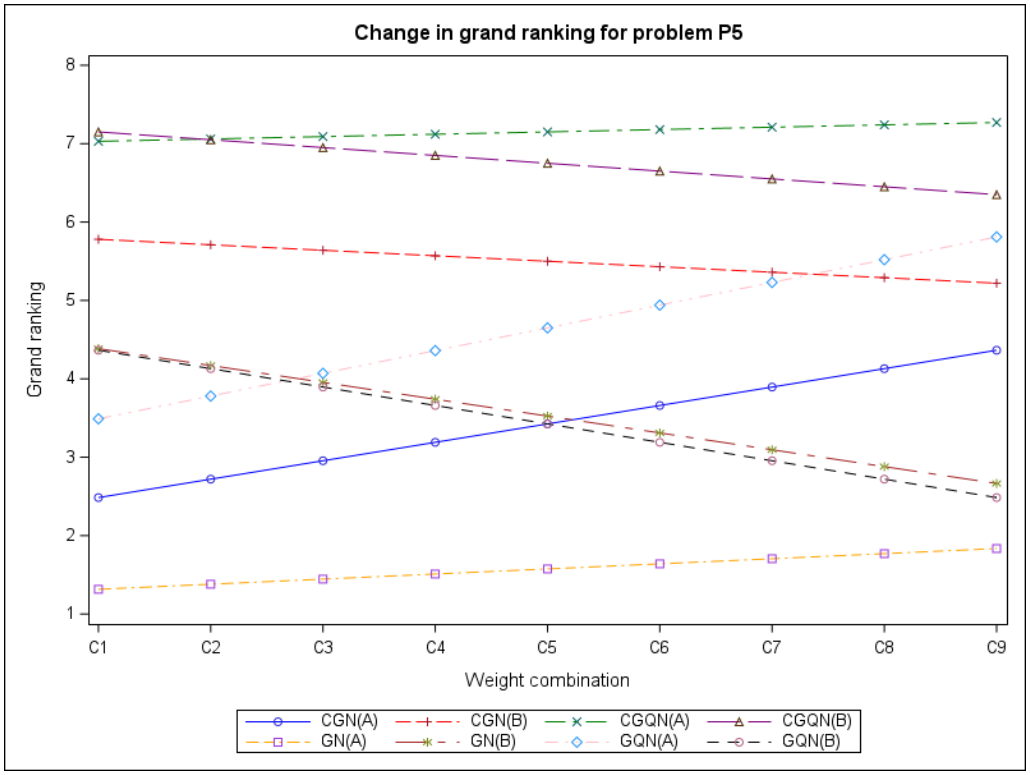

Figure 4.1: Change in grand ranking for P5 after applying weight combinations C1-C9.

The intersection of lines in Figure 4.1 indicates that grand rankings depend on the combinations of wip and wfp, but the trends remain relatively stable. We have made similar observations in other test problems. In some cases, lines have a positive slope, which means WFR is higher than WIR, and if the slope is negative, that means WIR is higher than WFR.

4.6 Hypothesis testing

In this section, we check whether all the methods have a statistically significant difference or not. Let us state two mutually exclusive, Null (H_0) and Alternative (H_1) hypotheses:

H_0 : Two methods have the same performance.

H_1 : Two methods perform differently.

The Wilcoxon signed-rank test [58] is a nonparametric statistical method used to evaluate the validity of a hypothesis. Unlike the conventional T-test, it does not assume data normality, making it a suitable alternative when the sample size is insufficient to establish normality.

As per the grand rank Table 4.7, we can see that CGN(A) is the most reliable and efficient method, so we test it against all the other methods.

Table 4.10: Calculated test statistic.

Sum	CGN(B)	CGQN(A)	CGQN(B)	GN(A)	GN(B)	GQN(A)	GQN(B)
Pos. Sum	125(-12)	55(-82)	62(-75)	284(147)	227(90)	107(-30)	127.5(-9.5)
Neg. Sum	334(197)	407(270)	400(263)	153(16)	228(91)	358(221)	337.5(200.5)

The Wilcoxon signed-rank test critical value is 137 (since the number of test problems for rank calculation in Table 4.5 is 30). In this context, the "critical value" refers to the threshold value used to evaluate the test statistic for the Wilcoxon signed-rank test. The number 137 comes from a table of critical values for the Wilcoxon signed-rank test, where the number of test problems (or pairs of observations) is given as 30. If the calculated test statistic (which is the sum of the signed ranks of the differences between pairs) exceeds the critical value, the null hypothesis is rejected, meaning that there is a significant difference between the two methods. If the test statistic is less than or equal to the critical value, we fail to reject the null hypothesis, suggesting no significant difference between the two methods.

As Table 4.10 shows, in the case of GN(A) and GN(B), the null hypothesis cannot be rejected, so there is no statistically significant reason for rejecting the fact that these two algorithms may produce a similar outcome in terms of the number of iterations, the number of function evaluations, or the ranks than CGN(A). However, for all other cases, we can reject the null hypothesis. Thus, we can say that these algorithms are statistically different than CGN(A).

4.7 Convergence

Convergence percentage Table 4.11 shows a summary of convergence for all algorithms based on Tables 4.3 and 4.5. Note that Table 4.5 also shows the cases where the method converges to the solution with some relaxation (ranks 9 and 10). Rank 9 is obtained when the convergence happens when we increase the number of iterations from 500 to 750, but keep the tolerance ϵ at 10^{-6}. However, if a method does not converge with this option, then we decrease the tolerance to 10^{-3}. If convergence happens in this case, then rank 10 is given. If the method fails to converge in both relaxations, then we give rank 11. GQN(B) returns solutions for almost all problems except one, although the number of iterations and the total number of function evaluations are higher than in other algorithms. CGN(A) and CGN(B) are the most reliable and efficient methods as they have mainly a lower number of iterations and a lower number of function evaluations than other methods, with more than 85% convergence and 100% relaxed convergence rates.

Table 4.11: Convergence percentages of the methods for the test problems.

Algorithm	Convergence	Relaxed Convergence	Not Converging
CGN(A)	86.7	100	0
CGN(B)	86.7	100	0
CGQN(A)	46.7	56.7	43.3
CGQN(B)	53.3	73.3	26.7
GN(A)	76.7	93.3	6.7
GN(B)	80	100	0
GQN(A)	76.7	86.7	13.3
GQN(B)	96.7	100	0

4.8 Graphical output

In this section, we present a few examples of the graphical output to demonstrate the outcome of the Algorithm 14. We found out that the CGN(A) method is the most efficient and reliable one on the test problems mentioned in Table 4.2. Let us analyze the graphical output of this method for the problem P2, which looks simple visually as we have only four functions.

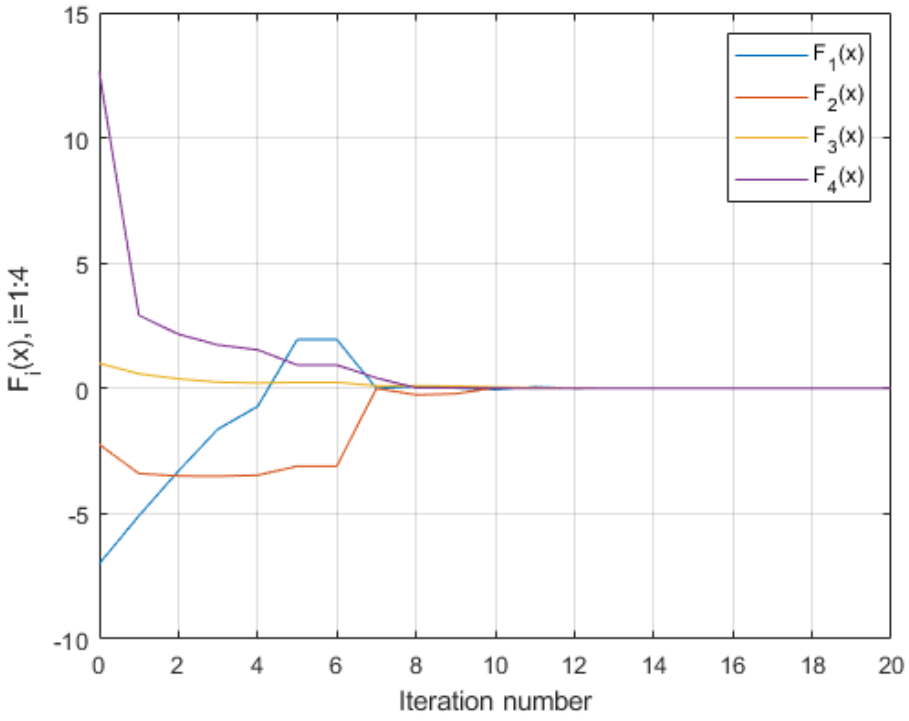

Figure 4.2: Function values vs. number of iterations for P2

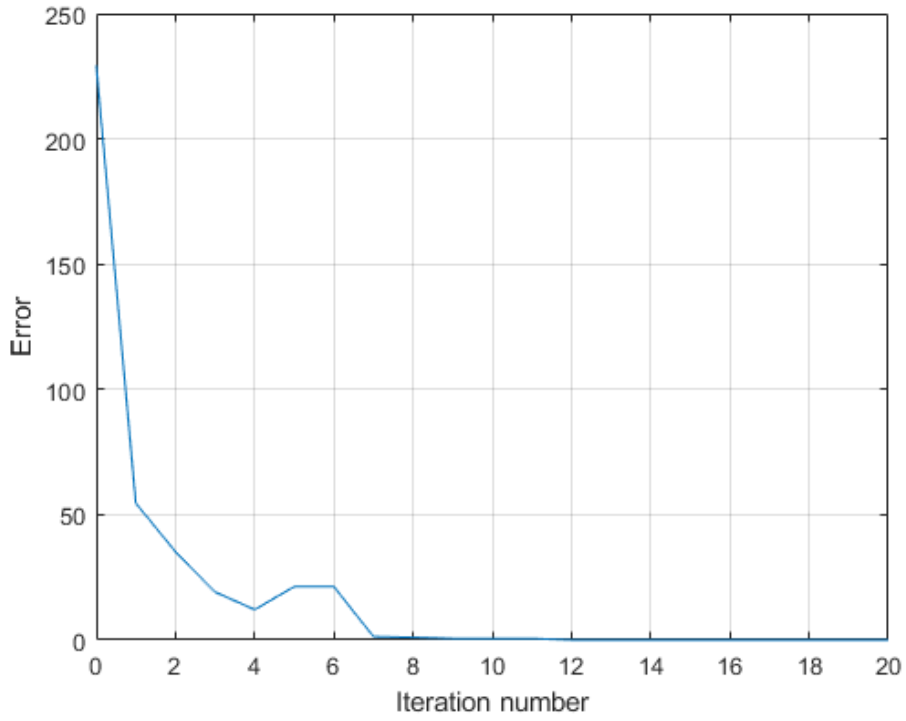

Figure 4.3: Error vs. number of iterations for P2

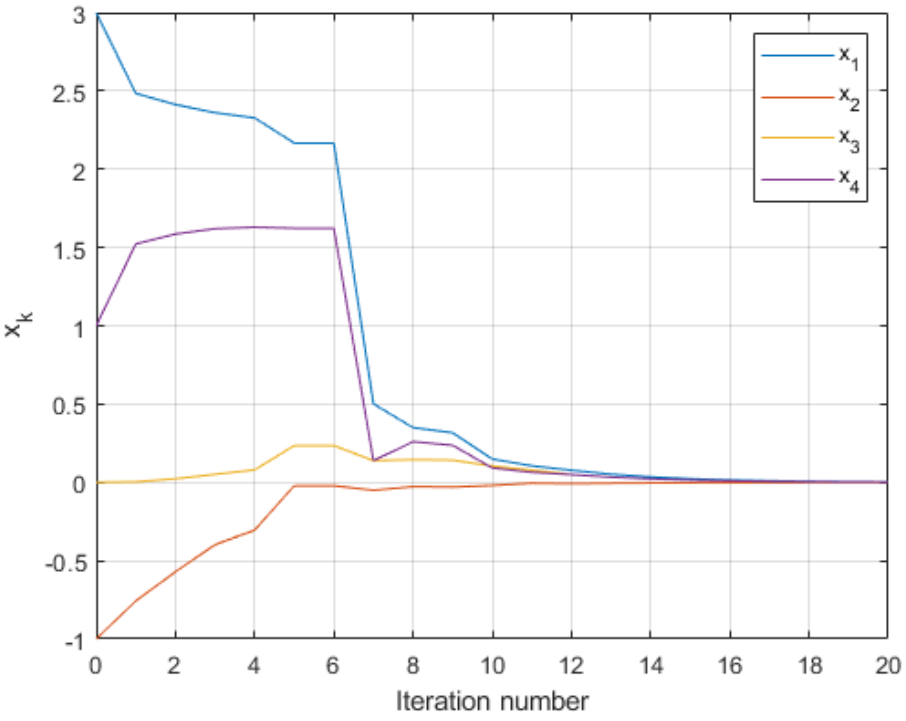

Figure 4.4: Root vs. number of iterations for P2

Figure 4.2 illustrates the progression of the function values across iterations. Some values converged to the root immediately, while others required a few iterations to begin converging, reflecting variations in the optimization dynamics. Figure 4.3 shows the error, measured as the norm of the gradient, steadily decreasing and converging to the user-defined tolerance of 10^{-6}, demonstrating the algorithm's accuracy. Figure 4.4 highlights the progression of the solution variables, visualizing their convergence toward the root. For problem P2, the global solution is achieved since all the function values go to zero, confirming the robustness and reliability of the CGN(A) algorithm in identifying the optimal solution.

Now, let us consider another problem P8 with the dimension $n = 50$. Results for this problem are presented in Figures 4.5-4.7. Similar to P2, a global solution for P8 is also achieved with all root values equal to zero. The results demonstrate the robustness and reliability of the CGN(A) algorithm in efficiently converging to the global solution.

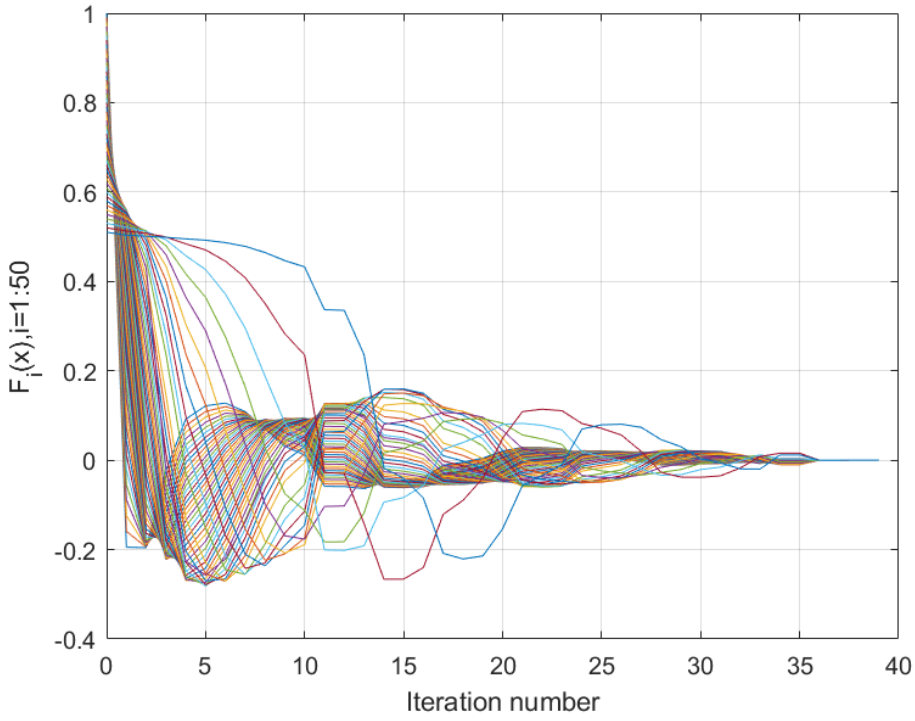

Figure 4.5: Function values vs. number of iterations for P8

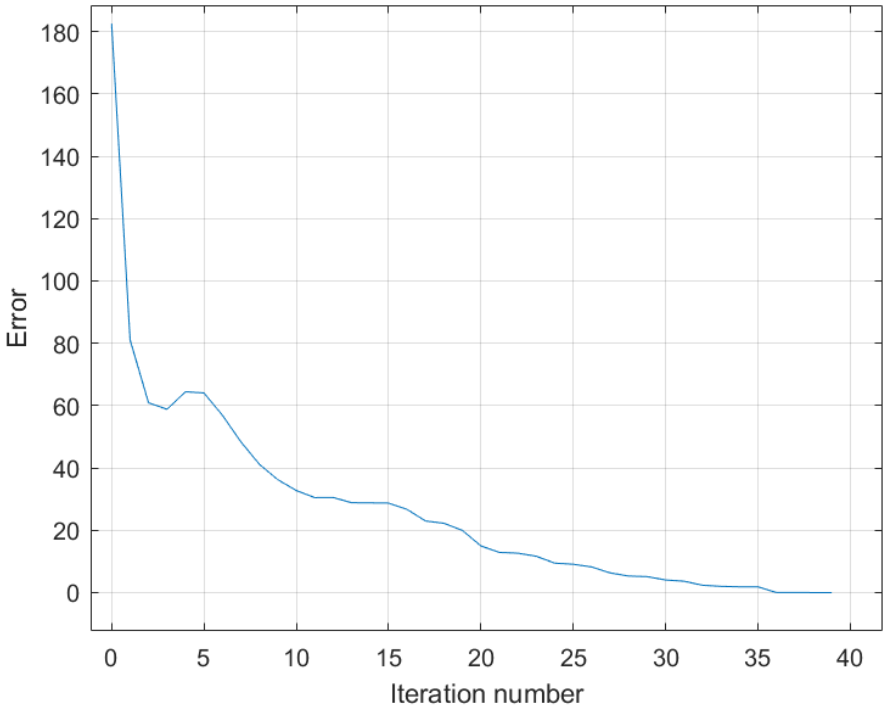

Figure 4.6: Error vs. number of iterations for P8

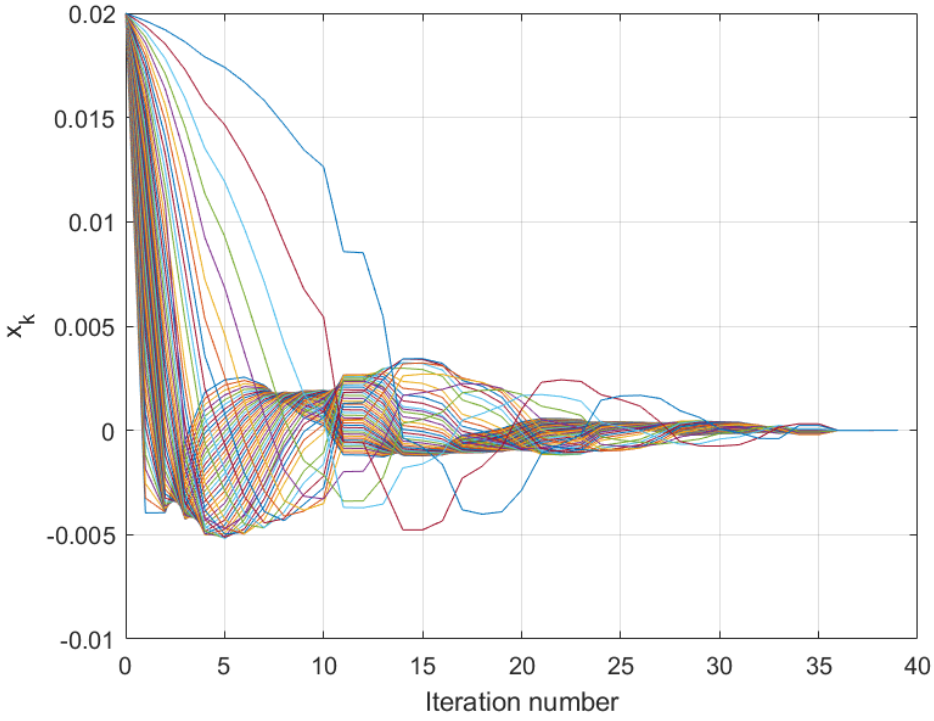

Figure 4.7: Root vs. number of iterations for P8

Chapter 5

Real Life Applications and further research

Optimization techniques are the core of data science and artificial intelligence. An explicit connection between optimization and machine learning was established in the 1960s on pattern separation using linear programming [50, 51]. Support vector machines and kernel learning were developed based on optimization (especially quadratic programming and duality) techniques during the 1990s [20, 57].

Optimization problems occur in many real-life applications, for example, in engineering, physical sciences, social sciences, and commerce [5, 41, 42]. They are particularly valuable in various engineering branches like mechanical, aeronautical, electrical, and chemical engineering [43, 55]. Many practical applications like robotics, pattern recognition, digital signal processing, and telecommunications can be solved by using optimization techniques [5, 41].

In this chapter, we describe two possible applications [5] where the presented hybrid methods, particularly Algorithm 14 from the previous chapter, could be of potential use and merit. These industrial applications, namely the inverse kinematics of robotic manipulators and the optimization of dissimilarity in hand-written text, are relevant to optimization theory [5]. As they fundamentally involve solving systems of nonlinear equations, these applications present a promising avenue for evaluating the practical effectiveness of the hybridization techniques considered in the thesis.

5.1 Engineering application

Typically, an industrial robot is composed of a chain of mechanical links with one end fixed relative to the ground and the other end, known as the end-effector, free to move. Manipulators allow motion by moving the joints of each link along its axis with an electric or hydraulic actuator. One of the main issues with robotics is the difficulty in describing the position and orientation of the end-effector relative to the joint variables. There are two types of joints: rotational joints, which allow the linked robot to rotate, and translational joints, which enable the linked robot to move along a straight line.

Inverse kinematics for robotic manipulators is essential for finding the joint vari-

ables that satisfy the desired configuration of the robotic arm during manipulation. The goal of the robot controller is to generate an acceptable motion of the end-effector by precisely actuating its joints for a specified task. In inverse kinematics, we need to find joint angles θ_i for all $i = 1, ..., n$ where n is the number of joints with which the manipulator's end-effector would achieve a prescribed position and orientation. The inverse kinematics of robotic manipulators entails a system of nonlinear equations that can be transformed into an unconstrained minimization problem and then solved with an optimization method.

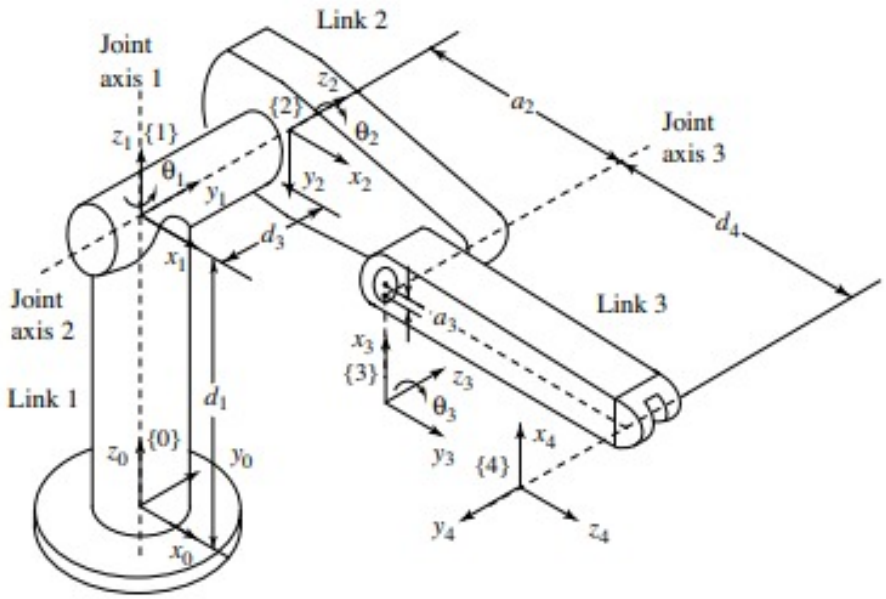

Figure 5.1: A three-link robotic manipulator [5].

Figure 5.1 depicts a 3-joint industrial robot manipulator so we have transformed vector coordinates $\boldsymbol{x} = [\theta_1, \theta_2, \theta_3]^T$ and a system of equations $\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{0}$ where each $F_i(\boldsymbol{x}) = 0$ for $i = 1, 2, 3$. For example, if we know the position of the end-effector $[P_x, P_y, P_z]^T$, then we can write the equations in the form

$$\begin{aligned} F_1(\boldsymbol{x}) &= c_1(a_2c_2 + a_3c_{23} - d_4s_{23}) - d_3s_1 - P_x \\ F_2(\boldsymbol{x}) &= s_1(a_2c_2 + a_3c_{23} - d_4s_{23}) - d_3c_1 - P_y \\ F_3(\boldsymbol{x}) &= d_1a_2s_2 - a_3s_{23} - d_2c_{23} - P_z \end{aligned} \tag{5.1}$$

where $c_i = \cos(\theta_i)$, $c_{23} = \cos(\theta_2 + \theta_3)$, $s_i = \sin(\theta_i)$, $s_{23} = \sin(\theta_2 + \theta_3)$, a_i is distance from the z_i axis to the x_{i+1} axis measured along the x_i axis and d_i is the distance from the x_{i-1} axis to the x_i axis measured along the z_i axis. The system of equations (5.1) can be expressed as the unconstrained optimization problem

$$\min_{\boldsymbol{x} \in \mathbb{R}^3} f(\boldsymbol{x}) = F_1^2(\boldsymbol{x}) + F_2^2(\boldsymbol{x}) + F_3^2(\boldsymbol{x}) \text{ where } \boldsymbol{x} = [\theta_1, \theta_2, \theta_3]^T$$

which can be solved by Algorithm 14.

Once we know θ_i for $i = 1, 2, 3$, the orientation of the end-effector becomes known. The main advantage of solving inverse kinematics as an unconstrained optimization problem is that when the desired position is not in the range of the manipulator, the optimization of f still yields an approximated solution $\boldsymbol{x}^*$. Although the objective function value $f(\boldsymbol{x}^*)$ may still not be zero, such an approximated solution can still be considered satisfactory for most engineering applications.

5.2 Machine learning application

In a digital image, edges characterize object boundaries, so edge detection remains a crucial stage in numerous applications, principally in pattern recognition [33], computational vision, image registration, computational chemistry, molecular biology, and medical data [17]. Point pattern matching is extensively used because boundaries include the most important part of the structure of the image, and edge detection could be used to qualify a region segmentation technique. An efficient boundary detection method should produce a contour image with edges accurately located and minimal pixel misclassification [5]. Minimal pixel misclassification means correctly classifying pixels as either belonging to an edge (boundary) or not, with as few errors as possible.

For example, in a point pattern matching problem, a pattern such as a printed or handwritten character, numeral, symbol, or even outline of a manufactured part can be described by a set of points

$$\mathcal{P} = \{\boldsymbol{P}_1, \boldsymbol{P}_2, \ldots, \boldsymbol{P}_n\} \tag{5.2}$$

where

$$\boldsymbol{P}_i = \begin{bmatrix} p_{i1} \\ p_{i2} \end{bmatrix}$$

is a vector in terms of the coordinates of the i^{th} sample point. If the number of points n in $\mathcal{P}$, is sufficiently large, then $\mathcal{P}$ in (5.2) describes the object accurately and $\mathcal{P}$ is referred to as a point pattern of the object.

Let us consider a database containing N standard point patterns $\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_N\}$ where each $\mathcal{P}_j$ is of the form (5.2) and we need to discover the database pattern that mostly resembles a particular point pattern

$$Q = \{\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_n\}.$$

where

$$\boldsymbol{q}_i = \begin{bmatrix} q_{i1} \\ q_{i2} \end{bmatrix}.$$

In real-life situations, the perfect match between some $\mathcal{P}$ and Q is not possible, so we use $\tilde{\mathcal{P}}$ a transformed version of pattern $\mathcal{P}$. So if the pattern $\mathcal{P}$ is given by (5.2) then

$$\tilde{\mathcal{P}}(\boldsymbol{x}) = \{\tilde{\boldsymbol{P}}_1, \tilde{\boldsymbol{P}}_2, \ldots, \tilde{\boldsymbol{P}}_n\}$$

where

$$\tilde{\boldsymbol{P}}_i = \eta \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \boldsymbol{P}_i + \begin{bmatrix} r_1 \\ r_2 \end{bmatrix},$$

θ is a rotational angle, η is a scaling parameter, $\boldsymbol{r} = [r_1, r_2]^T$ is a translation vector for $i = 1, 2, \ldots, n$, and $\boldsymbol{x} = [\eta\cos(\theta), \eta\sin(\theta), r_1, r_2]^T$. A transformed pattern can be obtained by solving an unconstrained optimization problem

$$\min_{\boldsymbol{x} \in \mathbb{R}^4_+} \quad f(\boldsymbol{x}) = \left|\!\left|\!\left| \tilde{P}(\boldsymbol{x}) - Q \right|\!\right|\!\right|^2, \tag{5.3}$$

where $|||\cdot|||$ means the matrix norm. The optimization problem (5.3) can be solved by Algorithm 14. Lets assume that $\boldsymbol{x}^*$ is the minimizer for the problem (5.3) then the error

$$e(\tilde{P}(\boldsymbol{x}^*), Q) = \sqrt{f(\boldsymbol{x}^*)} = \left|\!\left|\!\left| \tilde{P}(\boldsymbol{x}^*) - Q \right|\!\right|\!\right|$$

is a measure of the dissimilarity between patterns $\tilde{P}(\boldsymbol{x}^*)$ and Q. Obviously, $e(\tilde{P}(\boldsymbol{x}^*), Q)$ should be close to zero for a perfect match. So, the pattern encoded by vector $\boldsymbol{x}^*$ is selected.

5.3 Future research

Up to this point, the CGN algorithm 14 has been evaluated using academic test problems. The development of the hybrid algorithm opens several promising directions for future research aimed at strengthening both its theoretical underpinnings and practical effectiveness. One important avenue involves conducting a rigorous convergence analysis to determine the conditions under which the method achieves global convergence, and to characterize its convergence rate—whether it maintains Newton's superlinear convergence or exhibits the linear rate typical of CG in certain scenarios. In future work, we will focus on applying the CGN algorithm 14 to the problems discussed above and on establishing a formal convergence theorem for the method.

Chapter 6

Summary

Solving systems of nonlinear equations poses significant computational challenges due to their inherent nonlinearity. In recent years, constructive theories and advanced algorithms have been developed to enhance the accuracy and efficiency of solving such systems. Among those, Newton's method remains a classical approach but is notably sensitive to the choice of the initial starting point. To mitigate this limitation, hybrid approaches have been introduced, integrating multiple methods to achieve improved convergence rates.

This thesis investigates hybrid methods for approximating solutions to systems of nonlinear equations. The work begins with a comprehensive review of classical approaches for solving systems of nonlinear equations, followed by an overview of various techniques employed in nonlinear optimization. Optimization methods are considered since the systems of nonlinear equations can be reformulated as optimization problems. The focus then shifts to the development of novel hybrid algorithms that utilize Newton's method, the conjugate gradient method, the quasi-Newton method, and the gradient method.

A key contribution of this work is the empirical evaluation of computational efficiency for the new hybrid approach combining the conjugate gradient and Newton's methods. To assess the effectiveness of the proposed hybridization, a series of numerical experiments has been conducted, complemented by a ranking system to compare the performance of all presented new hybrids. The results demonstrate that the hybrid approach combining the conjugate gradient and Newton's methods is the most reliable and efficient. It requires fewer iterations, reduces the total number of function evaluations, achieves a convergence rate of nearly ninety percent, and attains a one-hundred-percent relaxed convergence rate.

While the findings are preliminary and the computational experiments have certain limitations, this study provides valuable insights. It identifies the most promising hybridization techniques for further research and potential advancements in solving nonlinear systems.

Bibliography

[1] F. Aragón, M. Goberna, M. López, M. Rodríguez: *Nonlinear Optimization*, Springer, 2019.

[2] M. Altman: *Iterative methods of contractor directions*, Nonlinear Analysis: Theory, Methods & Applications, 1980, vol. 4, pp. 761-771.

[3] N. Andrei: *An unconstrained optimization test functions collection*, Advanced Modeling and Optimization, 2008, vol. 10, pp. 147-161.

[4] H. An, Z. Mo, X. Liu: *A choice of forcing terms in inexact Newton method*, Journal of Computational and Applied Mathematics, 2007, vol. 200, pp. 47-60.

[5] A. Antoniou, W. Lu: *Practical Optimization: Applications of Unconstrained Optimization*, Springer, 2007.

[6] K. Atkinson: *An Introduction to Numerical Analysis. Nonlinear Systems of Equations*, John Wiley & Sons, 1978.

[7] A. Bagirov, N. Karmitsa, M. Mäkelä: *Introduction to Nonsmooth Optimization: Theory, Practice and Software*, Springer, 2014.

[8] R. Bartle, D. Sherbert: *Introduction to Real Analysis*, John Wiley & Sons, 2000.

[9] M. Bazaraa, H. Sherali, C. Shetty: *Nonlinear Programming: Theory and Algorithms*, Wiley-Interscience, 2006.

[10] S. Boyd, L. Vandenberghe: *Convex Optimization*, Cambridge University Press, 2004.

[11] J. Brinkhuis: *Convex Analysis for Optimization (Graduate Texts in Operations Research)*. Springer, 1st Edition, 2020.

[12] A. Buckley: *A combined conjugate-gradient quasi-Newton minimization algorithm*, Mathematical Programming, 1978, vol. 15, pp. 200-210.

[13] C. Broyden: *A class of methods for solving nonlinear simultaneous equations*, American Mathematical Society, Mathematics of Computation, 1965, vol. 19, pp. 577-593.

[14] C. Broyden: *Quasi-Newton methods and their application to function minimization*, Mathematics of Computation, 1967, vol. 21, pp. 368-381.

[15] R. Burden, J. Faires: *Numerical Analysis*, Brooks Cole, 9th Edition, 2010.

[16] R. Byrd, J. Nocedal, Y. Yuan: *Global convergence of a class of quasi-Newton methods on convex problems*, SIAM Journal on Numerical Analysis, 1987, vol. 24, pp. 1171-1190.

[17] J. Canny: *A Computational Approach to Edge Detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 1986, vol. 8, pp. 679-698.

[18] A. Cauchy: *Lecons sur le calcul differentiel*, chez De Bure freres, libraires du roi et de la bibliotheque du roi, rue Serpente, n. 7, 1829.

[19] A. Cauchy: *Méthode générale pour la résolution des systémes d'équations simultanées*, Comptes rendus de l'Académie des Sciences, 1847, vol. 25, pp. 536-538.

[20] C. Cortes, V. Vapnik: *Support-vector networks*, Machine Learning, 1995, vol. 20, pp. 273-297.

[21] W. Davidon: *Variable metric method for minimization*, AEC Research and Development Report ANL-5990, 1959.

[22] B. Datta: *Numerical Linear Algebra and Applications*, Society for Industrial and Applied Mathematics, 2010.

[23] R. Dembo, S. Eisenstat, T. Steihaug: *Inexact Newton methods*, SIAM Journal on Numerical Analysis, 1982, vol. 19, pp. 400-408.

[24] J. Dennis: *On the Convergence of Broyden's Method for Nonlinear Systems of Equations*, Mathematics of Computation, 1971, vol. 25, pp. 559-567.

[25] J. Dennis, R. Schnabel: *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Society for Industrial & Applied Mathematics, U.S., 1987.

[26] H. Fine: *On Newton's method of approximation*, Proceedings of National Academy of Sciences USA, 1916, vol. 2, pp. 546-552.

[27] R. Freund: *The steepest descent algorithm for unconstrained optimization and a bisection line-search method*. Available at `https://ocw.mit.edu/courses/15-084j-nonlinear-programming-spring-2004/9c07ac34609f36bb15967a35a4e77a7b_lec5_steep_desce.pdf` (retrieved on 11/07/2024).

[28] R. Fletcher: *A new approach to variable metric algorithms*, Computer Journal, 1970, vol. 13, pp. 317-322.

[29] R. Fletcher, M. Powell: *A rapidly convergent descent method for minimization*, Computer Journal, 1963, vol. 6, pp. 163-168.

[30] R. Fletcher, C. Reeves: *Function minimization by conjugate gradients*, Computer Journal, 1964, vol. 7, pp. 149-154.

[31] J. Fourier: *Question d'analyse algebrique*, In Oeuvres completes, Gauthier-Villars, 1890, vol. 2, pp. 243-253.

[32] D. Goldfarb: *A Family of variable metric updates derived by variational means*, Mathematics of Computation, 1970, vol. 24, pp. 23-26.

[33] R. Gonzalez, R. Woods: *Digital Image Processing*, Pearson, 2017.

[34] J. Gilbert, J. Nocedal: *Global convergence properties of conjugate gradient methods for optimization*, SIAM Journal on Optimization, 1992, vol. 2, pp. 21-42.

[35] F. Hanson: *MCS 471 Class Notes: Secant method error and convergence rate*. Available at `https://homepages.math.uic.edu/~jan/mcs471f05/Lec5/secant.pdf` (retrieved on 15/05/2024).

[36] M. Hestenes, E. Stiefel: *Methods of conjugate gradients for solving linear systems*, Journal of Research of the National Bureau of Standards, 1952, vol. 49, pp. 409-436.

[37] R. Hooke, T. Jeeves: *Direct search solution of numerical and statistical problems*, Journal of the ACM, 1961, vol. 8, pp. 212-229.

[38] R. Horn, C. Johnson: *Matrix Analysis*. Cambridge University Press, 2nd Edition, 2012.

[39] L. Kantorovich: *On Newton's method for functional equations*, Proceedings of Academy of Sciences USSR, 1948, vol. 59, pp. 1237-1240.

[40] J. Kiefer: *Sequential minimax search for a maximum*, Proceedings of the American Mathematical Society, 1953, vol. 4, pp. 502-506.

[41] E. Lee, C. Chen: *Modeling and Optimizing the Public-Health Infrastructure for Emergency Response*, Interfaces, 2009, vol. 39, pp. 476-490.

[42] S. Lee, S. Shin: *Wind turbine blade optimal design considering multi-parameters and response surface method*, Energies, 2020, vol. 13, pp. 16-39.

[43] Z. Lyu, J. Martins: *Performance optimization of gas turbine engine*, Journal of Aircraft, 2014, vol. 51, pp. 1604-1617.

[44] Y. Nesterov: *Introductory Lectures on Convex Optimization: A Basic Course.* Kluwer Academic Publishers, 2004.

[45] J. Nocedal, S. Wright: *Numerical Optimization*, Springer, 2006.

[46] A. Rodomanov, Y. Nesterov: *New results on superlinear convergence of classical Quasi-Newton methods.* Journal of Optimization Theory and Applications, 2021, vol. 188 pp. 744-769.

[47] M. Powell: *A hybrid method for nonlinear equations*, Numerical methods for nonlinear algebraic equations, 1970, ch. 6, pp. 87-114.

[48] W. Press, S. Teukolsky, W. Vetterling, B. Flannery: *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, 3rd Edition, 2007.

[49] J. Raphson: *Analysis aequationum universalis seu ad aequationses algebraicas resolvendas methodus generalis, et expedita, ex nova infinitarum serierum doctrina deducta ac demontrata*, Microfilm copy, Ann Arbor, MI, 1690.

[50] F. Rosenblatt: *The Perceptron: A probabilistic model for information storage and organization in the brain*, Psychological Review, 1958, vol. 65, pp. 386-408.

[51] A. Samuel: *Some studies in machine learning using the game of checkers*, IBM Journal of Research and Development, 1959, vol. 3, pp. 210-229.

[52] D. Shanno: *Conditioning of quasi-Newton methods for function minimization*, Mathematics of Computation, 1970, vol. 24, pp. 647-656.

[53] A. Shapiro: *On concepts of directional differentiability*, Journal of Optimization Theory and Applications, 1990, vol. 66, pp. 477-487.

[54] Y. Shi: *Globally convergent algorithms for unconstrained optimization*, Computational Optimization and Applications, 2000, vol. 16, pp. 295-308.

[55] V. Silva, W. Khatib, P. Fleming: *Performance optimization of gas turbine engine*, Engineering Applications of Artificial Intelligence, 2005, vol. 18, pp. 575-583.

[56] S. Taheri, M. Mammadov: *Solving systems of nonlinear equations using a globally Convergent Optimization algorithm*, Global Journal of Technology & Optimization, 2012, vol. 3, pp. 132-138.

[57] V. Vapnik: *Statistical Learning Theory*, John Wiley & Sons, 1998.

[58] F. Wilcoxon: *Individual Comparisons by Ranking Methods*, Biometrics Bulletin, 1945, vol. 1, pp. 80-83.

Appendices

Appendix A

CGN Algorithm Matlab code

```
clc; clear; close all

% define the function
f1=@(x) x(1)+10*x(2);
f2=@(x) sqrt(5)*(x(3)-x(4));
f3=@(x) (x(2)-2*x(3))^2;
f4=@(x) sqrt(10)*(x(1)-x(4))^2;
f=@(x)[f1(x);f2(x);f3(x);f4(x)];
F=@(x) 1/2*norm([f1(x),f2(x),f3(x),f4(x)])^2;

% parameters
tol=1e-6; % tolerance
maxIter=500; % max iteration number

delta0   = 0.001;
Lambda0  = 1;
eta      = 0.99;
rho      = 0.001;
sigma    = 0.9;
b1       = 0.01;
b2       = (1/b1);
b3       = 1.1;
tau      = 10e-10;
T        = 10e10;

gamma1   = 4;
gamma2   = 4;
dx       = 1e-5;
%-------------------------------------------------------
%-------------------------------------------------------
% initial guess
x0 = [3,-1,0,1]' ;
tic
f0=0;
Fit=F(x0);
xold=2*x0;
xk=x0;
errors=[];
Xs = [xk'];
Xit=xk';
Fs =f(xk);

falpha_count=0;
% Step1
for k = 0:maxIter
H = hessian(F,xk,dx);
gk = grad(F,xk,dx);
error = norm(gk);
errors =[errors, error];
% Step1
if error<tol
```

```
    break;
end
% Step 2:
if det(H)~=0 % Nonsingular matrix
    % Step 3
    d1=-H\gk; % Newton's direction
    if k>0
        beta = norm(gk)^2/norm(grad(F,xold,dx))^2;
        d2=-gk+beta*d2;       % gradient direction
    else
        d2=-gk;
    end

    % Step 4:
    delta=delta0; Lambda=Lambda0;
    if abs(F(xk)-F(xold))>gamma1 && norm(gk)>gamma2
        delta=b2*delta0;
    elseif k==1 || norm(gk)<=norm(grad(F,xold,dx))
        % Step 5
        x_bar=xk+d1;
        % Step6:
        if F(x_bar)<F(xk) && norm(grad(F,x_bar,dx))<=eta*norm(gk)
            delta=b1*delta0;
        end
    end

    % Step 7:
    if d1'*d2>=0
        while 1
            % Step 9
            xi=1/(Lambda+abs(F(xk)-F(xold)));
            dxi=(1-xi)*d2+xi*d1;
            % Step 10
            if dxi'*d2<delta*norm(dxi)*norm(d2)
                Lambda=b3*Lambda;
            else
                break;
            end

        end
        %------------------------ Step 11(a)---------------------:
%         dk=d2;
%         [alpha, count]=calc_alpha(F,@(x)grad(F,x,dx), xk, dk, rho);
%         falpha_count=falpha_count+count;
%         s_bar=alpha*(1-xi)*d2+xi*d1;
%         if F(xk+s_bar)<=F(xk)-tau*norm(s_bar)&& alpha*norm(d2)<=T*norm(d1)
%             sk=s_bar;
%         else
%             sk=alpha*dk;
%         end

        %----------------------Step 11(b)--------------

        % Step 11(b):
        dk=dxi;
        [alpha, count]=calc_alpha(F,@(x)grad(F,x,dx), xk, dk, rho);
        falpha_count=falpha_count+count;
        sk=alpha*dk;
        %-----------------------------------------------
        % Step 12
        x_new=xk+sk;

    else
        % Step 8:

        dk=d2;
        [alpha, count]=calc_alpha(F,@(x)grad(F,x,dx), xk, dk, rho);
        falpha_count=falpha_count+count;
        sk=alpha*dk;
        % Step 12
```

```
        x_new=xk+sk;
    end


else % if H is singular, goto step 8
    % Step 8:
    if k>0
        beta = norm(gk)^2/norm(grad(F,xold,dx))^2;
        d2=-gk+beta*d2;       % gradient direction
    else
        d2=-gk;
    end
    dk=d2;
    [alpha, count]=calc_alpha(F,@(x)grad(F,x,dx), xk, dk, rho);
    falpha_count=falpha_count+count;
    %Step12
    sk=alpha*dk;
    x_new=xk+sk;
end
xold=xk;
xk = x_new;
Xit =[Xit; xk'];
Fs =[Fs,f(xk)];
end
t=toc;
min_f = F(xk);
total_f_eval=falpha_count+k;

%% PLOT
fprintf('The optimized value of F(x): %f\n', min_f);
fprintf('Elasped time is: %f seconds\n', t)
fprintf('Total iteration number is: %i\n', k)
fprintf('Total function evaluation at Alpha is: %i\n', falpha_count)
fprintf('Total function evaluation is: %i\n', total_f_eval)

iter=0:k;
% plot function values vs iteration
plot(iter, Fs);
title('Function values vs iteration')
xlabel('#iteration')
ylabel('f(x)'); grid on
legend('f_1(x)','f_2(x)','f_3(x)','f_4(x)')

% plot the error
figure, plot(0:k, errors)
title('Error vs iteration')
xlabel('#iteration')
ylabel('Error'); grid on


% plot the xk
figure, plot(0:k, Xit)
title('Optimized roots vs iteration')
xlabel('#iteration')
ylabel('x_k'); grid on
legend('x_1','x_2','x_3','x_4')
```

Appendix B

CGQN Algorithm Matlab code

```
clc; clear; close all

% define the function
theta=@(x1,x2)1/2/pi*atan(x2/x1)+0.5*heaviside(-x1);

f1 = @(x) 10*(x(3)-10*theta(x(1),x(2)));
f2 = @(x) 10*((x(1)^2+x(2)^2)^0.5-1);
f3 = @(x) x(3);
f=@(x)[f1(x);f2(x);f3(x)];
F = @(x)1/2*norm([f1(x);f2(x);f3(x)])^2;

% parameters
tol=1e-6; % tolerance
maxIter=500; % max iteration number

delta0   = 0.001;
Lambda0  = 1;
eta      = 0.99;
rho      = 0.001;
sigma    = 0.9;
b1       = 0.01;
b2       = (1/b1);
b3       = 1.1;
tau      = 10e-10;
T        = 10e10;

gamma1   = 3;
gamma2   = 3;
dx       = 1e-5;
%-------------------------------------------------------------
% initial guess
tic
x0 = [-1,0,0]' ;
f0=0;
xold=2*x0;
xk=x0;
errors=[];
Xit = [xk'];
Fs =[f1(xk), f2(xk), f3(xk)];
N  = size(x0,1);
B=eye(N);
falpha_count=0;
% Step1
for k = 0:maxIter
    %H = hessian(F,xk,dx);
    gk = grad(F,xk,dx);
    error = norm(gk);
    errors =[errors, error];
    % Step1
    if error<tol
        break;
```

```
    end
    % Step 2:

    % Step 3
    d1=B\(-grad(F,xk,dx)); % Newton's direction
    if k>0
        beta = norm(gk)^2/norm(grad(F,xold,dx))^2;
        d2=-gk+beta*d2;       % gradient direction
    else
        d2=-gk;
    end
    % Step 4:
    delta=delta0; Lambda=Lambda0;
    if abs(F(xk)-F(xold))>gamma1 && norm(gk)>gamma2
        delta=b2*delta0;
    elseif k==1 || norm(gk)<=norm(grad(F,xold,dx))
        % Step 5
        x_bar=xk+d1;
        % Step6:
        if F(x_bar)<F(xk) && norm(grad(F,x_bar,dx))<=eta*norm(gk)
            delta=b1*delta0;
        end
    end

    % Step 7:
    if d1'*d2>=0
        while 1
            % Step 9
            xi=1/(Lambda+abs(F(xk)-F(xold)));
            dxi=(1-xi)*d2+xi*d1;
            % Step 10
            if dxi'*d2<delta*norm(dxi)*norm(d2)
                Lambda=b3*Lambda;
            else
                break;
            end

        end
        %------------------------- Step 11(a)---------------------:
%         dk=d2;
%         [alpha, count]=calc_alpha(F,@(x)grad(F,x,dx), xk, dk, rho);
%         falpha_count=falpha_count+count;
%         s_bar=alpha*(1-xi)*d2+xi*d1;
%         if F(xk+s_bar)<=F(xk)-tau*norm(s_bar)&& alpha*norm(d2)<=T*norm(d1)
%             sk=s_bar;
%         else
%             sk=alpha*dk;
%         end

        %----------------------Step 11(b)--------------

        % Step 11(b):
        dk=dxi;
        [alpha, count]=calc_alpha(F,@(x)grad(F,x,dx), xk, dk, rho);
        falpha_count=falpha_count+count;
        sk=alpha*dk;
        %-----------------------------------------------
        % Step 12
        x_new=xk+sk;

    else
        % Step 8:
        dk=d2;
        [alpha, count]=calc_alpha(F,@(x)grad(F,x,dx), xk, dk, rho);
        falpha_count=falpha_count+count;
        sk=alpha*dk;
        % Step 12
        x_new=xk+sk;
    end
    yk=grad(F,x_new, dx)-grad(F,xk,dx);
```

```
    B=B+yk*yk'/(yk'*sk)-B*sk*sk'*B'/(sk'*B*sk);
    xold=xk;
    xk = x_new;
    Xit =[Xit; xk'];
    Fs =[Fs;f1(xk), f2(xk), f3(xk)];
end
t=toc;
min_f = F(xk);
total_f_eval=falpha_count+k;

%% PLOT
fprintf('The optimized value of F(x): %f\n', min_f);
fprintf('Elasped time is: %f seconds\n', t)
fprintf('Total iteration number is: %i\n', k)
fprintf('Total function evaluation at Alpha is: %i\n', falpha_count)
fprintf('Total function evaluation is: %i\n', total_f_eval)
```

Appendix C

GQN Algorithm Matlab code

```
clc; clear; close all

% define the function
theta=@(x1,x2)1/2/pi*atan(x2/x1)+0.5*heaviside(-x1);

f1 = @(x) 10*(x(3)-10*theta(x(1),x(2)));
f2 = @(x) 10*((x(1)^2+x(2)^2)^0.5-1);
f3 = @(x) x(3);
f=@(x)[f1(x);f2(x);f3(x)];
F = @(x)1/2*norm([f1(x);f2(x);f3(x)])^2;

% parameters
tol=1e-6; % tolerance
maxIter=1000; % max iteration number

delta0   = 0.001;
Lambda0  = 1;
eta      = 0.99;
rho      = 0.001;
sigma    = 0.9;
b1       = 0.01;
b2       = (1/b1);
b3       = 1.1;
tau      = 10e-10;
T        = 10e10;

gamma1   = 3;
gamma2   = 3;
dx       = 1e-5;
%-------------------------------------------------------------
%-------------------------------------------------------------
% initial guess
tic
x0 = [-1,0,0]' ;
f0=0;
xold=2*x0;
xk=x0;
errors=[];
Xit = [xk'];
Fs =[f1(xk), f2(xk), f3(xk)];
N  = size(x0,1);
B=eye(N);
falpha_count=0;
% Step1
for k = 0:maxIter
    H = hessian(F,xk,dx);
    gk = grad(F,xk,dx);
    error = norm(gk);
    errors =[errors, error];
    % Step1
    if error<tol
```

```
        break;
    end
    % Step 2:

    % Step 3
    d2=-gk; % gradient direction
    d1=B\(-grad(F,xk,dx)); % Newton's direction

    % Step 4:
    delta=delta0; Lambda=Lambda0;
    if abs(F(xk)-F(xold))>gamma1 && norm(gk)>gamma2
        delta=b2*delta0;
    elseif k==1 || norm(gk)<=norm(grad(F,xold,dx))
        % Step 5
        x_bar=xk+d1;
        % Step6:
        if F(x_bar)<F(xk) && norm(grad(F,x_bar,dx))<=eta*norm(gk)
            delta=b1*delta0;
        end
    end

    % Step 7:
    if d1'*d2>=0
        while 1
            % Step 9
            xi=1/(Lambda+abs(F(xk)-F(xold)));
            dxi=(1-xi)*d2+xi*d1;
            % Step 10
            if dxi'*d2<delta*norm(dxi)*norm(d2)
                Lambda=b3*Lambda;
            else
                break;
            end

        end
        %------------------------- Step 11(a)---------------------:
        dk=d2;
        [alpha, count]=calc_alpha(F,@(x)grad(F,x,dx), xk, dk, rho);
        falpha_count=falpha_count+count;
        s_bar=alpha*(1-xi)*d2+xi*d1;
        if F(xk+s_bar)<=F(xk)-tau*norm(s_bar)&& alpha*norm(d2)<=T*norm(d1)
            sk=s_bar;
        else
            sk=alpha*dk;
        end

        %----------------------Step 11(b)---------------

        % Step 11(b):
        %dk=dxi;
        %[alpha, count]=calc_alpha(F,@(x)grad(F,x,dx), xk, dk, rho);
        %falpha_count=falpha_count+count;
        %sk=alpha*dk;
        %-------------------------------------------------

        % Step 12
        x_new=xk+sk;

    else
        % Step 8:
        dk=d2;
        [alpha, count]=calc_alpha(F,@(x)grad(F,x,dx), xk, dk, rho);
        falpha_count=falpha_count+count;
        sk=alpha*dk;
        % Step 12
        x_new=xk+sk;
    end
    yk=grad(F,x_new, dx)-grad(F,xk,dx);
    B=B+yk*yk'/(yk'*sk)-B*sk*sk'*B'/(sk'*B*sk);
    xold=xk;
```

```
    xk = x_new;
    Xit =[Xit; xk'];
    Fs =[Fs;f1(xk), f2(xk), f3(xk)];
end
t=toc;
min_f = F(xk);
total_f_eval=falpha_count+k;

%% PLOT
fprintf('The optimized value of F(x): %f\n', min_f);
fprintf('Elasped time is: %f seconds\n', t)
fprintf('Total iteration number is: %i\n', k)
fprintf('Total function evaluation at Alpha is: %i\n', falpha_count)
fprintf('Total function evaluation is: %i\n', total_f_eval)
```

Appendix D

GN Algorithm Matlab code

```
clc; clear; close all

% define the function
theta=@(x1,x2)1/2/pi*atan(x2/x1)+0.5*heaviside(-x1);
f1 = @(x) 10*(x(3)-10*theta(x(1),x(2)));
f2 = @(x) 10*((x(1)^2+x(2)^2)^0.5-1);
f3 = @(x) x(3);
f=@(x)[f1(x);f2(x);f3(x)];
F = @(x)1/2*norm([f1(x);f2(x);f3(x)])^2;

% parameters
tol=1e-6; % tolerance
maxIter=500; % max iteration number

delta0   = 0.001;
Lambda0  = 1;
eta      = 0.99;
rho      = 0.001;
sigma    = 0.9;
b1       = 0.001;
b2       = (1/b1);
b3       = 1.1;
tau      = 10e-10;
T        = 10e10;

gamma1   = 3;
gamma2   = 3;
dx       = 1e-5;
%-------------------------------------------------------------
% initial guess
tic
x0 = [-1,0,0]' ;
f0=0;
xold=2*x0;
xk=x0;
errors=[];
Alpha=[];
Xit = [xk'];
Fs =[f1(xk), f2(xk), f3(xk)];
falpha_count=0;

% Step1
for k = 0:maxIter
H = hessian(F,xk,dx);
gk = grad(F,xk,dx);
error = norm(gk);
errors =[errors, error];
% Step1
if error<tol
    break;
end
```

```
% Step 2:
if det(H)~=0 % Nonsingular matrix
    % Step 3
    d2=-gk; % gradient direction
    d1=-H\gk; % Newton's direction
    % Step 4:
    delta=delta0;
    Lambda=Lambda0;
    if abs(F(xk)-F(xold))>gamma1 && norm(gk)>gamma2
        delta=b2*delta0;
    elseif k==1 || norm(gk)<=norm(grad(F,xold,dx))
        % Step 5
        x_bar=xk+d1;
        % Step6:
        if F(x_bar)<F(xk) && norm(grad(F,x_bar, dx))<=norm(eta*gk)
            delta=b1*delta0;
        end
    end
    % Step 7:
    if d1'*d2>=0
        while 1
            % Step 9
            xi=1/(Lambda+abs(F(xk)-F(xold)));
            dxi=(1-xi)*d2+xi*d1;
            % Step 10
            if dxi'*d2<delta*norm(dxi)*norm(d2)
                Lambda=b3*Lambda;
            else
                break;
            end

        end
        %------------------------- Step 11(a)---------------------:
%         dk=d2;
%         [alpha, count]=calc_alpha(F,@(x)grad(F,x,dx), xk, dk, rho);
%         falpha_count=falpha_count+count;
%         s_bar=alpha*(1-xi)*d2+xi*d1;
%         if F(xk+s_bar)<=F(xk)-tau*norm(s_bar)&& alpha*norm(d2)<=T*norm(d1)
%             sk=s_bar;
%         else
%             sk=alpha*dk;
%         end
        %----------------------Step 11(b)--------------

        % Step 11(b):
        dk=dxi;
        [alpha, count]=calc_alpha(F,@(x)grad(F,x,dx), xk, dk, rho);
        falpha_count=falpha_count+count;
        sk=alpha*dk;
        %-----------------------------------------------
        % Step 12
        x_new=xk+sk;

    else
        % Step 8:
        dk=d2;
        [alpha, count]=calc_alpha(F,@(x)grad(F,x,dx), xk, dk, rho);
        falpha_count=falpha_count+count;
        sk=alpha*dk;
        % Step 12
        x_new=xk+sk;
    end
else % if H is singular, goto step 8
    % Step 8:
    dk=-gk;
    [alpha, count]=calc_alpha(F,@(x)grad(F,x,dx), xk, dk, rho, sigma);
    falpha_count=falpha_count+count;
    %Step12
    sk=alpha*dk;
    x_new=xk+sk;
```

```
end
Alpha =[Alpha,alpha];
xold=xk;
xk = x_new;
Xit =[Xit; xk'];
Fs =[Fs;f1(xk), f2(xk), f3(xk)];
end
t=toc;
min_f = F(xk);
total_f_eval=falpha_count+k;

%% PLOT
fprintf('The optimized value of F(x): %f\n', min_f);
fprintf('Elasped time is: %f seconds\n', t)
fprintf('Total iteration number is: %i\n', k)
fprintf('Total function evaluation at Alpha is: %i\n', falpha_count)
fprintf('Total function evaluation is: %i\n', total_f_eval)
```