FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF BACHELOR'S THESIS

Title:	Nonlinear conjugate gradient method
Student:	Aleksandr Efremov
Supervisor:	doc. Ing. Ivan Šimeček, Ph.D.
Study Programme:	Informatics
Study Branch:	Computer Science
Department:	Department of Theoretical Computer Science
Validity:	Until the end of winter semester 2019/20

Instructions

1) Study the nonlinear conjugate gradient method [1,2].
2) Discuss advantages and drawbacks of different choices for the CG update parameter and restart criteria.
3) Explore various proposed hybridizations of the basic methods (see, e.g., [1]).
4) Compare the convergence properties of different methods.
5) Consider possible optimizations (e.g. heuristic scheme for parameter selection).
6) Implement the improved method, analyze achieved results.

References

[1] William W Hager and Hongchao Zhang. A survey of nonlinear conjugate gradient methods. Pacific journal of Optimization, 2(1):35–58, 2006.
[2] Jonathan Richard Shewchuk et al. An introduction to the conjugate gradient method without the agonizing pain, 1994.

doc. Ing. Jan Janoušek, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague March 1, 2018

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Nonlinear Conjugate Gradient Method

Aleksandr Efremov

Department of Theoretical Computer Science
Supervisor: doc. Ing. Ivan Šimeček, Ph.D.

May 15, 2018

Declaration

In Prague on May 15, 2018 ....

Citation of this thesis

Efremov, Aleksandr. *Nonlinear Conjugate Gradient Method.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2018.

Abstract

In this thesis we study nonlinear conjugate gradient methods for unconstrained optimization. We outline the possibilities and limits of existing methods for unconstrained optimization. Theoretical properties of the conjugate gradient methods are compared with other basic algorithms. The thesis reviews different variants of nonlinear conjugate gradient methods.

The conjugate gradient update parameter plays an important role in the convergence properties of nonlinear conjugate gradient method. Several formulas for the conjugate gradient update parameter exist and were proven to have plausible convergence properties. No generally optimal choice exists. In practice, the performance with different choices of the formula can vary significantly on different problems.

We propose a heuristic method that automatically adjusts the value of the parameter. Numerical results show that the performance of the heuristic method is often close to the performance of the best choice of the formula for a given problem.

Keywords conjugate gradient method, unconstrained optimization, continuous optimization, local optimization, mathematical optimization

Abstrakt

V této práci studujeme nelineární metody sdružených gradientů pro nepodmíněnou optimalizaci. Uvádíme možnosti a limity stávajících metod nepodmíněné optimalizace. Teoretické vlastnosti metod sdružených gradientů se porovnávají s dalšími základními algoritmy. Práce porovnává různé varianty nelineárních metod sdružených gradientů.

Parametr β (tzv. conjugate gradient update parameter) má významný vliv na konvergenční vlastnosty nelineárních metod sdružených gradientů. Existuje několik vzorců pro volbu parametru které mají vyhovující konvergenční vlastnosti, nicméně neexistuje optimální volba. V praxi se výkon metody s různými vzorci může výrazně lišit v závislosti na různých problémech.

Navrhujeme heuristickou metodu, která automaticky upravuje hodnotu parametru. Experimentální výsledky ukazují, že výkonnost heuristické metody je často blízká výkonnosti nejlepší volby vzorce pro daný problém.

Klíčová slova metoda sdružených gradientů, nepodmíněná optimalizace, spojitá optimalizace, lokální optimalizace, matematická optimalizace

Contents

List of Figures

List of Tables

Introduction

Mathematical optimization is widely used in science, engineering, economics, and industry. An optimization problem can be defined as the problem of finding the best solution between all feasible solutions where a solution is a set of variables and its “cost” is determined by some objective function.

In this thesis, we focus our attention on unconstrained continuous optimization problems where variables are real numbers and objective functions are multivariate real functions. That is, we have no constraints and any vector from $\mathbb{R}^n$ is a feasible solution. Therefore, given some objective function $f : \mathbb{R}^n \to \mathbb{R}$, our problem is to find $x \in \mathbb{R}^n$ that minimizes $f(x)$. The problem might appear to be easy on the first sight, but we will soon find out that, in general, such problems are unsolvable and our best hope is to find an approximate solution numerically. In recent years, there has been a growing interest in solving unconstrained optimization problems, since such problems often arise in increasingly popular field of machine learning.

The main interest of the thesis is nonlinear conjugate gradient methods, which are widely used in many practical large-scale optimization problems because of their simplicity and low memory requirements. By “methods” here we mostly refer to variants of a general method that differ by the choice of a single parameter β. Motivated by the fact that no optimal choice is currently known, we propose a heuristic method that automatically adjusts the value of the parameter as the iterations of the method evolve.

Thesis Structure

- Chapter 1 contains the necessary preliminaries on multivariate functions and optimality conditions.
- In Chapter 2 we discuss the possibilities and limitations of unconstrained optimization methods. To provide the necessary context in which nonlinear conjugate gradient methods are studied, we briefly outline several fundamental methods and discuss their properties.
- Chapter 3 starts with the description of the *linear* conjugate gradient method that was initially proposed to minimize quadratic functions. Consequently, we describe its extension to the nonlinear conjugate gradient method for general functions. Properties of different choices for the direction update parameter β, hybrid methods and restart criteria are discussed.
- In Chapter 4 we propose a heuristic method, discuss the proof-of-concept implementation and present numerical results.

Chapter 1

Prerequisites

This chapter is devoted to a review of the required mathematical background on multivariate functions and optimality conditions.

1.1 Notation

For vectors in $\mathbb{R}^n$ we use bold font and normal font for their coordinates, e.g $\mathbf{x} = (x_1, \dots, x_n)$. Throughout this thesis we consider vectors in $\mathbb{R}^n$ as column vectors and use transpose symbol $\mathbf{x}^T$ to transform to row vectors when required. Similarly, we use bold font and capital letters for matrices and use $\mathbf{A}^T$ to denote the matrix transpose.

We often work with sequences of vectors $\{\mathbf{x}_0, \mathbf{x}_1, \dots\} = \{\mathbf{x}_k\}_{k=0}^{\infty}$ and use simplified notation $\{\mathbf{x}_k\}$. Thus, $\mathbf{x}_k$ is used to denote k-th vector in a sequence rather than k-th variable x_k.

We use $\|\mathbf{x}\|$ to denote the Euclidean norm, that is, for $\mathbf{x} \in \mathbb{R}^n$

$$\|\mathbf{x}\| = \|\mathbf{x}\|_2 = \sqrt{\mathbf{x}^T\mathbf{x}}.$$

1.2 Multivariate Functions

In this section we review the mathematical tools required to recognize minimas of multivariate functions. We assume that the reader is familiar with basic calculus concepts such as derivatives and continuity for univariate functions.

Gradient and Hessian

Unconstrained optimization algorithms often rely on derivative information about the objective function. Generalizations of derivatives to multivariate functions, the gradient and Hessian, are essential to multivariate optimization.

Definition 1.1 (Partial derivative, [1, p. 63])**.** Let $\{\mathbf{e}_1, \dots, \mathbf{e}_n\}$ be the natural basis of $\mathbb{R}^n$. For a function $f : \mathbb{R}^n \to \mathbb{R}$ we use $\partial_{x_i} f$ to denote the function $\mathbb{R}^n \to \mathbb{R}$ defined for any $\mathbf{x} = (x_1, \dots, x_n)$ as

$$\partial_{x_i} f(\mathbf{x}) = \lim_{t \to 0} \frac{f(\mathbf{x} + t\mathbf{e}_i) - f(\mathbf{x})}{t}$$

and we call it the *partial derivative of f with respect to variable x_i.*

We say that a function $f : \mathbb{R}^n \to \mathbb{R}$ is *differentiable* if all its partial derivatives exist for any $\mathbf{x} \in \mathbb{R}^n$. Moreover, if all partial derivatives are continuous, we say that f is *continuously differentiable.*

Definition 1.2 (Gradient, [1, p. 65])**.** Let $f : \mathbb{R}^n \to \mathbb{R}$ be a differentiable function. Then the function $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$ defined by

$$\nabla f = \begin{pmatrix} \partial_{x_1} f \\ \partial_{x_2} f \\ \vdots \\ \partial_{x_n} f \end{pmatrix},$$

is called the *gradient* of f.

Definition 1.3 (Hessian, [1, p. 65])**.** If ∇f is differentiable, we say that f is twice differentiable. Then the *Hessian* of f is the function $\nabla^2 f : \mathbb{R}^n \to \mathbb{R}^{n \times n}$ defined as

$$\nabla^2 f = \begin{pmatrix} \partial^2_{x_1 x_1} f & \partial^2_{x_2 x_1} f & \cdots & \partial^2_{x_n x_1} f \\ \partial^2_{x_1 x_2} f & \partial^2_{x_2 x_2} f & \cdots & \partial^2_{x_n x_2} f \\ \vdots & \vdots & \ddots & \vdots \\ \partial^2_{x_1 x_n} f & \partial^2_{x_2 x_n} f & \cdots & \partial^2_{x_n x_n} f \end{pmatrix},$$

where $\partial^2_{x_i x_j} f$ represents taking the partial derivative of f with respect to x_j first, then with respect to x_i.

Taylor's Theorem and Approximations

Taylor's theorem is the basis for many theoretical results in optimization. Here we present its reduced version which leads to an important idea of approximating general functions by suitable functions that are easy to analyze.

Theorem 1.1 (Taylor's Theorem [2, Theorem 2.1])**.** *Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable and that $\boldsymbol{\delta} \in \mathbb{R}^n$. Then there exists a constant $t \in (0, 1)$ such that*

$$f(\mathbf{x} + \boldsymbol{\delta}) = f(\mathbf{x}) + \nabla f(\mathbf{x} + t\boldsymbol{\delta})^T \boldsymbol{\delta} \tag{1.1}$$

for any $\mathbf{x} \in \mathbb{R}^n$. Also, we have:

$$f(\mathbf{x} + \boldsymbol{\delta}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^T \boldsymbol{\delta} + o(\|\boldsymbol{\delta}\|). \tag{1.2}$$

Moreover, if f is twice continuously differentiable, then there exists a constant $t \in (0, 1)$ such that

$$f(\mathbf{x} + \boldsymbol{\delta}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^T \boldsymbol{\delta} + \frac{1}{2} \boldsymbol{\delta}^T \nabla^2 f(\mathbf{x} + t\boldsymbol{\delta}) \boldsymbol{\delta}, \tag{1.3}$$

and

$$f(\mathbf{x} + \boldsymbol{\delta}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^T \boldsymbol{\delta} + \frac{1}{2} \boldsymbol{\delta}^T \nabla^2 f(\mathbf{x}) \boldsymbol{\delta} + o(\|\boldsymbol{\delta}\|^2). \tag{1.4}$$

Note that the order symbol o here relates to behavior near zero, that is, $f(x) = o(g(x))$ is used to mean that

$$\lim_{x \to 0} \frac{f(x)}{g(x)} = 0,$$

in other words, $f(x)$ goes to zero "faster" than $g(x)$. This notation is common in numerical analysis but, conversely, is typically used in computer science to denote [3, Chapter 3] a bound for $x \to \infty$ and can be quite misleading. Further in the thesis we use O and Ω notation with the typical computer science meaning of the growth rate as x goes to infinity.

Equations (1.2) and (1.4) lead to a straightforward way to approximate a function in a small neighborhood around any point $\mathbf{x}$. For a differentiable function f and a point $\mathbf{y} = \mathbf{x} + \boldsymbol{\delta}$, where $\|\boldsymbol{\delta}\|$ is sufficiently small, we obtain the following *linear* approximation:

$$f(\mathbf{y}) \approx f(\mathbf{x}) + \nabla f(\mathbf{x})^T \boldsymbol{\delta}. \tag{1.5}$$

Furthermore, for a twice differentiable function f we have the *quadratic* approximation

$$f(\mathbf{y}) \approx f(\mathbf{x}) + \nabla f(\mathbf{x})^T \boldsymbol{\delta} + \frac{1}{2} \boldsymbol{\delta}^T \nabla^2 f(\mathbf{x}) \boldsymbol{\delta}. \tag{1.6}$$

1.3 Optimality Conditions

In unconstrained optimization problems we are seeking for a solution that minimizes the objective function. In this section we define how such a solution can be recognized.

Generally, we would be the happiest if we find a solution $\mathbf{x}$ that minimizes the objective function f over all $\mathbb{R}^n$.

Definition 1.4 (Global minimum)**.** We call a point $\mathbf{x}^*$ a *global minimum* of a function f if $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{R}^n$.

However, we often do not have enough information about the function structure (or do not have the analytical tools to use the information from the formula of the function) and can only rely on local techniques.

Definition 1.5 (Local minimum)**.** A point $\mathbf{x}^*$ a *local minimum* of a function f if there exists $\epsilon > 0$ such that $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{R}^n$ and $\|\mathbf{x} - \mathbf{x}^*\| < \epsilon$.

Generally, given a local minimum point $\mathbf{x}^*$, there is no much easier way to decide if $\mathbf{x}^*$ is also a global minimum than to find *all* local minimas and compare their values. On the other hand, several conditions allow us to check if some point is a local minimum efficiently.

Theorem 1.2 (First-order necessary condition [2, Theorem 2.2])**.** *If $\mathbf{x}^*$ is a local minimum of a continuously differentiable function f then $\nabla f(\mathbf{x}^*) = \mathbf{0}$.*

Hence, if $\nabla f(\mathbf{x}^*) \neq \mathbf{0}$ we can say that $\mathbf{x}^*$ is not a local minimum. However $\nabla f(\mathbf{x}^*) = \mathbf{0}$ is not a sufficient condition of $\mathbf{x}^*$ being a minimum.

Definition 1.6 (Stationary point)**.** A point $\mathbf{x}^*$ is a *stationary point* of f if $\nabla f(\mathbf{x}^*) = \mathbf{0}$.

A stationary point can be a minimum, maximum, or neither of them (a saddle point). Therefore, additional information from the Hessian values is required.

Definition 1.7 (Positive definite matrix)**.** We say that a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is *positive definite* if $\mathbf{x}^T\mathbf{A}\mathbf{x} > 0$ for all nonzero vectors $\mathbf{x}$. If $\mathbf{x}^T\mathbf{A}\mathbf{x} \geq 0$ for all $\mathbf{x} \in \mathbb{R}^n$ we say that $\mathbf{A}$ is *positive semidefinite.*

Theorem 1.3 (Second-order necessary condition, [2, Theorem 2.3])**.** *If $\mathbf{x}^*$ is a local minimum of a twice continuously differentiable function f then $\nabla^2 f(\mathbf{x}^*)$ is positive semidefinite.*

Hence, a point is not a minimum if the Hessian is not positive semidefinite at this point. Consequently, we have the following sufficient condition:

Theorem 1.4 (Second-order sufficient condition, [2, Theorem 2.4])**.** *Let f be a twice continuously differentiable function. If $\nabla f(\mathbf{x}) = \mathbf{0}$ and $\nabla^2 f(\mathbf{x})$ is positive definite then the point $\mathbf{x}$ is a local minimum of f.*

Note that if $\nabla f(\mathbf{x}^*) = 0$ and $\nabla^2 f(\mathbf{x}^*)$ is positive semidefinite but not positive definite, then this second-order condition does not allow us to decide if $\mathbf{x}^*$ is a minimum and higher order tests or other methods must be used.

On the other hand, no such general conditions exist for global minimas. As a result, most of unconstrained optimization algorithms only aim at finding a local solution.

However, there exists a class of functions for which global and local optimization are equally hard.

Definition 1.8 (Strictly convex function)**.** A function $f : \mathbb{R}^n \to \mathbb{R}$ is *strictly convex* if

$$f(\lambda\mathbf{x} + (1-\lambda)\mathbf{y}) < \lambda f(\mathbf{x}) + (1-\lambda) f(\mathbf{y})$$

for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and all $0 \leq \lambda \leq 1$.

Theorem 1.5. *Let f be a differentiable strictly convex function. If f has a stationary point $\mathbf{x}^*$, then $\mathbf{x}^*$ is a unique global minimum of f.*

Therefore, a local optimum of a strictly convex function is also a global optimum (Figure 1.1) and can be found by local optimization methods. On the other hand, general nonconvex functions can have infinitely many local minimas with significaltly higher function values (Figure 1.2). Therefore, global optimization is naturally much harder for nonconvex functions.

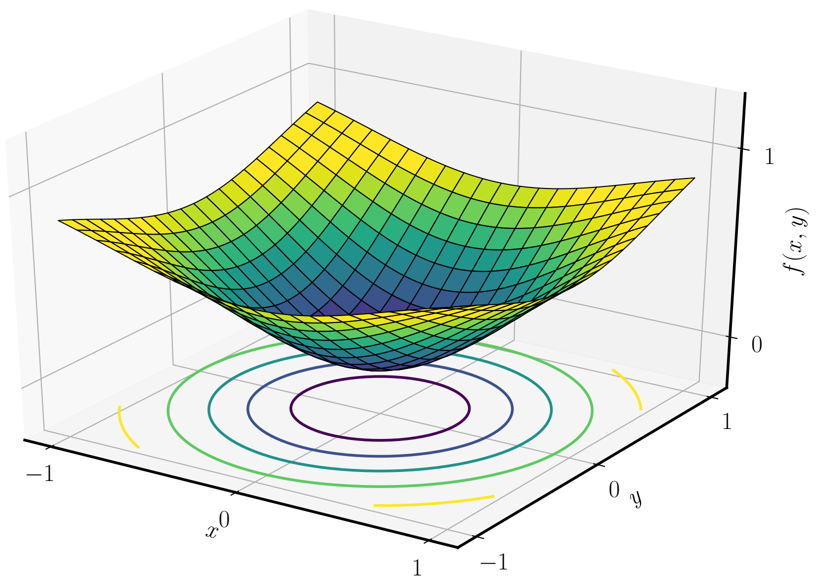


Figure 1.1: An example of a strictly convex function $f = 1 - e^{-(x^2+y^2)}$. It has a unique global minimum at $(0, 0)$.

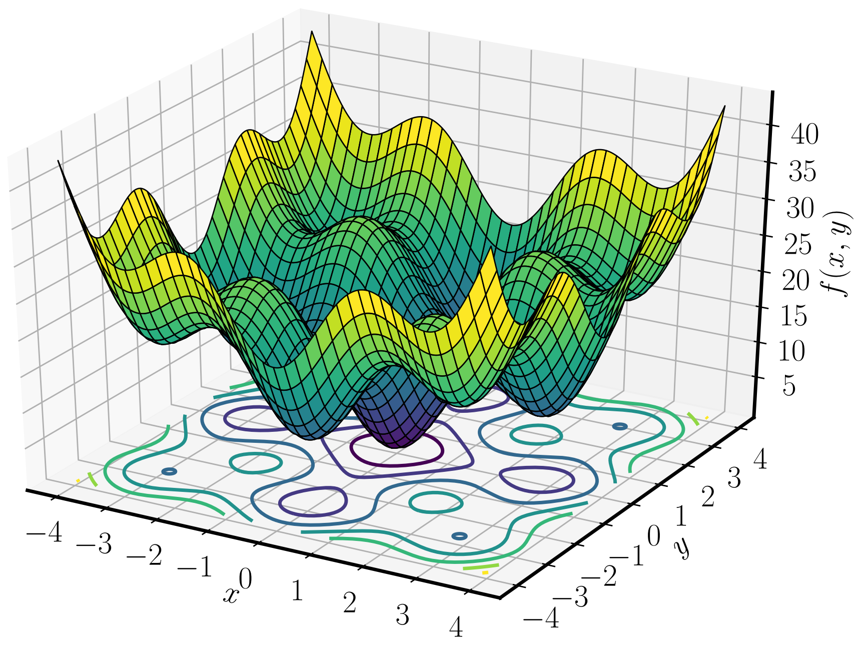


Figure 1.2: An example of a nonconvex function $f = x^2 + y^2 + 10 \sin^2 x + 10 \sin^2 y$. It has a global minimum at $(0, 0)$ and inifintely many local minimas with higher function values.

Chapter 2

Unconstrained Optimization

In this thesis we consider unconstrained optimization problems:

$$\begin{aligned}&\text{minimize } f(\mathbf{x})\\&\text{subject to } \mathbf{x} \in \mathbb{R}^n,\end{aligned}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a real-valued multivariate function.

In most of the cases, for an arbitrary, nonconvex function f we have little hope to find a global minimum. In fact, it is known that continuous global optimization is *undecidable*[1] [4]. Existing (general) methods for global optimization usually rely on some heuristic or randomization (e.g., particle swarm methods [5], simulated annealing [6]) and only *try* to solve the problem and offer no guarantee to find a global minimum in finite time.

Quite often, in real-world applications, many local minima have sufficiently low function values and it is not important to find the global minimum. In such cases local optimization methods are widely used. Local optimization theory is developed substantially deeper and many numerical methods are widely available. Moreover, most of the popular algorithms for global optimization make use of local methods and move from one local minimum to another, better one (e.g., tunneling algorithm [7], filled function method [8]). This thesis is subsequently focused on local optimization.

We also limit our attention to methods that are based on the information about function derivatives and require the objective functions to be *smooth*, that is, their second derivatives exist and are continuous. Nonsmooth optimization problems are not uncommon in practice. One possible way to solve them are derivative-free methods [9] which are not reviewed in this thesis, see, e.g., [2, Chapter 9]. Another ways are to use *subgradients*, generalization of the concept of gradients to non-differentiable functions [10], or *smoothing*, a way to approximate the objective function by a smooth function [11].

2.1 Methods for Local Optimization

Most practical algorithms for unconstrained local optimization have similar structure (Algorithm 2.1) that relies on the idea of *iterative descent*. The algorithm starts at some point $\mathbf{x}_0$ (an initial guess) and successively generates

[1]That is, there can exist no algorithm that solves every possible problem of the class in finite time.

a sequence of points $\{\mathbf{x}_k\}$, decreasing the value of f at each iteration:

$$f(\mathbf{x}_k) < f(\mathbf{x}_{k-1}),$$

until some local minimum $\mathbf{x}^*$ is found (or approximated with sufficient accuracy). The algorithm usually converges to a minimum that is close to the initial guess $\mathbf{x}_0$ (though not necessarily to the closest one). Therefore, if some knowledge about the objective function structure or rough location of satisfactory minimum is available, it is reasonable to choose $\mathbf{x}_0$ to be an estimate of the solution, thus guiding the algorithm towards the required minimum or speeding up the convergence. Otherwise, an initial guess may be chosen in some arbitrary manner.

Algorithm 2.1 General Iterative Descent $(f, \mathbf{x}_0)$

```
Determine the initial search direction $\mathbf{d}_0$;
Set $k \leftarrow 1$;
while $\mathbf{x}_{k-1}$ is not optimal do
    Determine the step size $\alpha_{k-1}$;
                    ▷ usually by minimizing $f$ along the line $\mathbf{x}_{k-1} + \alpha_{k-1}\mathbf{d}_{k-1}$
    Set $\mathbf{x}_k \leftarrow \mathbf{x}_{k-1} + \alpha_{k-1}\mathbf{d}_{k-1}$;
    Determine the next search direction $\mathbf{d}_k$;
    Set $k \leftarrow k + 1$;
end while
return $\mathbf{x}_{k-1}$;
```

In each iteration, algorithms of this type determine some descent direction $\mathbf{d}_k$, usually relying on first-order derivative (gradient ∇f) or second-order derivative (Hessian $\nabla^2 f$) information.

For a direction $\mathbf{d}_k$, the step size α_k is determined by a *line search* method and the next point $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$ is obtained so that the value of the objective function is decreased, i.e.

$$f(\mathbf{x}_{k+1}) = f(\mathbf{x}_k + \alpha_k \mathbf{d}_k) < f(\mathbf{x}_k). \tag{2.1}$$

The ideal choice for α_k would be to minimize f along the line $\mathbf{x}_k + \alpha \mathbf{d}_k$, that is

$$f(\mathbf{x}_k + \alpha_k \mathbf{d}_k) = \min_{\alpha \in \mathbb{R}} f(\mathbf{x}_k + \alpha \mathbf{d}_k). \tag{2.2}$$

Note that solving (2.2) is itself an unconstrained optimization problem of minimizing the one-dimensional function $\phi(\alpha) = f(\mathbf{x}_k + \alpha \mathbf{d}_k)$. For a general function f we do not have any explicit formula for the solution of (2.2). This exact line search strategy is completely theoretical and is never used in practice since even in one dimension case we cannot find an exact minimum of a function in finite time [12, p. 28]. Even finding a local minimum of ϕ with a sufficient precision can be expensive and require many evaluations of f and ∇f.

Instead, it is often sufficient to perform inexact search to find α that satisfies certain conditions that are required for the convergence of a particular method. It is not enough to simply reduce the function value by any amount for which the inequality (2.1) holds. Consider an example of a function which has minimum value $f(\mathbf{x}^*) = -1$ and a sequence of iterates $\{\mathbf{x}_k\}$ for which $f(\mathbf{x}_k) = c/k$ for some constant $c > 0$. Clearly, the values satisfy the inequality (2.1) but the sequence $\{f(\mathbf{x}_k)\}$ has its limit in zero and the algorithm never reaches the minimum. For this reason, some additional conditions must be enforced to ensure the convergence. In practice, efficient inexact line search methods exist and typically require a small number of function and gradient evaluations to find sufficiently good α. We will get back to the line search conditions in the context of nonlinear conjugate methods in the Section 3.3.

Another approach for the choice of direction and distance to the next iterate $\mathbf{x}_k$ are *trust region* methods [13]. The trust region approach first determines a maximum step length (trust region radius) and only then seeks a direction and step to improve the function value. Many common algorithms can be used with both line search and trust region approaches [2, Chapter 4], however there appears to be no straightforward way of using trust region strategy with the main interest of this thesis, conjugate gradient methods, and we do not discuss it further.

The Algorithm 2.1 can be terminated when the necessary condition for a local minimum is satisfied, i.e. $\nabla f(\mathbf{x}_{k-1}) = 0$. The sufficient condition, positive definiteness of $\nabla^2 f(\mathbf{x}_{k-1})$, is usually not checked in practice because of high computational cost of such tests. This implies that the algorithm may end up in any stationary point, i.e. local maximum or saddle point. Algorithms can indeed get stuck in any stationary point (see Figure 2.1), however, and practical experience suggests that algorithms are attracted to non-minimum points only under tolerably rare special conditions.

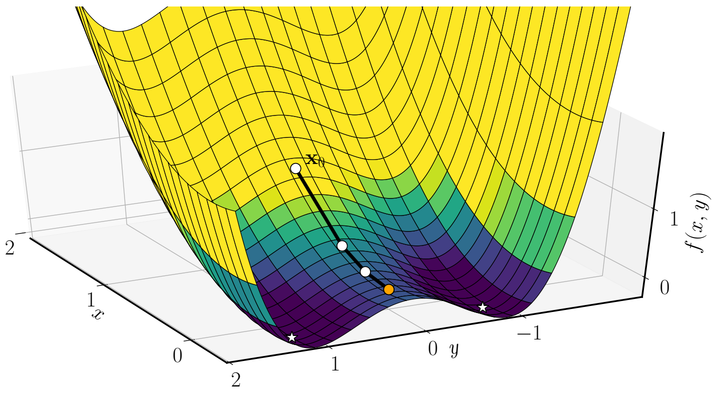


Figure 2.1: An example of a descent algorithm converging to a saddle point. Consider a function $f(x, y) = 0.5x^2 - 0.5y^2 + 0.25y^4$. It has two local minimas at points $(0, -1)$ and $(0, 1)$ and a saddle point $(0, 0)$. Now, consider a descent algorithm with an iteration of the type $\mathbf{x}_k = \mathbf{x}_{k-1} - \alpha\nabla f(\mathbf{x}_{k-1})$, where α is any scalar value. If started at any point with $y = 0$, all consecutive iterations have $y = 0$ and the algorithm has no chance to reach any of the mimimas and can only converge to a saddle point [12, Example 1.2.2].

However, as we will see in the next section, even local optimization methods are not guaranteed to reach a solution in a finite number of steps. Typically, algorithms are terminated when the gradient becomes sufficiently small, i.e. $\|\nabla f(\mathbf{x}_{k-1})\| < \epsilon$, or even $\|\nabla f(\mathbf{x}_{k-1})\| < \epsilon \|\nabla f(\mathbf{x}_0)\|$, for some tolerance value $\epsilon > 0$. This is also useful to resolve problems caused by round-off errors. Other, problem-dependent, termination criteria are used sometimes.

2.2 Complexity and Convergence Properties

In this section, we summarize the existing theoretical results for unconstrained optimization.

Despite the fact that local optimization appears to be "easier" than global optimization, it is still "hard" from computational complexity theory perspective. Local minimization is known to be NP-hard[2] in the worst case [14]. In contrast to discrete optimization, where rigorous theoretical classification in terms of complexity classes is often available, continuous optimization field appears to offer substantially less deep theoretical analysis and tends to study algorithms more from numerical, experimental perspective.

To our best knowledge, local minimization algorithms are only guaranteed to converge towards a solution in each iteration, and strictly convex quadratic functions are the only class of functions for which we have methods with finite termination guarantees. It is worth to mention, however, that after some finite number of iterations such methods reduce the function to a value that is essentially indistinguishable from the solution in finite precision arithmetic. Moreover, the optimal value of a general function can also be an irrational number, hence it has no finite representation in typical computational models anyway. Therefore, by "solving the problem" we often means finding an approximate solution $\mathbf{x}$ within some tolerance ϵ of the exact solution $\mathbf{x}^*$, e.g. $\|\mathbf{x} - \mathbf{x}^*\| < \epsilon$.

Global convergence

The major question is whether the sequence $\{\mathbf{x}_k\}$ converges to a local minimum $\mathbf{x}^*$ at all when an algorithm is started far from all minimas. Typical result [2, Section 3.2] for descent algorithms is

$$\lim_{k \to \infty} \|\nabla f(\mathbf{x}_k)\| = 0$$

under mild assumptions that f is bounded below on $\mathbb{R}^n$ and its gradient ∇f is continuous. In other words, we can be sure that iterations of the algorithm get closer to a stationary point in each iteration. However, this does not imply

[2]The class of NP-hard [3, Chapter 34] problems contains all the problems that are at least as hard as the hardest problems in NP. It is widely believed that for some of the problems in NP there can be no faster than exponential time algorithm. No efficient algorithms to solve NP-hard problems are currently known.

that the algorithm reaches the point in a finite number of iterations. In fact, in most of the cases, the algorithms are not guaranteed to reach a stationary point after finitely many iterations even if no round-off error occurs.

Furthermore, for some algorithms, including conjugate gradient methods, only slightly weaker result can be proven:

$$\liminf_{k\to\infty} \|\nabla f(\mathbf{x}_k)\| = 0.$$

That is, just some subsequence of gradients converges to 0. The algorithm still iterates towards a stationary point but in a less predictable manner.

Rate of local convergence

The mere knowledge of the fact that an algorithm converges towards a stationary point is of little value in practice unless we know how quickly it approaches the solution. Local convergence rate describes the speed of convergence of an algorithm when it reaches some sufficiently close neighborhood of the solution $\mathbf{x}^*$, where precise requirements for the neighborhood vary for different algorithms.

Results are usually [1, Section 8.4] of the type

$$\|\mathbf{x}_k - \mathbf{x}^*\| \le c\|\mathbf{x}_{k-1} - \mathbf{x}^*\|^p \tag{2.3}$$

for all sufficiently large k and constants $c, p > 0$. The higher the value of p, the faster the convergence.

In particular, when $c \in (0,1)$ and $p = 1$, we say that the convergence is *linear*. Expanding the inequality (2.3), we can get

$$\|\mathbf{x}_k - \mathbf{x}^*\| \le c^k\|\mathbf{x}_0 - \mathbf{x}^*\| + \xi, \tag{2.4}$$

where ξ is negligible, assuming that $\mathbf{x}_0$ lies inside the local convergence region and the sequence $\{\mathbf{x}_k\}$ behaves reasonably close to (2.3) for small k as well. This rate is quite fast. For example, to reduce the distance between the starting point $\mathbf{x}_0$ and the solution $\mathbf{x}^*$ by a factor of ϵ, i.e.

$$\|\mathbf{x}_k - \mathbf{x}^*\| \le \frac{1}{\epsilon}\|\mathbf{x}_0 - \mathbf{x}^*\|, \tag{2.5}$$

only $O(\log(1/\epsilon))$ iterations are required.

For $p = 2$, the convergence is *quadratic*. It follows that

$$\|\mathbf{x}_k - \mathbf{x}^*\| \le c^{(2^k-1)}\|\mathbf{x}_0 - \mathbf{x}^*\|^{2^k} + \xi. \tag{2.6}$$

This rate is extremely fast and the reduction (2.5) can be achieved in only $O(\log\log(1/\epsilon))$ iterations.

The cases where $p > 1$, or equivalently,

$$\lim_{k\to\infty} \frac{\|\mathbf{x}_k - \mathbf{x}^*\|}{\|\mathbf{x}_{k-1} - \mathbf{x}^*\|} = 0 \tag{2.7}$$

are referred to as *superlinear* convergence.

Local convergence does not precisely describe the rate of convergence of the initial iterations, which may be far from the solution. For general non-convex functions there are especially few guarantees of the size of the local convergence region and there is hardly any practical way to determine it beforehand. Mentioned results only show that *eventually* the algorithm will reach the region. Nonetheless, in practice the convergence of initial iterations is typically adequately fast.

Computational complexity and storage requirements

Computational complexity approach aims to quantify the number of elementary operations to find the solution exactly or within some ϵ tolerance. However, such analysis is complicated for several reasons.

As we have seen before, algorithms are not guaranteed to terminate in a finite number of iterations. Also, descent algorithms usually require to evaluate the objective function or its derivatives in every iteration to perform the line search and determine the direction. We consider the objective function f to be a “black-box”, i.e. that we can access the function through some “oracle” which can give us the function value (and, when required, derivatives) at any given point. The objective function can be arbitrarily complex, therefore it is impossible to bound the computational cost of its evaluation. Any computational complexity approach would also have to account for computation in real numbers and use the appropriate computational model.

Only a small number of nontrivial results is available. Computational cost of a single iteration, excluding the cost of the function evaluations and the line search is often studied. Storage requirements also play an important role. In many practical applications, e.g. deep learning, the dimension[3] n can typically be as large as several millions, thus making it impossible to use $O(n^2)$ memory to store any matrices. Moreover, even matrix-vector multiplication in time $O(n^2)$ can be impractical. In such conditions, preference is given to methods which avoid storage of any matrices and use only $O(n)$ memory for vectors and perform only vector-vector additions and scalar-vector multiplications in time $O(n)$. However, as we will see in Section 2.3, lower time and memory complexity is overbalanced by slower convergence.

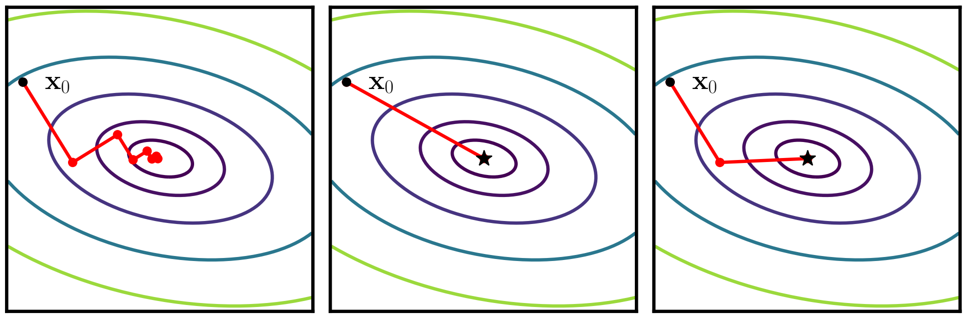
[3]Here and further in the text we use n to denote the dimension of the problem, i.e. $f : \mathbb{R}^n \to \mathbb{R}$ and $\mathbf{x}_i \in \mathbb{R}^n$.

2.3 Overview of Iterative Descent Algorithms

In this section, we provide a brief and informal overview of widely known algorithms based on the iterative descent. We describe the steepest descent method, Newton-type and Quasi-Newton methods, and, our main interest, conjugate gradient methods, which we further discuss in more detail in Chapter 3. More detailed analysis can be easily found in any of numerous books on nonlinear optimization. The algorithms differ mainly in the way of how descent direction is selected.

The reviewed methods have been known for a long time, and are still widely used in practice. Steepest descent method can be traced back to a paper of 1847 by Cauchy [15], and peak research interest in unconstrained local optimization methods was attained in the 1970s and 1980s. No substantially better methods have been found since then. More complex, specialized or heuristic algorithms were proposed, but there exists no single optimal algorithm that works well on all problems in general. This fact is also supported by the "No free lunch theorems for optimization" [16], which, informally, state that any two optimization methods have similar average performance when measured over all possible problems.

There exist several ways of how to interpret the algorithms. In the following review, we view them from the perspective of approximating function f by an appropriate model ϕ, which we can easily minimize analytically. Suppose that at iteration k we reached some point $\mathbf{x}_k$. By finding a suitable model ϕ_k and its minimum $\mathbf{y}^*$, we may hope that setting $\mathbf{x}_{k+1} = \mathbf{y}^*$ would bring us closer to the solution.



(a) Steepest descent (b) Newton's method (c) Conjugate gradient

Figure 2.2: Three methods applied to the quadratic function $f(\mathbf{x}) = \mathbf{x}^T \left(\begin{smallmatrix}2&1\\1&6\end{smallmatrix}\right)\mathbf{x}$. Plots show iterations of the methods on the contour plot of f. All three methods are started at the point $\mathbf{x}_0 = (-9, 5)$ and iterate towards the minimum at $(0, 0)$. Note that the steepest descent method does not reach the solution in a finite number of steps.

Steepest descent

The steepest descent is the simplest and least computationally expensive (per iteration) of the methods. It does not require to evaluate the Hessian and relies entirely on the gradient values. The main advantage of the method is its simplicity, both theoretical and practical. This leads to numerous analytical results of the convergence as well as wide application in practice.

Recall the first-order Taylor series expansion (1.2) of f around some stationary point $\mathbf{x}$. For every $\mathbf{y} \in \mathbb{R}^n$ we have

$$f(\mathbf{y}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y} - \mathbf{x}) + o(\|\mathbf{y} - \mathbf{x}\|).$$

Consider the approximation

$$\phi_k(\mathbf{y}) = f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T(\mathbf{y} - \mathbf{x}_k) + \frac{1}{2\alpha}\|\mathbf{y} - \mathbf{x}_k\|^2, \tag{2.8}$$

where α is some constant. Note that in terms of second-order Taylor series expansion (1.3) this can also be interpreted as approximating the Hessian by $\mathbf{I}/\alpha$, where $\mathbf{I}$ is the identity matrix.

Finding the stationary point $\mathbf{y}^*$ of ϕ from the equation

$$\nabla\phi_k(\mathbf{y}^*) = \nabla f(\mathbf{x}_k) + \frac{1}{\alpha}(\mathbf{y}^* - \mathbf{x}_k) = 0,$$

we obtain

$$\mathbf{y}^* = \mathbf{x}_k - \alpha\nabla f(\mathbf{x}_k),$$

The choice of α, such that the line minimization rule (2.2) holds, gives us the best possible approximation of type (2.8) that yields the greatest decrease of a function value. This leads exactly to the iterate of the steepest descent method:

$$\mathbf{x}_k = \mathbf{x}_{k-1} - \alpha_{k-1}\nabla f(\mathbf{x}_{k-1}). \tag{2.9}$$

The method derives its name from the fact that, among all possible directions, $\mathbf{d}_k = -\nabla f(\mathbf{x}_k)$ is the one along which f decreases most rapidly in the sense that

$$\mathbf{d}_k = \underset{\mathbf{v}\in\mathbb{R}^n}{\operatorname{argmin}} \left\{\nabla f(\mathbf{x}_k)^T\mathbf{v} \mid \|\mathbf{v}\| = \|\nabla f(\mathbf{x}_k)\|\right\}.$$

The method is strongly globally convergent even with practical inexact line search [2, p. 39] in the sense that

$$\lim_{k\to\infty} \|\nabla f(\mathbf{x}_k)\| = 0.$$

The iteration (2.9) requires only gradient evaluation and two vector operations. Hence, the step computation needs only $O(n)$ operations and $O(n)$ storage (excluding operations required for the line search and gradient evaluation). This is the best complexity among all the known algorithms of similar type.

The steepest descent converges linearly when sufficiently close to a stationary point [12, p. 34]. The fact that in each iteration the algorithm uses only the gradient at the current point and completely ignores any second-order information about the function is the primary reason behind its simplicity, but it is also a huge drawback. When the line search is exact, consecutive gradients, and therefore directions, are orthogonal, i.e $\mathbf{d}_{k+1}^T \mathbf{d}_k = \nabla f(\mathbf{x}_{k+1})^T \nabla f(\mathbf{x}_k) = 0$. This gives rise to a zigzagging behavior, where, in a sense, each iteration "undoes" part of the progress of the previous iteration. Even on some strictly convex quadratic functions the steepest descent method does not reach the solution in a finite number of iterations and converges towards it infinitely (Figure 2.2a).

Stochastic gradient descent

An important extension of the steepest descent method, the stochastic gradient method [17, Section 5.9] can be used to minimize functions that can be expressed as sums of other functions, i.e. $f(x) = \sum_{i=1}^{m} f_i(x)$. On such functions, the standard steepest descent method would need to evaluate the gradient of all m subfunctions per iteration, which can be prohibitively expensive. The iterations of the stochastic gradient descent are defined as

$$\begin{aligned}&\text{Randomly choose subset } t \in \{1, \dots, m\},\\ &\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_{k-1} \sum_{i \in t} \nabla f_i(\mathbf{x}_{k-1}).\end{aligned}$$

The size of the subset t can be fixed to be independent of m, hence only $O(1)$ gradients must be evaluated per iteration.

In expectation, the method is globally convergent when all functions f_i are convex [18]. The iteration of the method is much cheaper than the iteration of the steepest descent and achieves comparable progress.

Moreover, the method is often used without the line searches (which can be expensive) and the value of α is obtained by some efficient procedure [19], typically in time $O(1)$. Even though such methods need not to be globally convergent, they are extremely efficient when exact solution is not required and it is sufficient to achieve just a good decrease of the objective function value. For example, such methods are extremely popular in machine learning [17, Chapter 6].

Newton-type methods

In contrast to the steepest descent method, Newton-type methods are the more computationally expensive and make full use of the second-order information by evaluation the Hessian matrix in each iteration.

Given $\mathbf{x}_k$, the Newton's method obtains $\mathbf{x}_{k+1}$ by minimizing the quadratic approximation (1.6) of f around $\mathbf{x}_k$:

$$\phi_k(\mathbf{y}) = f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T(\mathbf{y} - \mathbf{x}_k) + \frac{1}{2}(\mathbf{y} - \mathbf{x}_k)^T \nabla^2 f(\mathbf{x}_k)(\mathbf{y} - \mathbf{x}_k).$$

A point $\mathbf{y}^*$ is a stationary point of ϕ_k if

$$\nabla \phi(\mathbf{y}^*) = \nabla f(\mathbf{x}_k) + \nabla^2 f(\mathbf{x}_k)(\mathbf{y}^* - \mathbf{x}_k) = 0.$$

Moreover, if $\nabla^2 f(\mathbf{x}_k)$ is invertible, we have that

$$\mathbf{y}^* = \mathbf{x}_k - (\nabla^2 f(\mathbf{x}_k))^{-1} \nabla f(\mathbf{x}_k)$$

and this is the pure Newton iteration. In order to avoid the possible divergence, more general iteration

$$\mathbf{x_k} = \mathbf{x}_{k-1} - \alpha_{k-1}(\nabla^2 f(\mathbf{x}_{k-1}))^{-1} \nabla f(\mathbf{x}_{k-1}) \tag{2.10}$$

is used in practice.

Curiously, the basic method of the form (2.10) is not guaranteed to converge globally when started far away from a stationary point and only converges locally when started sufficiently close [20]. Several methods exist to ensure global convergence [21]. Hessian is not guaranteed to be positive definite and the term $(\nabla^2 f(\mathbf{x}_{k-1}))^{-1}$ in (2.10) is often replaced by $(\nabla^2 f(\mathbf{x}_{k-1}) + c\mathbf{I})^{-1}$ for some choice of value c so that the matrix $\nabla^2 f(\mathbf{x}_{k-1}) + c\mathbf{I}$ is positive definite.

Newton methods attain fast quadratic rate of convergence (2.6) when close enough to a stationary point [20]. In particular, they minimize strictly convex quadratic functions in a single iteration (Figure 2.2b).

In practice, Newton-type methods are often much faster than steepest descent in terms of the number of iterations. The price to pay is a high iteration cost. The computational burden of evaluating, storing and inverting Hessian in each iteration makes Newton methods impractical even on problems with quite moderate dimension. As we have seen, Newton-type methods require to compute both gradient $\nabla f(\mathbf{x}_k)$ and Hessian inversion $(\nabla^2 f(\mathbf{x}_k))^{-1}$ in each iteration, which leads to $O(n^2)$ storage and up to $O(n^3)$ operations per iteration (excluding operations required for the line search and gradient evaluation).

Quasi-Newton methods

The motivation behind Quasi-Newton methods is to avoid the high computational cost of Newton-type methods, caused by the requirement to calculate the inverse Hessian at each step. Instead of working with the Hessian directly, Quasi-Newton methods maintain the approximation of the inverse Hessian $\mathbf{H}^{-1}$.

Then, each iteration is computed as

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

Therefore, the Hessian is no longer required to be evaluated and inverted at each step and time complexity is reduced to $O(n^2)$ operations required to compute the matrix-vector product. The method still requires $O(n^2)$ memory to store the matrix $\mathbf{H}^{-1}$.

Typically, Quasi-Newton methods do not achieve the quadratic convergence rate of Newton-type methods, but still attain quite fast superlinear rate (2.7) of local convergence [22].

However, for large-scale problems even $O(n^2)$ time and memory requirements can be prohibitive. This motivated the interest in the limited-memory Quasi-Newton methods. This algorithms do not store $\mathbf{H}_k^{-1}$ explicitly, but instead keep a certain information from m previous iterations, for some constant value of m, requiring only $O(mn)$ storage. This information allows to compute the approximate product $\mathbf{H}_k^{-1}\nabla f(\mathbf{x_k})$ by performing (excluding the cost of gradient evaluation) a constant number of vector operations in time $O(mn)$. Limited-memory methods have only linear rate of convergence in the worst case, but on many problems show comparable performance to standard Quasi-Newton methods and give satisfactory results even for small values of m, say $m \leq 7$ [23].

Conjugate gradient methods

Here we briefly outline the properties of conjugate gradient methods, which are deeply analyzed in Chapter 3.

When applied to a strictly convex quadratic function f, the linear conjugate gradient method iterations decrease the function value along the sequence of directions which are *conjugate* with respect to the Hessian matrix. This method avoids zigzagging behavior of the steepest descent method and is guaranteed to minimize a strictly convex quadratic function of n variables in at most n iterations (Figure 2.2c).

From Taylor's theorem (1.4) we can see that any function can be approximated by a positive definite quadratic function, when close enough to a local minimum. Hence, we can hope that the algorithm's fast convergence on quadratic functions can be applied towards general functions and yield good local convergence properties. In the next chapter, we will describe a remarkably straightforward way to extend the algorithm to its nonlinear version which minimizes any continuously differentiable functions.

Iterations of nonlinear conjugate gradient methods are of the form

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_{k-1}\mathbf{d}_{k-1},$$

where $\mathbf{d}_0$ is the direction of steepest descent, i.e $\mathbf{d}_0 = -\nabla f(\mathbf{x}_0)$, and

$$\mathbf{d}_k = -\nabla f(\mathbf{x}_k) + \beta_k \mathbf{d}_{k-1}, \quad \text{for all } k \geq 1, \tag{2.11}$$

where the scalar β_k can be obtained by various methods which we will discuss in detail in Chapter 3. Typical choices for β_k do not involve the evaluation of the Hessian and can be computed in $O(n)$ operations. Therefore, the whole iteration (excluding the cost of the line search and gradient evaluation) requires only $O(n)$ operations and space. Hence, each iteration is almost as simple to compute as the iteration of the steepest descent.

The motivation behind the directions (2.11) and avoiding the evaluation of Hessian is to keep the algorithm suitable for large-scale problems while attaining faster convergence rate than that of the steepest descent method. From the equation (2.11) we can see that at each iteration the search direction $\mathbf{d}_k$ can be expressed as a linear combination of gradients at preceding steps, i.e.

$$\mathbf{d}_k = \sum_{i=0}^{k} c_i \nabla f(\mathbf{x}_i),$$

where the precise values of coefficients c depend on the formula for β. Informally, we can hope that the direction update (2.11) with a suitable choice of β would allow the algorithm to accumulate enough second-order information about the function structure and would lead to a similar positive effect on performance as the implicit approximation of the Hessian in the case of limited-memory Quasi-Newton methods.

The global convergence depends on the choice of β and generally only the weaker result can be shown

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

which also holds for inexact line searches.

Crowder and Wolfe show [24] that the rate of convergence is only linear, even with exact line searches. Thus, as far as the theoretical results on convergence are concerned, conjugate gradient methods are not better than the steepest descent method. Nonetheless, conjugate gradient methods typically solve problems significantly more rapidly than the steepest descent method and are widely used in large-scale optimization. We devote Chapter 3 to more detailed description of conjugate gradient methods.

Comparison of methods

Generally, when the Hessian is available and it is affordable to invert and store it, one may expect to achieve the best results with Newton-type methods. Otherwise, Quasi-Newton methods attain reasonably close performance and are the method of choice when the requirement of $O(n^2)$ storage and $O(n^2)$ operations per iteration is tolerable.

One should also be aware of the cost of a function and gradient evaluation. If the computation needed for a function and gradient evaluation is $\Omega(n^2)$ operations, Quasi-Newton methods require only slightly more computation than methods with $O(n)$ iteration cost. Therefore, when $O(n^2)$ storage is available, Quasi-Newton methods are preferable in such cases.

In large-scale optimization, $O(n^2)$ storage can be prohibitive and functions typically can be evaluated in $O(n)$. Hence, limited-memory Quasi-Newton methods and conjugate gradient methods are preferred for solving large problems. There is no simple way to tell which one of them is the best for any particular problem and it is recommended to measure their performance experimentally [23]. Limited-memory Quasi-Newton methods typically need less iterations but are more complex and require more computation per iteration. On the other hand, conjugate gradient methods have the advantage of a very cheap iteration and are quite popular for that reason.

Nowadays, the steepest descent method is rarely used in its basic form on general functions. On the other hand, the stochastic gradient method is extremely popular and is perhaps the most frequently used optimization algorithm in statistics and machine learning, where it is often used without the line search procedure with constant values of α. This method is faster as there is no overhead related to running the one-dimensional optimization method in every iteration, and there is typically no need to find the exact minimum and good decrease of the function value is acceptable.

However, one must be careful when applying such kind of advice in practice. On some ill-posed problems even more complex methods can fail or be significantly slower than some of the simpler methods. Therefore, it is always recommended to perform numerical experiments on the available data.

Implementations

All the discussed methods are widely available in various mathematical software packages. Some examples are:

- MATLAB provides the function `fminunc` that is implemented with Quasi-Newton method [25].
- In Mathematica, the `FindMinimum` function can be used with Newton-type, Quasi-Newton or conjugate gradient methods [26].
- Scipy package for Python contains Newton-type, Quasi-Newton and conjugate gradient methods that can be accessed via the `scipy.optimize.minimize` function [27].
- NLopt library is written in C and provides interfaces to C++, Fortran, Julia, Python, R, etc. It provides Newton-type and Quasi-Newton methods for unconstrained optimization [28].

Chapter 3

Conjugate Gradient Methods

As we have seen in Chapter 2, conjugate gradient methods have attractive properties due to their simplicity. We devote this chapter to a deeper study of the method. We start our review with the linear conjugate method, predecessor of nonlinear methods, which was initially designed for solving linear systems of equations. After that we describe how the linear method was extended to a general nonlinear scheme. In the further sections, we describe different variants of the nonlinear method and their properties.

3.1 Linear Conjugate Gradient Method

Consider the problem of minimizing a quadratic function $f : \mathbb{R}^n \to \mathbb{R}$:

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x}, \tag{3.1}$$

where $\mathbf{A} \in \mathbb{R}^{n\times n}$ is a symmetric[4] positive definite matrix and $\mathbf{b} \in \mathbb{R}^n$.

For the symmetric $\mathbf{A}$ it holds that

$$\nabla f(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b}$$

and

$$\nabla^2 f(\mathbf{x}) = \mathbf{A}.$$

When $\mathbf{A}$ is positive definite, the function f is strictly convex and has the unique global minimum in the point $\mathbf{x}^*$ such that $\nabla f(\mathbf{x}^*) = \mathbf{0}$. Therefore, the solution to the problem of minimizing (3.1) is also a solution to the problem of finding $\mathbf{x} \in \mathbb{R}^n$ satisfying

$$\mathbf{A}\mathbf{x} = \mathbf{b}. \tag{3.2}$$

The conjugate gradient method was first introduced by Hestenes and Stiefel in 1950s [29] as an iterative method for solving linear systems of equations $\mathbf{A}\mathbf{x} = \mathbf{b}$ with square, symmetric positive definite $\mathbf{A} \in \mathbb{R}^{n\times n}$. It can also be applied to more general systems with invertible, but not symmetric or positive definite $\mathbf{A}$ after transformation to $\mathbf{A}^T\mathbf{A}x = \mathbf{A}^T\mathbf{b}$ (note that $\mathbf{A}^T\mathbf{A}$ is always positive definite). As we will see further in this section, the conjugate gradient method solves these problems in at most n iterations. This method is often referred to as the "*linear* conjugate gradient method".

[4]Note that the symmetricity of the matrix $\mathbf{A}$ is not a restriction. It is easy to show that any quadratic form $\mathbf{x}^T\mathbf{A}\mathbf{x}$ can be replaced by the equivalent $1/2 \cdot \mathbf{x}^T(\mathbf{A} + \mathbf{A}^T)\mathbf{x}$ and $(\mathbf{A} + \mathbf{A}^T)$ is always symmetric.

The name of the method stems from the fact that it produces a sequence of directions $\{\mathbf{d}_0, \dots \mathbf{d}_{n-1}\}$ such that the directions are *conjugate* with respect to matrix $\mathbf{A}$, that is

$$\mathbf{d}_i \mathbf{A} \mathbf{d}_j = 0, \quad \text{for all } i \neq j. \tag{3.3}$$

Algorithm 3.1 Linear Conjugate Gradient $(\mathbf{A}, \mathbf{b}, \mathbf{x}_0)$

```
Initialize      $\mathbf{r}_0 \leftarrow \mathbf{A}\mathbf{x}_0 - \mathbf{b};\quad \mathbf{d}_0 \leftarrow -\mathbf{r}_0;\quad k \leftarrow 1;$
while $\mathbf{r}_{k-1} \neq \mathbf{0}$ do
    $\alpha_{k-1} \leftarrow \frac{\mathbf{r}_{k-1}^T \mathbf{r}_{k-1}}{\mathbf{d}_{k-1}^T \mathbf{A} \mathbf{d}_{\mathbf{k-1}}};$      (3.4a)
    $\mathbf{x}_k \leftarrow \mathbf{x}_{k-1} + \alpha_{k-1} \mathbf{d}_{k-1};$      (3.4b)
    $\mathbf{r}_k \leftarrow \mathbf{r}_{k-1} + \alpha_{k-1} \mathbf{A} \mathbf{d}_{k-1};$      (3.4c)
    $\beta_k \leftarrow \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{r}_{k-1}^T \mathbf{r}_{k-1}};$      (3.4d)
    $\mathbf{d}_k \leftarrow -\mathbf{r}_k + \beta_k \mathbf{d}_{k-1};$      (3.4e)
    $k \leftarrow k + 1;$      (3.4f)
end while
return $\mathbf{x}_{k-1};$
```

The Algorithm (3.1) starts at some point x_0 (an initial guess that may be given as an input or selected arbitrarily, e.g., as a zero vector) and generates a sequence $\{\mathbf{x}_k\}$ by successively minimizing the quadratic function f (3.1) along the directions $\{\mathbf{d}_k\}$.

Values of $\mathbf{r}_k$ are exactly the values of the gradient $\nabla f(\mathbf{x}_k)$ (or, equivalently, the residual of the linear system: $\mathbf{r}_k = \mathbf{A}\mathbf{x}_k - \mathbf{b}$). Equation (3.4c) is used instead of direct gradient computation for efficiency reasons to avoid the matrix-vector product $\mathbf{A}\mathbf{x}_k$ and reuse the value $\mathbf{A}\mathbf{d}_{k-1}$ from previous iteration.

It is easy to show that, in case of the quadratic function f, the equation (3.4a) gives us the exact analytical solution for the one-dimensional minimization problem

$$f(\mathbf{x}_k + \alpha_k \mathbf{d}_k) = \min_{\alpha \in \mathbb{R}} f(\mathbf{x}_k + \alpha \mathbf{d}_k)$$

and no line search procedure is required.

Note that the method does not require to store the whole sequence of directions $\{\mathbf{d}_0, \dots, \mathbf{d}_{k-1}\}$ and computes each new direction $\mathbf{d}_k$ using only the negative gradient $-\nabla f(\mathbf{x}_k)$ (which is the steepest descent direction for f at point $\mathbf{x}_k$) and the previous direction $\mathbf{d}_{k-1}$. The choice (3.4d) of the scalar β_k ensures the conjugacy property (3.3). The name of the method is a bit misleading in the sense that it is the search directions, not the gradients (which are in fact orthogonal), that are conjugate with respect to $\mathbf{A}$.

For the proofs of the mentioned properties, we refer the interested reader to the original publication [29]. Many sources with modernized and simplified analysis are available, see, e.g., [30] or [2, Section 5.1].

Convergence

In their original paper [29], Hestenes and Steifel proved that conjugate gradient method finds the solution of a linear system in at most n steps, if no rounding-off error occurs.

Theorem 3.1 ([2, Theorem 5.2]). *For every $k > 0$, $\mathbf{x}_k$ minimizes f over the set*

$$\{\mathbf{x} \mid \mathbf{x} = \mathbf{x}_0 + \operatorname{span}(\mathbf{d}_0, \ldots, \mathbf{d}_{k-1})\},$$

where $\operatorname{span}(S)$ *denotes the set of all finite linear combinations of a set of vectors S, that is*

$$\operatorname{span}(\mathbf{v}_0, \ldots, \mathbf{v}_{n-1}) = \left\{ \sum_{i=0}^{n-1} c_i \mathbf{v}_i \;\middle|\; c_i \in \mathbb{R} \right\}.$$

A well-known fact that a set of conjugate vectors is linearly independent implies that $\mathbf{x}_0 + \operatorname{span}(\mathbf{d}_0, \ldots, \mathbf{d}_{k-1}) = \mathbb{R}^n$ and $\mathbf{x}_n$ minimizes f over $\mathbb{R}^n$. This directly leads to the following result.

Theorem 3.2. *For any starting point $\mathbf{x}_0 \in \mathbb{R}^n$ the sequence $\{\mathbf{x}_k\}$ generated by the Algorithm (3.1) converges to the solution of the system (3.2), or equivalently, to the minimum point of the function (3.1) in at most n steps.*

Another advantage of the method is that it can approach the solution in much fewer than n iterations. Recall that for a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ *eigenvalues* $\lambda_1, \ldots, \lambda_n$ are scalars such that $\mathbf{A}\mathbf{v}_i = \lambda \mathbf{v}_i$ for some vectors $\mathbf{v}_i \in \mathbf{R}^n$. In particular, eigenvalues of positive definite matrices are all positive numbers. When eigenvalues of $\mathbf{A}$ are distributed in a favorable way, the algorithm approaches solution quickly [30]. For example, we have the following result.

Theorem 3.3 ([2, Theorem 5.4]). *If $\mathbf{A}$ has only m distinct eigenvalues, then the linear conjugate gradient method will terminate at the solution in at most m iterations.*

Preconditioning methods can be used to speed up the convergence by transforming the linear system to $\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b}$, where $\mathbf{M}^{-1}\mathbf{A}$ has more favorable eigenvalue distribution [30]. For example, the smaller the condition number of $\mathbf{A}$, the faster the convergence. For a matrix $\mathbf{A}$ with eigenvalues $0 < \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ the *condition number* is defined as $\kappa(\mathbf{A}) = \lambda_n / \lambda_1$.

Today, the linear conjugate gradient method is often applied to a large problems where it is infeasible to run n iterations and only approximate solutions are required. Usually, a sufficiently accurate solution is reached in much fewer than n iterations.

Complexity

In every step, the Algorithm (3.1) computes one matrix-vector product $\mathbf{Ad}_k$, two inner products $\mathbf{d}_k^T(\mathbf{Ad}_k)$ and $\mathbf{r}_k^T\mathbf{r}_k$, and three vector sums for $\mathbf{x}_k, \mathbf{r}_k, \mathbf{d}_k$. Therefore, the time-complexity per iteration is $O(n^2)$ and execution of n steps requires $O(n^3)$ operations. However, the matrix-vector multiplication can be performed in time $O(m)$, where m is the number of nonzero elements in the matrix. For many real-world problems $\mathbf{A}$ is sparse and $m \in O(n)$ and the algorithm has complexity of $O(mn)$. At any point in the algorithm we need to store the vectors $\mathbf{x}, \mathbf{r}, \mathbf{d}$ only for the current and previous iteration. Therefore the space complexity is $O(n)$. For large problems the linear conjugate gradient method has the advantage that it does not require to store any matrices.

3.2 Nonlinear Conjugate Gradient Methods

In 1960s Fletcher and Reeves showed [31] that the linear conjugate gradient method can be extended to minimize general smooth functions $f : \mathbb{R}^n \to \mathbb{R}$. This surprisingly simple extension gives us the *nonlinear* conjugate gradient method.

Algorithm 3.2 General Nonlinear Conjugate Gradient $(f, \mathbf{x}_0)$

```
Evaluate $\mathbf{g}_0 = \nabla f(\mathbf{x}_0)$;
Initialize
        $\mathbf{d}_0 \leftarrow -\mathbf{g}_0; \quad k \leftarrow 1;$
while $\mathbf{g}_{k-1} \neq \mathbf{0}$ do
        Compute $\alpha_{k-1}$;                                  (3.5a)
        $\mathbf{x}_k \leftarrow \mathbf{x}_{k-1} + \alpha_{k-1}\mathbf{d}_{k-1}$;    (3.5b)
        Evaluate $\mathbf{g}_k = \nabla f(\mathbf{x}_k)$;                    (3.5c)
        Compute $\beta_k$;                                       (3.5d)
        $\mathbf{d}_k \leftarrow -\mathbf{g}_k + \beta_k\mathbf{d}_{k-1}$;          (3.5e)
        $k \leftarrow k + 1$;                                    (3.5f)
end while
return $\mathbf{x}_{k-1}$;
```

Similarly to the linear method, the Algorithm 3.2 takes the first step in the direction of the steepest descent and generates consecutive directions by the formula (3.5e). For a general function, the conjugacy condition (3.3) is no longer well defined, as the Hessian matrix is no longer constant. The extension destroys all properties of the method, specific for quadratic functions. Thus, the nonlinear method is based on a heuristic idea that suitable choice of β, which we discuss in Section 3.5, would yield some conjugacy-like property and the method would be fast in a neighborhood of a local minimum, where the function is very close to its quadratic approximation. Different conjugate gradient methods mostly correspond to different choices of the parameter β, which we will discuss in the Section 3.5, but in the following review we will see that other improvements are also possible.

3.3 Line Search Methods

For a general function, we can no longer use the value of α (3.4a) which was used for quadratic functions and cannot obtain similar formula analytically. Therefore some practical inexact line search procedure must be used. As we have briefly mentioned in the Chapter 2, some condition on the step must be enforced for the algorithm to be convergent.

In the case of conjugate gradient algorithms, the Wolfe line search conditions [32] are the most frequently used, however they are also applicable to other algorithms mentioned in the Chapter 2.

The standard Wolfe conditions are:

$$f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) + \delta\alpha_k \nabla f(\mathbf{x}_k)^T \mathbf{d}_k, \tag{3.6}$$

$$\nabla f(\mathbf{x}_{k+1})^T \mathbf{d}_k \geq \sigma \nabla f(\mathbf{x}_k)^T \mathbf{d}_k, \tag{3.7}$$

where $\mathbf{d}_k$ must be a descent direction, i.e. $\nabla f(\mathbf{x}_k)^T \mathbf{d}_k < 0$ and δ, σ are constants such that $0 < \delta \leq \sigma < 1$. The condition (3.6) ensures that the reduction of function value $f(\mathbf{x}_k + \alpha_k \mathbf{d}_k) - f(\mathbf{x}_k)$ is proportional to the value of α_k and the derivative along the direction $\mathbf{d}_k$. In practice, quite small values of δ are used, e.g., $\delta = 10^{-4}$. The second requirement, often called the *curvature condition*, ensures that the slope of the function at the new point is greater than the initial slope.

The strong Wolfe conditions enforce the stronger bound instead of the condition (3.7):

$$|\nabla f(\mathbf{x}_{k+1})^T \mathbf{d}_k| \leq \sigma |\nabla f(\mathbf{x}_k)^T \mathbf{d}_k|. \tag{3.8}$$

For the conjugate gradient algorithms, small value of $\sigma = 0.1$ is preferable, as it ensures higher search precision, even though slightly more computation is required to satisfy the condition compared to higher values of σ.

Sometimes generalized Wolfe conditions are used, where the curvature bound (3.8) is replaced by more general

$$\sigma_1 \nabla f(\mathbf{x}_k)^T \mathbf{d}_k \leq \nabla f(\mathbf{x}_{k+1})^T \mathbf{d}_k \leq -\sigma_2 \nabla f(\mathbf{x}_k)^T \mathbf{d}_k. \tag{3.9}$$

There hardly exist more intuitive explanations for the Wolfe conditions, however they play crucial role in every convergence proof. For our purpose, it is important to know that they can always be satisfied for every function that is smooth and bounded from below [2, Lemma 3.1], and practical algorithms exist (e.g. a method by Moré and Thuente [33] for strong Wolfe conditions) that can find suitable α using just a small number of function and gradient evaluation. In order to ensure good results in finite-precision arithmetic, Wolfe line search algorithms are based on rather technical and complex steps and we do not discuss them in this thesis.

In 2005, Hager and Zhang introduced [34] the *approximate Wolfe conditions*:

$$\sigma \nabla f(\mathbf{x}_k)^T \mathbf{d}_k \leq \nabla f(\mathbf{x}_{k+1})^T \mathbf{d}_k \leq (1-2\delta) \nabla f(\mathbf{x}_k)^T \mathbf{d}_k, \tag{3.10}$$

where $\delta < \min\{0.5, \sigma\}$. The motivation for this change was to increase the search accuracy with finite precision arithmetic. It is believed that the higher accuracy of the step length can speed up the convergence of the algorithm. The method performed well in their numerical tests but no theoretical results exist that guarantee the global convergence.

In 2011, Day showed [35] that nonlinear conjugate gradient methods can be globally convergent when the step size α is chosen to be a constant value. This is a promising result as running a line search method in each iteration imposes significant overhead on the performance. However, the right choice of the constant requires some nontrivial information about the function structure and numerical results in the paper suggest that this method is unlikely to be efficient in practice.

3.4 Convergence Properties

We will see in the next section that nonlinear conjugate gradient methods with different β are known to be globally convergent under suitable line search conditions.

Formally, such results make the following mild assumptions [36] about the function:

Let $\mathcal{L}$ denote the set of all points where the value of f is smaller than $f(\mathbf{x}_0)$:

$$\mathcal{L} = \{x \in \mathbb{R}^n \mid f(\mathbf{x}) \leq f(\mathbf{x}_0)\}.$$

Smoothness Assumption: the function f is L-smooth[5] in some neighborhood $\mathcal{N}$ of $\mathcal{L}$. That is, there exists a constant $L > 0$ such that

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\|, \quad \text{for all } \mathbf{x}, \mathbf{y} \in \mathcal{N}.$$

Intuitively, L measures how much the gradient of f can change between two nearby points. The constant bound on L implies that the function does not make infinitely steep “jumps” and behaves “reasonably”.

Boundedness Assumption: there exists a constant B such that $\|\mathbf{x}\| \leq B$ for all $\mathbf{x} \in \mathcal{L}$.

In the following sections, we say that the method is globally convergent if, for every function satisfying the assumptions, the sequence of gradient norms converges to zero in the sense that

$$\liminf_{k\to\infty} \|\nabla f(\mathbf{x}_k)\| = 0. \tag{3.11}$$

[5]The same condition is often also referred to as ∇f being *Lipschitz continuous*

Equation (3.11) implies that there exists at least one cluster point of the sequence $\{\mathbf{x}_k\}$, which is a stationary point [36]. However, in practice the sequence would typically have only one cluster point.

As for the local convergence rate, nonlinear conjugate gradient methods converge linearly and, as far as theoretical results are concerned, there is no difference in the convergence rate for different choices of β. At the same time, numerical results suggest that their performance can vary greatly.

3.5 Formulas for the Direction Update Parameter

To simplify the notation, we define and use in the subsequent text

$$\begin{aligned}\mathbf{g}_k &= \nabla f(\mathbf{x}_k),\\ \mathbf{y}_k &= \mathbf{g}_{k+1} - \mathbf{g}_k,\\ \mathbf{s}_k &= \mathbf{x}_{k+1} - \mathbf{x}_k = \alpha_k \mathbf{d}_k.\end{aligned}$$

Many different formulas for the parameter β were proposed since the first publication of the method by Fletcher and Reeves. We now present several basic choices.

Fletcher-Reeves formula

In their initial paper [31], the following coefficient was used:

$$\beta_k^{FR} = \frac{\mathbf{g}_k^T \mathbf{g}_k}{\mathbf{g}_{k-1}^T \mathbf{g}_{k-1}} = \frac{\|\mathbf{g}_k\|^2}{\|\mathbf{g}_{k-1}\|^2}. \tag{3.12}$$

In 1970, Zoutendijk [37] showed that the FR method with the exact line search is globally convergent. Al-Baali extended [38] this result to the inexact strong Wolfe line searches (3.6), (3.8) with $\sigma < 0.5$. Powell [39] demonstrated that the FR method is susceptible to "jamming" behavior. That is, if the method generates a poor search direction, which is almost orthogonal to the gradient, then it is likely that the step will be tiny and the next direction will be poor as well. It follows that the FR method can generate long sequences of unproductive iterations and be very slow.

Polak-Ribiére formula

Another choice of the parameter was independently proposed by Polak and Ribiére [40] and Polyak [41] in 1969:

$$\beta_k^{PR} = \frac{\mathbf{g}_k^T \mathbf{y}_{k-1}}{\|\mathbf{g}_{k-1}\|^2} = \frac{\mathbf{g}_k^T (\mathbf{g}_k - \mathbf{g}_{k-1})}{\|\mathbf{g}_{k-1}\|^2}$$

Note that the Polak-Ribiére method avoids the drawback of the FR method. Whenever the previous step is tiny, it holds that $\mathbf{g}_{k+1} \approx \mathbf{g}_k$ and $\beta_{k+1}^{PR} \approx 0$.

From the direction update formula (3.5e) we can notice that the next direction $\mathbf{d}_{k+1}$ will be very close to the direction of the steepest descent $-\mathbf{g}_{k+1}$. Thus, whenever a poor direction is generated, the PR method essentially performs a restart and makes next move in the the steepest descent direction.

Quite surprisingly, the PR method is not globally convergent even with exact line searches on nonconvex functions. Powell [42] showed a 3 dimensional example where the PR method cycles infinitely without converging to a solution. He also suggested to modify the method by setting

$$\beta_k^{PR+} = \max\{0, \beta_k^{PR}\}. \tag{3.13}$$

Gilbert and Nocedal [43] proved that this method is globally convergent with strong Wolfe line searches (3.6), (3.8). There is no simple explanation of why setting $\beta = 0$ when β^{PR} becomes negative yields global convergence. The result of Gilbert and Nocedal is based on quite sophisticated proof which does not give much insight on why negative values of β cause nonconvergence of the PR method. Despite its simplicity, the PR+ method is regarded as one of the most efficient methods (see, e.g., performance evaluation by Dai and Ni [44]) and remains to be perhaps the most frequently used method.

Hestenes-Steifel formula

In the original method that was proposed for solving linear systems of equations by Hestenes and Steifel [29], the coefficient β was given by

$$\beta_k^{HS} = \frac{\mathbf{g}_k^T \mathbf{y}_{k-1}}{\mathbf{d}_{k-1}^T \mathbf{y}_{k-1}}.$$

Note that it differs from the formula (3.4d) that we used in our presentation of the Algorithm 3.1, but both formulas are in fact equivalent on quadratic functions and exact line searches. The formula (3.5) is interesting because of the remarkable property that it implies that

$$\mathbf{d}_{k+1}^T \mathbf{y}_k = 0 \tag{3.14}$$

holds on general functions with inexact searches as well [45]. For the linear algorithm on quadratic functions, the equation (3.14) is equivalent to the conjugacy property $\mathbf{d}_{k+1}^T \mathbf{A} \mathbf{d}_k = 0$. For this reason, the condition (3.14) is sometimes referred to as the *conjugacy condition* for general functions.

For the exact line search, $\beta^{HS} = \beta^{PR}$, therefore the HS method has similar properties to the PR method [45]. In particular, Powell's [42] counterexample of convergence also holds for the HS method and the following modification is used to ensure the global convergence:

$$\beta_k^{HS+} = \max\{0, \beta_k^{HS}\}.$$

Conjugate Descent formula

Fletcher proposed the conjugate descent methods with β calculated by

$$\beta_k^{CD} = \frac{\|\mathbf{g}_k\|^2}{-\mathbf{d}_{k-1}^T \mathbf{g}_{k-1}}.$$

The name "conjugate descent" stems from the fact that with the strong Wolfe line search the method is guaranteed to produce the *sufficient descent* direction, that is

$$\mathbf{g}_k^T \mathbf{d}_k < -c\|\mathbf{g}_k\|^2 \tag{3.15}$$

holds for all k and some constant $c > 0$. However, Dai and Yuan showed [46] that the CD method is not guaranteed to converge with the strong Wolfe line search conditions, but converges under the generalized strong Wolfe conditions (3.6), (3.9) with $\sigma_1 < 0$ and $\sigma_2 = 0$.

Dai-Yuan formula

The CD method turns out to be inferior even to the FR method [36] in practice. However, the sufficient descent property (3.15) motivated Dai and Yuan to seek for other methods with that property. They proposed [47] a new method with the sufficient descent property, where

$$\beta_k^{DY} = \frac{\|\mathbf{g}_k\|^2}{\mathbf{d}_{k-1}^T \mathbf{y}_{k-1}}$$

and proved that the standard Wolfe conditions (3.6), (3.7) are sufficient for its convergence.

Hager-Zhang formula

More recently, Hager and Zhang proposed [34] a new method with

$$\beta_k^N = \left(\mathbf{y}_{k-1} - 2\mathbf{d}_{k-1}\frac{\|\mathbf{y}_{k-1}\|^2}{\mathbf{d}_{k-1}^T \mathbf{y}_{k-1}}\right)^T \frac{\mathbf{g}_k}{\mathbf{d}_{k-1}^T \mathbf{y}_{k-1}}$$

They proved that the truncated method

$$\bar{\beta}_k^N = \max\{\beta_k^N, \eta_k\}, \tag{3.16}$$

where

$$\eta_k = \frac{-1}{\|\mathbf{d}_{k-1}\| \min\{\eta, \|\mathbf{g}_{k-1}\|\}}$$

and $\eta > 0$ is a constant ($\eta = 0.01$ in the experiments in [34]), is globally convergent with the standard Wolfe line searches (3.6), (3.7). Moreover, their implementation CG_DECENT, written in C, where the $\bar{\beta}^N$ method is combined with the approximate line searches (3.10), showed significantly improved numerical performance on average [34].

Discussion

We have outlined several of the best known methods for the conjugate gradient parameter choice. It is important to note that all the described methods reduce to the same linear conjugate gradient method (Algorithm 3.1), when applied to strictly convex quadratic functions with exact line searches. This is a favorable property and is perhaps one of the main reason for good performance on general functions, because, as we have mentioned in Chapter 1, any function is well approximated by a quadratic function in a local minimum neighborhood.

A reader may perhaps feel that the motivation behind the methods is described too briefly and obscurely, but, as far as we know, there is hardly any deeper motivation for the outlined methods. The primary aim in the design of the methods was to satisfy the existing results describing sufficient conditions for the global convergence. For example, see Theorem 2.1 and Theorem 2.2 in the survey by Hager and Zhang [45].

We would like to point out that the described methods (as well as the hybrid methods which are discussed in Section 3.6) rely purely on the function gradient values and can be computed in $O(n)$ operations. Daniel [48] proposed fundamentally different parameter that required the evaluation of the Hessian:

$$\beta_k^D = \frac{\mathbf{g}_k^T \nabla^2 f(\mathbf{x}_{k-1}) \mathbf{d}_{k-1}}{\mathbf{d}_{k-1}^T \nabla^2 f(\mathbf{x}_{k-1}) \mathbf{d}_{k-1}},$$

but it did not attain much further interest due to the high computational cost.

3.6 Hybrid Conjugate Gradient Methods

As we have seen in the previous section, various methods for the choice of the direction update parameter β exist and have different advantages and drawbacks. Therefore, hybrid methods have been proposed to combine the existing methods and try to attain the best features while minimizing the influence of drawbacks.

Early hybrid methods were aimed to combine methods, which were known to be globally convergent but were susceptible to jamming (e.g., FR, DY, CD), with the other type of methods, which performed better in practice but were not convergent without modifications (e.g., PR and HS).

Combinations of FR and PR methods

Touati-Ahmed and Storey [49] suggested the following hybrid method:

$$\beta_k = \begin{cases} \beta_k^{PR} & , \quad \text{if } 0 \le \beta_k^{PR} \le \beta_k^{FR}, \\ \beta_k^{FR} & , \quad \text{otherwise.} \end{cases}$$

Similar method was also proposed by Hu and Storey [50]:

$$\beta_k = \max\{0, \min\{\beta_k^{PR}, \beta_k^{FR}\}\}.$$

Nocedal and Gilbert [43] suggested

$$\beta_k = \max\{-\beta_k^{FR}, \min\{\beta_k^{PR}, \beta_k^{FR}\}\}$$

and consequently proved that any method with $|\beta_k| < \beta_k^{FR}$ is globally convergent with the strong Wolfe line search (3.6), (3.8). Numerical results by Dai [44] suggest that such combinations are not significantly better than the PR method, if at all.

DYHS method

On the other hand, the method suggested and proven to be globally convergent by Dai and Yuan [51] with

$$\beta_k^{DYHS} = \max\{0, \min\{\beta_k^{HS}, \beta_k^{DY}\}\} \tag{3.17}$$

turned out to perform much better than the PR method [51] and is believed to be one of the most efficient methods along with the PR+ method.

This is a good demonstration of the fact that different methods are designed in a heuristic manner and studied more from the perspective of numerical experiments rather than theoretical results. To our best knowledge, no precise theoretical explanations exist for the huge difference in performance between the DYHS method and methods based on a combination of FR and PR.

Discussion

Several other hybrid methods were proposed, e.g. methods based on parameterized combinations $c_1\beta_1 + c_2\beta_2$, however no significant improvement was achieved [45].

The parameter β plays a crucial role in the performance of a nonlinear conjugate method. To our best knowledge, no generally optimal choice for β currently exists, though PR+, DYHS and the method of Hager and Zhang are believed to be the best of currently known and give better numerical results on average. Furthermore, there exist no guidelines for the choice of the most suitable parameter β for a particular function types. In fact, on some problems, methods that are believed to be the best may be significantly inferior to some other method (see, for example, Table 2 and Table 8 in the testing by Dai [44]). This happens in quite a chaotic manner and, as far as we know, there were no attempts to establish if such behavior is caused by some properties of the parameter β, or the structure of particular objective functions, or by some other reason.

3.7 Restart Strategies

Another modification that is used in nonlinear conjugate gradient methods is to restart the iteration by setting $\mathbf{d}_k = -\nabla f(\mathbf{x}_k)$ when some *restart condition* is satisfied. Essentially, this is equivalent to starting the algorithm anew from the point $\mathbf{x}_k$.

Regular restarts and n-step quadratic convergence

In the very first paper on the application of conjugate gradient method to general functions, Fletcher and Reeves suggested [31] to restart the method every n steps. This idea comes from the observation that only the set of at most n vectors in n dimensional space can be conjugate. Even though conjugacy condition is not well defined for the nonlinear method, it seems plausible to restart the algorithm to prevent the imagined loss of conjugacy. Cohen [52] proved that, when this technique is applied, the algorithm attains n-step quadratic local convergence rate, that is

$$\|\mathbf{x}_{k+n} - \mathbf{x}^*\| \leq c\|\mathbf{x}_k - \mathbf{x}^*\|^2$$

for all sufficiently large k and some constant c. Applying the same reasoning as in the case of quadratic convergence (2.6), one can expect to reduce the distance between the first point $\mathbf{x}_0$ and the solution $\mathbf{x}^*$ by a factor of ϵ in $O(n \log\log(1/\epsilon))$ iterations when the region of local convergence is reached. However, this result is not so surprising, if we recall that any function is approximated well by a quadratic function near a minimum point. At some point, the algorithm will reach that quadratic region and will behave like the linear conjugate gradient method from that point.

Even though this result is interesting, it may not be very useful in practice. Firstly, when the method is applied to large-scale problems, one would typically expect to obtain satisfactory solution in much less than n iterations. Secondly, the n-step restart method does not take in consideration any information about the algorithm state, other than the number of iterations. For example, it is undesirable to restart the algorithm when it is in a favorable region and is making long steps towards the solution. Such restarts would make the algorithm discard all second-order information and would degrade the performance. On the other hand, it is unreasonable to wait for the next n-th iteration if the algorithm is stuck in a sequence of tiny steps. This motivated the interest in other restart strategies that are based on other conditions than iteration counts.

Restart conditions

One popular strategy suggested by Powell [39] makes use of the fact that the gradients are mutually orthogonal when the linear conjugate gradient method is applied to a quadratic functions. Even though it does not hold for nonlinear method on general functions, it is reasonable to expect that gradients are close to being orthogonal. Therefore, the algorithm is restarted when

$$|\mathbf{g}_k^T \mathbf{g}_{k-1}| \geq 0.1 \|\mathbf{g}_k\|^2. \tag{3.18}$$

Powell also suggested to restart the method when the direction is too far from the steepest descent direction, i.e. the following condition is *not* satisfied

$$-1.2\|\mathbf{g}_k\|^2 \geq \mathbf{g}_k^T \mathbf{d}_k \geq -0.8\|\mathbf{g}_k\|^2.$$

Birgin and Martínez [53] proposed similar condition that restarts the algorithm whenever the direction is almost orthogonal to the gradient and the next step is likely to be tiny:

$$|\mathbf{g}_k^T \mathbf{d}_k| > -10^{-3}\|\mathbf{g}_k\|\|\mathbf{d}_k\|.$$

Discussion

Similarly to the choice of the parameter β, no choice of the restart condition is guaranteed to be optimal and it is recommended to try different options or combinations of them. In some cases, an algorithm can achieve the best performance without any restarts at all.

Chapter **4**

Heuristic Method

In this chapter we propose a heuristic method for the parameter β selection. We describe our implementation and provide numerical results. Finally, directions for the further research are discussed.

4.1 Ideas for Heuristic Methods

Motivation

Motivated by the fact that no optimal choice of β exists, we seek for a generic procedure that would automatically select the appropriate parameter for a given function. A straightforward idea to achieve the optimal behavior is to take m different methods $\beta^1, \ldots, \beta^m$, and at each iteration k consecutively perform m independent iterations $\beta_k^1, \ldots, \beta_k^m$, stopping as soon as one of the methods reaches the solution. Obviously, such algorithm would terminate in the optimal (among the chosen methods) number of iterations but its efficiency is debatable. It would require us to perform m times more computation and use m times more storage (to store the vectors $\mathbf{x}^i, \mathbf{d}^i$ and $\mathbf{g}^i$ for each independent method β^i). As a result, it is likely that the performance of such algorithm would be inferior even to the worst of the selected methods.

Goals

In the resulting algorithm we would like to preserve the attractive properties of nonlinear conjugate gradient methods (Algorithm 3.2): low computational cost and memory consumption. Furthermore, it seems reasonable to maintain algorithm's close relation to the linear conjugate gradient method (Algorithm 3.1) due to the fact that the nonlinear method's efficiency by a large part stems from the efficiency of the linear method on quadratic functions. Therefore, we would like our algorithm to be equivalent to the linear conjugate gradient method on quadratic functions with exact line search.

The lack of the general knowledge about the reasons of bad performance of various methods rules out the possibility to make use of information that we can directly obtain from the function formula. We would also like to avoid the direct use of second-order derivative information as computing the Hessian would break the attractive property of conjugate gradient methods. Thus, we have decided to use only a *heuristic* approach, where at each iteration we would choose the value of β according to some *local* reasoning about the state of the algorithm and *hope* that it would result in a good performance globally.

Example

We show a somewhat naive initial idea to illustrate this heuristic approach. Again, we take a set of different formulas $\mathbf{b} = \{\beta^1, \ldots, \beta^m\}$. This time, we store only one instance of $\mathbf{x}, \mathbf{d}$ and $\mathbf{g}$ per iteration and perform a separate iteration for each β^i and, for the next step, among all $\mathbf{x}_k^i$ we take the one at which the function value is the smallest. That is:

Algorithm 4.1 A naive example of a heuristic approach

```
Evaluate $\mathbf{g}_0 = \nabla f(\mathbf{x}_0)$;
Initialize
        $\mathbf{d}_0 \leftarrow -\mathbf{g}_0; \quad k \leftarrow 1;$
        Compute $\alpha_0$;
        $\mathbf{x}_1 \leftarrow \mathbf{x}_0 + \alpha_0 \mathbf{d}_0$;
        Evaluate $\mathbf{g}_1 = \nabla f(\mathbf{x}_1)$;
while $\mathbf{g}_{k-1} \neq \mathbf{0}$ do
    for $\beta^i \in \mathbf{b}$ do
        Evaluate $\beta_k^i$;
        $\mathbf{d}_k^i \leftarrow -\mathbf{g}_k + \beta_k^i \mathbf{d}_{k-1}$;
        Perform independent line search to find $\alpha_k^i$;
        $\mathbf{x}_{k+1}^i \leftarrow \mathbf{x}_k + \alpha_k^i \mathbf{d}_k^i$;
        Evaluate $f(\mathbf{x}_{k+1}^i)$;
    end for
        Find $I = \operatorname*{argmin}_{1 \le i \le m} f(\mathbf{x}_{k+1}^i)$;
        Set $\mathbf{x}_{k+1} = \mathbf{x}_{k+1}^I$;
        Evaluate $\mathbf{g}_{k+1} = \nabla f(\mathbf{x}_{k+1})$;
        $k \leftarrow k + 1$;
end while
return $\mathbf{x}_{k-1}$;
```

Discussion

The obvious disadvantage of the Algorithm 4.1 is the need to perform additional line search and function evaluation per each β^i. The line search and function evaluation are typically the bottleneck of each iteration and the computational cost of this heuristic is too high to be of any use in practice.

Experiments on our set of benchmark problems (see Table 4.1) suggest that this method is unlikely to be of any use in practice. Even taking a larger number of various formulas for β (e.g. all mentioned in Chapter 3) reduces the number of iterations by at most 30% compared to the average number of iterations of the standard method (Algorithm 3.2), though on a significant number of problems there is no reduction at all. However, for m different formulas of β we have to perform approximately m times more function and gradient evaluations and the method is almost m times slower even for functions which take only $O(n)$ time to evaluate.

Numerical experiments (e.g. [44]) suggest that "better" formulas for β (e.g., β^{PR+} and β^{DYHS}) converge in a number of iterations that differs from the best formula for many problems by at most 30%. It is likely that any heuristic method can perform at most around 30% more computations per iteration to bring any significant improvement in overall performance. This allows us to perform only a small number of additional vector-vector operations per iteration. Therefore, we conjecture that, to be efficient, any heuristic algorithm has to rely only on rather limited amount of data available at each iteration: a previous point $\mathbf{x}_k$, function value $f(\mathbf{x}_k)$ (which is typically evaluated in the line search procedure), gradient value $\mathbf{g}_k$ and direction $\mathbf{d}_k$ and possibly the same entries from a constant number of previous iterations.

4.2 The New Method

Subsequently, we consider a new method, which again depends on a set of different methods $\mathbf{b} = \{\beta^1, \ldots, \beta^m\}$. Now, at each iteration we compute β_k as

$$\beta_k = \sum_{i=1}^{m} w_i \beta_k^i,$$

or equivalently

$$\beta_k = \mathbf{w}_k^T \mathbf{b}_k, \tag{4.1}$$

where $\mathbf{w} = (w_1, \ldots, w_m)$ is a vector of weights with each weight w_i describing how "good" is the value of β^i compared to other values of β. At each iteration, based on the values of β^i and some condition of optimality (to be discussed in the next subsection), we compute a vector of local weights $\mathbf{v}_k$ and update the global weights as

$$\mathbf{w}_k = \begin{cases} \mathbf{v}_1 & , \quad \text{if } k = 1, \\ (1-c)\mathbf{w}_{k-1} + c\mathbf{v}_k & , \quad \text{otherwise.} \end{cases} \tag{4.2}$$

The constant $0 \leq c \leq 1$ controls how strongly the local weights of each iteration influence the global weights. Note that setting $c = 0$ makes the algorithm ignore all the updates and uses the same weights as computed in the first iteration, i.e. $\mathbf{w}_k = \mathbf{v}_1$. On the other hand, with $c = 1$ all the previous weights are ignored and $\mathbf{w}_k = \mathbf{v}_k$.

We also maintain an important property of $\mathbf{w}$ at each iteration:

$$w_i \geq 0, \text{ for } 1 \leq i \leq m, \tag{4.3}$$

and

$$\sum_{i=1}^{m} w_i = 1. \tag{4.4}$$

We feel that this property is important for two reasons:

Firstly, if β_k is computed by (4.2), it is easy to show that the following inequality holds:

$$\min_{1\leq i\leq m} \beta_k^i \leq \beta_k \leq \max_{1\leq i\leq m} \beta_k^i.$$

Although we do not have a direct proof, we believe that this fact plays an important role in the global convergence of our algorithm (which happens in our numerical experiments). If methods with the chosen formulas β^i are globally convergent under the chosen line search condition, it is likely that our method is globally convergent too.

Secondly, the property of $\mathbf{w}$ allows us to use an alternative approach to (4.2) to compute β_k. We can let β_k be a discrete random variable such that

$$\Pr[\beta_k = \beta_k^i] = w_i. \tag{4.5}$$

Then, we have that the expected value of β_k is

$$\mathbb{E}[\beta_k] = \mathbf{w}_k^T \mathbf{b}_k,$$

which exactly matches the equation (4.2). One may hope that the fact that the value of β_k exactly matches one of the formulas (i.e. $\beta_k = \beta_k^i$ for some i) may bring some plausible properties for our algorithm.

Optimality condition for β

Now it only remains to find a suitable way to determine how "good" is the value of β^i to compute the weights $\mathbf{v}$. This appears to be a nontrivial task as no way to find a generally optimal β (or even to determine which of two formulas β^i and β^j will give better results) is currently known. We describe a possible solution, however this remains to be an open problem and requires further research.

We need to find some condition to reason about the optimality of the values of β^i based on a very limited information available at each iteration. In Section 3.5 we have already mentioned[6] the extended *conjugacy condition* (3.14) for general functions:

$$\mathbf{d}_k^T \mathbf{y}_{k-1} = 0.$$

[6]Recall that we have defined $\mathbf{g}_k = \nabla f(\mathbf{x}_k)$, $\mathbf{y}_k = \mathbf{g}_{k+1} - \mathbf{g}_k$ and $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$.

As we have seen before, this condition holds when β is chosen by the HS method. It also holds for the linear method on quadratic functions with the exact line search. However, in practice the HS method is far from the most efficient one. In the case of the inexact line searches this condition can be unreasonable and even harmful for the performance. In case of inexact line searches, Perry [54] proposed to replace it by the following condition:

$$\mathbf{d}_k^T \mathbf{y}_{k-1} = -\mathbf{g}_k^T \mathbf{s}_{k-1}. \tag{4.6}$$

The condition is based on the fact that from the mean value theorem it follows [55] that, for any twice continuously differentiable function, there exists some $t \in [0, 1]$ such that

$$\mathbf{d}_k^T \mathbf{y}_{k-1} = \mathbf{d}_k^T \nabla^2 f(\mathbf{x}_{k-1} + t\alpha_{k-1}\mathbf{d}_{k-1})\mathbf{s}_{k-1}. \tag{4.7}$$

Perry proposed to use the approximation of the Hessian by a matrix $\mathbf{B}_k$ (also used in Quasi-Newton methods) such that

$$\mathbf{B}_k\mathbf{s}_{k-1} = \mathbf{y}_{k-1}, \text{ and} \tag{4.8}$$
$$\mathbf{B}_k\mathbf{d}_k = -\mathbf{g}_k. \tag{4.9}$$

Combining (4.7) and (4.9), we obtain

$$\mathbf{d}_k^T \mathbf{y}_{k-1} = \mathbf{d}_k^T(\mathbf{B}_k\mathbf{s}_{k-1}) = (\mathbf{B}_k\mathbf{d}_k)^T \mathbf{s}_{k-1} = -\mathbf{g}_k^T \mathbf{s}_{k-1},$$

which is exactly the condition (4.6).

Furthermore, Dai and Liao extended [56] the condition with a parameter t such that

$$\mathbf{d}_k^T \mathbf{y}_{k-1} = -t\mathbf{g}_k^T \mathbf{s}_{k-1}. \tag{4.10}$$

Finding the right t in some adaptive manner could probably improve the performance of an algorithm, however, no formula to choose t in an optimal manner is currently known. Hence, we leave it for the further research and consider $t = 1$, which is equivalent to the condition (4.6).

Weights update procedure

We describe how to find the vector of weights $\mathbf{v}_k$ based on the conjugacy condition (4.6). After having evaluated β_k^i, we can find the direction produced by this formula as

$$\mathbf{d}_k^i = -\mathbf{g}_k + \beta_k^i \mathbf{d}_{k-1}.$$

Subsequently, we can determine how "far" is the direction from satisfying the condition (4.6). Suppose that the condition (4.6) is satisfied by some $\mathbf{d}^{opt}$. Then we can compute the vector of "errors" $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_m)$ as

$$\gamma_i = |(\mathbf{d}_k^i)^T \mathbf{y}_{k-1} - (\mathbf{d}^{opt})^T \mathbf{y}_{k-1}| = |(\mathbf{d}_k^i)^T \mathbf{y}_{k-1} + \mathbf{g}_k^T \mathbf{s}_{k-1}|. \tag{4.11}$$

Then, we have to compute the vector of weights $\mathbf{v} = (v_1, \ldots, v_m)$ by giving higher weights to entries with smaller γ_i. Moreover, for the resulting vector $\mathbf{w}$ to satisfy the properties (4.3, 4.4), weights $\mathbf{v}$ must also satisfy the properties.

Hence, we need to rescale the vector of errors $\boldsymbol{\gamma}$ in a "reverse" order. The smaller is γ_i, the higher should be v_i. Additionally, all weights v_i should sum up to 1. One possible way to achieve it is to apply the *softmin* function to the vector $\boldsymbol{\gamma}$, which is defined as

$$\operatorname{softmin}(\boldsymbol{\gamma}) = \frac{\exp(-\boldsymbol{\gamma})}{\sum_{i=1}^{m} \exp(-\gamma_i)}, \tag{4.12}$$

where all vector operations are applied elementwise, for example $\exp(-\boldsymbol{\gamma}) = (\exp(-\gamma_1), \ldots, \exp(-\gamma_m))$.

One problem of using the function as defined by (4.12) is that when all values of γ_i are rather small, $\exp(-\gamma_i)$ is very close to 1, and it is likely that numerical result of $\operatorname{softmin}(\boldsymbol{\gamma})_i$ is indistinguishable from $1/m$ and no weights change happens.

To avoid that, we compute $\mathbf{v}$ as

$$\mathbf{v} = \operatorname{softmin}\left(\frac{\boldsymbol{\gamma}}{\mu}\right), \tag{4.13}$$

where μ is a mean value of the vector $\boldsymbol{\gamma}$, i.e.

$$\mu = \frac{\sum_{i=1}^{m} \gamma_i}{m}.$$

This rescaling procedure gives plausible results, although many other ways are possible. It appears that the exact way of weights rescaling has no significant influence on the performance of the algorithm.

The algorithm description

We can now present the algorithm in its entirety.

Algorithm 4.2 The heuristic algorithm$(f, \mathbf{x}_0, \mathbf{b} = (\beta^1, \ldots, \beta^m), c)$

```
Evaluate $\mathbf{g}_0 = \nabla f(\mathbf{x}_0)$;
Initialize
        $\mathbf{d}_0 \leftarrow -\mathbf{g}_0$;   $k \leftarrow 1$;   $w_i = 0$ for all $1 \leq i \leq m$;
while sufficiently good solutuion is not found do
        Compute $\alpha_{k-1}$;
        $\mathbf{x}_k \leftarrow \mathbf{x}_{k-1} + \alpha_{k-1}\mathbf{d}_{k-1}$;
        Evaluate $\mathbf{g}_k = \nabla f(\mathbf{x}_k)$;
        Evaluate $\mathbf{b}_k = (\beta_k^1, \ldots, \beta_k^m)$;
        Compute $\mathbf{d}_k^i$ for all $1 \leq i \leq m$ as
            $\mathbf{d}_k^i \leftarrow -\mathbf{g}_k + \beta_k^i \mathbf{d}_{k-1}$;
        Compute $\boldsymbol{\gamma}_k$ as in (4.11);
        Compute $\mathbf{v}_k$ as in (4.13);
        Compute $\mathbf{w}_k$ by (4.2);
        Compute $\beta_k$ either by (4.1) OR
            select randomly with probability (4.5);
        $\mathbf{d}_k \leftarrow -\mathbf{g}_k + \beta_k \mathbf{d}_{k-1}$;
        $k \leftarrow k + 1$;
end while
return $\mathbf{x}_{k-1}$;
```

Discussion of the algorithm

The Algorithm 4.2 has only rather moderate overhead per iteration compared to the standard conjugate gradient method (Algorithm 3.2). Per each iteration, $O(m)$ additional vector operations are performed, where m is the number of different formulas for β. To be precise, it takes around $2m$ vector operations to compute the directions $\mathbf{d}_k^i$ and the vector $\boldsymbol{\gamma}_k$, and a small constant number of operations to maintain other vectors. Additionally, the evaluation of the formulas β_k^i requires to perform some vector operations, however different formulas rely on similar dot products such as $\mathbf{g}_k^T \mathbf{g}_{k-1}$ and $\mathbf{g}_k^T \mathbf{d}_{k-1}$ which can be reused so it is likely that the evaluation of all formulas will take a lot less than m vector operations. It is important to note that no additional function or gradient evaluations are required.

4.3 Numerical Results

Our method (Algorithm 4.2) is compared with the standard method (Algorihm 3.2) with existing formulas for β (Section 3.5) on a subset of benchmark unconstrained problems from the CUTEst [57] testing environment. The methods are compared based on the number of iterations required to achieve the solution. Appendix A contains the description of the testing environment. Implementation details are explained in Appendix B.

For each unconstrained differentiable problem, the CUTEst testing environment provides the interface to evaluate the objective function f and its gradient ∇f at any point. An initial point $\mathbf{x}_0$ is also provided.

Tested methods

For our tests, we choose the subset of the formulas

$$\mathbf{B}_{test} = (\beta_{FR}, \beta_{PR+}, \beta_{DYHS}, \bar{\beta}_N)$$

The motivation for this choice is that β_{FR} is the basic method and β_{PR+}, β_{DYHS}, $\bar{\beta}_N$ show better results that other widely known formulas. We do not include other formulas mentioned in Section 3.5 and 3.6 as they show either slightly inferior or comparable results on our test problems.

The experiments on the following methods:

- FR : nonlinear conjugate gradient (Algorithm 3.2) with β_{FR} (3.12)
- PR+ : Algorithm 3.2 with β_{PR+} (3.13)
- DYHS : Algorithm 3.2 with β_{DYHS} (3.17)
- HZ : Algorithm 3.2 with $\bar{\beta}_N$ as proposed by Hager and Zhang (3.16)
- H$_{min}$: An algorithm from the example (Algorithm 4.1) with $\mathbf{B} = \mathbf{B}_{test}$
- H$_w$: The proposed heuristic method (Algorithm 4.2) with deterministic formula (4.1) and $\mathbf{B} = \mathbf{B}_{test}$
- H$_{rand}$: The proposed heuristic method (Algorithm 4.2) with probabilistic selection of formula (4.5) and $\mathbf{B} = \mathbf{B}_{test}$

Testing parameters

All methods are run with Moré-Thuente line search algorithm [33] that satisfies the strong Wolfe conditions (3.6, 3.8). We chose the strong Wolfe conditions over the standard Wolfe conditions (3.6, 3.7) as they are required for global convergence of methods FR and PR+. The values of δ and σ (see Section 3.3 for the discussion) are set to be 0.01 and 0.1.

The Moré-Thuente algorithm accepts an additional parameter, an initial guess of the value of α. Although it is not necessary for convergence, providing a suitable initial guess can reduce the number of operations significantly. As suggested in [2, Chapter 3.5], we use

$$\alpha\text{-guess}_k = \begin{cases} \frac{1}{\|\mathbf{g}_0\|} & , \quad \text{if } k = 0, \\ \alpha_{k-1} \frac{\mathbf{g}_{k-1}^T \mathbf{d}_{k-1}}{\mathbf{g}_k^T \mathbf{d}_k} & , \quad \text{otherwise.} \end{cases}$$

For the restarts (Section 3.7), the Powell's condition (3.18) is chosen, although other mentioned conditions give similar results on average and bring slight improvement over unrestarted method.

The used termination condition is

$$\|\mathbf{g}\| \leq 10^{-4}.$$

We set the maximum limit on iterations as $10n$, where n is the dimensionality of the problem. If a method fails to achieve the termination condition on some problem, we label its result as 'F'. Additionally, a method may fail internal conditions of the Moré-Thuente algorithm, which causes an error. This is not uncommon in tests done in the published papers (e.g., [34]) on very large problems. The errors are caused by precision problems, in our tests the errors happened when the value of α is very small, e.g., less than 10^{-13}, which is way beyond the adequate precision of the floating point computations. Then we label the result as 'E'. Otherwise, the result is the number of iterations required to achieve the termination condition.

There appears to be no generally optimal value for the weights update parameter c of our heuristic method H_w and H_{rand}, in this tests we show the results for $c = 0.25$, which seems to be a reasonable choice. In most of the cases, values of $0.1 < c < 0.9$ seem to have low effect on the performance.

The result of the probabilistic method H_{rand} is computed as the average result of 10 runs with different random generator seeds. For reproducibility purposes, we refer to the implementation or the implementation details in Appendix A.

Test problems selection

The CUTEst environment contains around 100 unconstrained test problems. Out of this problems, we selected a subset by the following criteria:

- We excluded the problems on which every method fails (typically problems of very high dimension).
- We excluded the problems on which every method succeeded in less than 50 iterations. Such tests are of a small interest since there would typically hardy be any difference in the performance between different methods.

- We excluded the problems on which different methods converged to different local minimas. Many test problems are nonconvex and have several minimas. Difference in the direction update can cause methods to converge to different minimas with which have significantly different function value and distance from the starting point. This happens chaotically and it is unfair to claim that one method is better than other if it converges to a closer minimum. Hence, we excluded such tests. Formally, we exclude a problem if the set of resulting points of all methods contains two points $\mathbf{x}_i^*, \mathbf{x}_j^*$ such that $\|\mathbf{x}_i^* - \mathbf{x}_j^*\| > 2 \cdot 10^{-3}$.

Additionally, some problems can be parameterized and be used with several different dimensionalities. On such problems, among all the variants, we took ones with highest and lowest dimensionalities which satisfy the previous criteria.

As a result, our testing set contains 35 problems, which may appear to be quite low, but it is typical for the research paper to use similar number of testing problems for numerical results (e.g., [44, 34]).

The names of the problems together with their dimensionalities are listed in Table 4.1. Initially, the problems in the CUTEst do not contain formulas of their functions, however, they can be found, e.g., in [58]. We do not include the formulas in this thesis, as it appears there is hardly any noticeable connection between the formula and the performance of methods on the problem (other than dimensionality).

Experiments

Test based on the number of iteration

The results of testing of all the methods can be seen in Table 4.1. Heuristic methods use the value of $c = 0.25$. Results of basic methods are comparable to those reported in other papers, e.g. [44]. Each cell contains the number of iteration in which an algorithm reaches the termination condition for a problem. If an algorithm does not terminate in the maximum iterations bound, we mark its result as 'F'. If it fails due to round-off errors, result is marked with 'E'.

Performance profile

Subsequently, we have use the performance profiles proposed in [59] to display the performance of each method, in terms of number of iterations. The performance profile plot (Figure 4.1) for a given method shows the fraction $p(\tau)$ of problems for which the method is within a factor τ of the best solver. Formally, for the set of problems $\mathcal{P}$, set of methods $\mathcal{S}$, and each problem $p \in \mathcal{P}$ and method $s \in \mathcal{S}$, let

$$t_{p,s} = \text{number of iterations required to solve } p \text{ by method } s.$$

If a method fails to solve a problem, we set $t_{p,s} = \infty$. The performance of method s on problem p is compared with the best performance by any solver on this problem. The performance ratio is defined as

$$r_{p,s} = \frac{t_{p,s}}{\min_{m \in \mathcal{S}} \{t_{p,m}\}}.$$

To obtain an overall assessment of the performance of the method, the performance profile

$$\rho_s(\tau) = \frac{1}{|\mathcal{P}|} |p \in \mathcal{P} : r_{p,s} < \tau| .$$

The value of $\rho_s(\tau)$ can be thought of as an estimate of the probability for the solver to have a performance within a factor of τ of the best possible performance for a random problem from $\mathcal{P}$. The methods with higher values of $\rho_s(\tau)$ are to be preferred as they are more likely to show near-optimal performance.

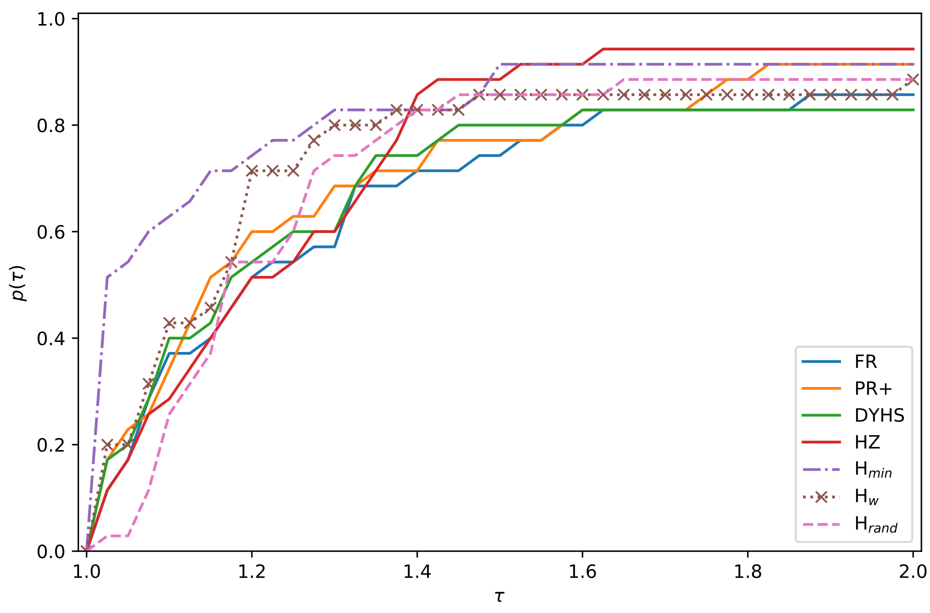


Figure 4.1: Performance profiles based on the number of iterations.

Values of weights update parameter

Additionally, in Table 4.2 we specifically compare the performance of H_w with different values of the weight update parameter c in order to determine its influence on the performance. Extreme values of $c = 0.1$ and $c = 0.9$ are compared to $c = 0.5$ and already tested $c = 0.25$.

Problem	N	FR	PR+	DYHS	HZ	H_{min}	H_w	H_{rand}
CHNROSNB	50	341	363	377	405	323	346	377
CRAGGLVY	100	72	84	68	85	73	80	74
DIXMAANE	3000	197	198	201	197	185	188	199
DIXMAANE	9000	299	277	322	319	294	302	317
DIXMAANG	3000	167	143	166	166	127	151	160
DIXMAANG	9000	326	207	248	270	207	233	261
DIXMAANH	3000	172	140	144	187	139	137	159
DIXMAANH	9000	211	372	263	280	217	230	301
DIXMAANJ	9000	157	223	129	175	190	188	165
DIXMAANK	9000	175	166	182	173	199	181	175
DIXMAANL	9000	142	155	166	153	160	164	156
DIXON3DQ	1000	4327	2948	7821	3177	2673	3656	3720
DECONVU	63	128	138	98	136	98	98	109
EIGENALS	110	259	271	229	308	340	288	250
EIGENBLS	110	343	349	334	344	312	333	344
EIGENCLS	462	1656	1830	1911	2158	1432	1712	1779
EIGENCLS	30	119	116	111	115	101	111	111
FLETCHCR	1000	7887	8350	7626	8543	7782	8037	8017
FMINSRF2	1024	449	378	255	1004	240	255	390
FMINSRF2	49	71	74	62	47	60	60	64
FMINSURF	1024	211	280	239	248	240	250	263
FMINSURF	5625	540	602	616	753	507	465	592
GENHUMPS	1000	4779	4721	4531	3726	1065	4713	4654
GENHUMPS	500	4647	F	1588	2215	4543	3161	F
GENROSE	500	2220	2225	2232	1859	2058	2185	2181
LIARWHD	10000	63	E	237	18	13	E	E
MOREBV	1000	39	30	40	37	38	38	35
PENALTY2	50	109	108	113	105	F	120	104
POWELLSG	10000	1285	58	180	33	32	171	68
POWELLSG	5000	672	50	666	46	112	96	58
POWER	10000	471	419	509	420	358	363	418
POWER	1000	143	117	149	116	126	103	119
SPARSINE	1000	2766	2830	6698	2910	4111	3260	3153
SPMSRTLS	1000	112	108	108	111	104	105	110
TRIDIA	5000	2290	2226	2503	2229	1579	1850	2121

Table 4.1: Numerical experiment based on the number of iterations.
First two columns represent the problem name and dimensionality. Numbers show the count of iterations untill the termination condition is satisfied. For every problem, we mark the best performance among all the methods with green color and the worst with red.

Problem	N	$c = 0.1$	$c = 0.5$	$c = 0.9$
CHNROSNB	50	343	348	347
CRAGGLVY	100	77	79	66
DIXMAANE	3000	187	188	186
DIXMAANE	9000	299	298	301
DIXMAANG	3000	133	138	154
DIXMAANG	9000	283	282	267
DIXMAANH	3000	150	152	151
DIXMAANH	9000	222	246	258
DIXMAANJ	9000	188	191	195
DIXMAANK	9000	183	184	187
DIXMAANL	9000	167	164	167
DIXON3DQ	1000	4147	2829	3595
DECONVU	63	103	124	103
EIGENALS	110	245	478	253
EIGENBLS	110	327	323	328
EIGENCLS	462	1537	1583	1698
EIGENCLS	30	111	118	112
FLETCHCR	1000	7976	7943	7970
FMINSRF2	1024	251	326	226
FMINSRF2	49	60	62	60
FMINSURF	1024	228	250	209
FMINSURF	5625	611	538	480
GENHUMPS	1000	4649	4598	4737
GENHUMPS	500	3721	636	2855
GENROSE	500	2152	2261	2147
LIARWHD	10000	24	19	17
MOREBV	1000	38	38	38
PENALTY2	50	158	136	120
POWELLSG	10000	102	66	40
POWELLSG	5000	72	40	92
POWER	10000	361	362	365
POWER	1000	104	106	104
SPARSINE	1000	3318	3499	3488
SPMSRTLS	1000	104	104	113
TRIDIA	5000	1733	2098	1616

Table 4.2: Experiment on the heuristic method H_w with different values of weights update parameter.

First two columns represent the problem name and dimensionality. Tested values are $c = 0.1$, $c = 0.5$ and $c = 0.9$. Numbers show the count of iterations untill the termination condition is satisfied. For every problem, we mark the best performance among all the methods with green color and the worst with red.

Discussion

Optimal range for weights update parameter

Firstly, we address the choice of weights update parameter c and discuss the results in Table 4.2. There appears to be no significantly outstanding choice, however the performance tends to improve slightly with the decrease of c in many cases. Smaller values of c mean that the algorithm gives more preference to its "memory" of previous weights, while higher values tend to overwrite previous weights quicker. After comparing the results in Table 4.2 and Table 4.1, we conclude that the choice of $c = 0.25$ was adequate.

Performance of the heuristic methods

Secondly, we attempt to reason about how the proposed heuristic algorithms compare to the existing formulas of β. The results in Table 4.1 show that the proposed methods converge to solutions on the test problems.

It is noticeable that the naive Algorithm 4.1 converges in the smallest number of iterations in most of the cases (although it is notable that it can also be the *worst* on some occasions). However, after comparing the actual numbers of iterations, we conclude that the improvement is rather marginal and is unlikely to compensate for the increased computational cost (see also the discussion in Section 4.1).

It is harder to say something about the Algorithm 4.2. It does not attain the best results, but it is also far from the worst case on the majority of the problems. Overall, the results in Table 4.1 appear to be quite chaotic, both for old and new methods.

Surprisingly, the probabilistic algorithm H_{rand} is rarely better than the deterministic version H_w. It appears that the randomized choice of β equal to one of the formulas (4.5) does not lead to the performance improvement. Additionally, the overhead of pseudo-random number generation is likely to make it even slower in practice.

The performance profile plot (Figure 4.1) allows us compared the performance of methods to the optimal performance for each problem. It turns out that the heuristic methods H_{min} and H_w are more likely to show near-optimal performance, compared to basic methods. The heuristic methods perform within the factor of approx. 1.4 of the best performance on more than 80% of problems and for values of $\tau \in [1.0, 1.4]$ have the highest probability among all the methods (except for the small range where the PR+ method outperforms the H_w method). For higher values of τ, the HZ algorithm is more robust, but only slightly. The profile also confirms that the H_{prob} method is inferior.

We end this section with a conclusion that the H_w method is significantly more likely to show close-to-optimal performance than the basic methods, however it is unlikely to be significantly faster than the best of the other methods.

4.4 Conclusion

Even though the proposed heuristic method (Algorithm 4.2) does not bring fundamental improvements, the experiments show that its performance is close to optimal with high probability. Hence, when the best formula of β for a given problem is not known in advance, it may be reasonable to apply the heuristic method with the weights update formula (4.1), as it ensures that the performance (measured as the number of iterations) will not be much worse than that of the best method. However, it also adds some computational overhead per iteration and it is hard to determine under what conditions it leads to overall performance improvement. Notably, the method does not require to perform additional function and gradient evaluations or line searches, hence it is likely to be practical when the cost of function evaluation is high and even a small number of additional iterations is expensive and undesirable.

4.5 Future Work

There is still a lot of space for further research and improvement. Here we describe possible future work directions.

Best choice of formulas β_i

We tested the proposed method with the choice of formulas

$$\mathbf{B} = (\beta_{FR}, \beta_{PR+}, \beta_{DYHS}, \bar{\beta}_N).$$

It is interesting to consider other combinations and determine if inclusion of some formula gives significant performance improvement or if some combinations are redundant.

Conjugacy condition

We have used the conjugacy condition (4.10) with the firm value of $t = 1$. It is natural to ask if there are better values of t and try to find an adaptive procedure to adjust the value of t to a local structure of the problem.

Optimality conditions

Currently, it is not known how to determine which values of β are better based on some local (or even global) information about the function. We have used the conjugacy condition (4.10) to rate the optimality of different vaules of β. It is interesting to find a better condition. In general, it could be a great breakthrough and lead to improved conjugate gradient methods.

Global convergence proof

The experiments showed that the proposed method converges to a solution on the test problems, when invoked with globally convergent formulas of β. Thus, the next step would be to attempt to find a proof of the global convergence using, e.g., Theorem 2.1 and Theorem 2.2 in [45], which are widely used in such proofs. However, it seems there is no straightforward way to proof the convergence of the generic method based on $\beta = \sum_i w_i \beta_i$ for any globally convergent β_i. Even proving that the algorithm decreases the function value at each iteration is hard and it seems that it is unavoidable that such proof must be based on formulas of particular choice of β_i as in the generic case, the fact that some property holds for the iterates of a method with $\beta = \beta_i$ does not directly imply that the same property holds for the method with $\beta = \sum_i w_i \beta_i$.

In several research papers (e.g., [60], [61]), methods with $\beta = c_1\beta_1 + c_2\beta_2$ are considered and convergence proofs are given, however every separate combination of β_1 and β_2 requires a separate proof based on the particular formulas. It is not currently known if global convergence of any β_1 and β_2 implies the convergence of the method with $\beta = c_1\beta_1 + c_2\beta_2$ and $c_1 + c_2 = 1$.

Conclusion

The goal of the thesis was to study methods for unconstrained optimization. In particular, we focused on the study of nonlinear conjugate gradient methods and attempted to find a better method.

We presented the theoretical background for the unconstrained optimization algorithms. The possibilities of unconstrained optimization algorithms and theoretical results were presented. We compared several basic methods such as the steepest descent method, Newton-type methods and conjugate gradient methods.

The properties of the conjugate gradient method were studied in detail. We outlined several options for the choice of the conjugate gradient update parameter as well as their possible combinations and the effect on the performance of the method.

A heuristic method based on several existing formulas was proposed. Our method automatically combines the selected formulas and adapts to the given problem, thus avoiding the drawbacks of existing hybrid methods. To our best knowledge, no similar ideas were considered in the existing research papers.

Numerical experiments showed that our heuristic method converges to the solutions of the test problems and often performs close to the best choice of the formula for a given problem and requires only a small computational overhead. In its current state, it does not bring a fundamental improvement, however, it gives a promising direction for the further research.

Bibliography

[1] E. K. Chong and S. H. Zak, *An introduction to optimization.* John Wiley & Sons, 4 ed., 2013.

[2] J. Nocedal and S. J. Wright, *Numerical Optimization.* Operations Research and Financial Engineering, Springer New York, 2 ed., 2006.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms.* MIT press, 3 ed., 2009.

[4] W. Zhu, "Unsolvability of some optimization problems," *Applied Mathematics and Computation*, vol. 174, no. 2, pp. 921–926, 2006.

[5] Y. Shi *et al.*, "Particle swarm optimization: developments, applications and resources," in *Proceedings of the 2001 Congress on Evolutionary Computation*, vol. 1, pp. 81–86, IEEE, 2001.

[6] M. Locatelli, "Simulated annealing algorithms for continuous global optimization: convergence conditions," *Journal of Optimization Theory and applications*, vol. 104, no. 1, pp. 121–133, 2000.

[7] A. V. Levy and A. Montalvo, "The tunneling algorithm for the global minimization of functions," *SIAM Journal on Scientific and Statistical Computing*, vol. 6, no. 1, pp. 15–29, 1985.

[8] G. Renpu, "A filled function method for finding a global minimizer of a function of several variables," *Mathematical programming*, vol. 46, no. 1-3, pp. 191–204, 1990.

[9] A. R. Conn, K. Scheinberg, and P. L. Toint, "Recent progress in unconstrained nonlinear optimization without derivatives," *Mathematical programming*, vol. 79, no. 1-3, p. 397, 1997.

[10] N. Z. Shor, *Minimization methods for non-differentiable functions.* Springer Science & Business Media, 3 ed., 2012.

[11] S. Xu, "Smoothing method for minimax problems," *Computational Optimization and Applications*, vol. 20, no. 3, pp. 267–279, 2001.

[12] Y. Nesterov, *Introductory Lectures on Convex Optimization: A Basic Course.* Springer Science & Business Media, 2003.

[13] J. J. Moré, "Recent developments in algorithms and software for trust region methods," in *Mathematical programming, The state of the art*, pp. 258–287, Springer, 1983.

[14] K. G. Murty and S. N. Kabadi, "Some NP-complete problems in quadratic and nonlinear programming," *Mathematical programming*, vol. 39, no. 2, pp. 117–129, 1987.

[15] A. Cauchy, "Méthode générale pour la résolution des systemes d'équations simultanées," *Comp. Rend. Sci. Paris*, vol. 25, no. 1847, pp. 536–538, 1847.

[16] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE transactions on evolutionary computation*, vol. 1, no. 1, pp. 67–82, 1997.

[17] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016.

[18] S. Bubeck *et al.*, "Convex optimization: Algorithms and complexity," *Foundations and Trends in Machine Learning*, vol. 8, no. 3-4, pp. 231–357, 2015.

[19] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint*, 2016. arXiv:1609.04747.

[20] J. J. Moré and D. C. Sorensen, "Newton's method," tech. rep., Argonne National Lab., IL (USA), 1982.

[21] W. Murray, "Newton-type methods," in *Wiley Encyclopedia of Operations Research and Management Science* (J. J. Cochran *et al.*, eds.), John Wiley & Sons, 2011.

[22] C. G. Broyden, J. Dennis Jr, and J. J. Moré, "On the local and superlinear convergence of quasi-newton methods," *IMA Journal of Applied Mathematics*, vol. 12, no. 3, pp. 223–245, 1973.

[23] J. Nocedal, "Theory of algorithms for unconstrained optimization," *Acta numerica*, vol. 1, pp. 199–242, 1992.

[24] H. Crowder and P. Wolfe, "Linear convergence of the conjugate gradient method," *IBM Journal of Research and Development*, vol. 16, no. 4, pp. 431–433, 1972.

[25] Mathworks, "Matlab documentation, `fminunc`." [online]. Accessed 2018-05-15.

[26] Wolfram, "Mathematica documentation, `FindMinimum`." [online]. Accessed 2018-05-15.

[27] "Scipy v1.1.0 reference guide, `scipy.optimize.minimize`." [online]. Accessed 2018-05-15.

[28] S. G. Johnson, "Nlopt documentation." [online]. Accessed 2018-05-15.

[29] M. R. Hestenes and E. Stiefel, "Methods of conjugate gradients for solving linear systems," *Journal of Research of the National Bureau of Standards*, vol. 49, no. 6, 1952.

[30] J. R. Shewchuk, "An introduction to the conjugate gradient method without the agonizing pain," tech. rep., Carnegie Mellon University, PA (USA), 1994.

[31] R. Fletcher and C. M. Reeves, "Function minimization by conjugate gradients," *The computer journal*, vol. 7, no. 2, pp. 149–154, 1964.

[32] P. Wolfe, "Convergence conditions for ascent methods," *SIAM review*, vol. 11, no. 2, pp. 226–235, 1969.

[33] J. J. Moré and D. J. Thuente, "Line search algorithms with guaranteed sufficient decrease," *ACM Transactions on Mathematical Software (TOMS)*, vol. 20, no. 3, pp. 286–307, 1994.

[34] W. W. Hager and H. Zhang, "A new conjugate gradient method with guaranteed descent and an efficient line search," *SIAM Journal on optimization*, vol. 16, no. 1, pp. 170–192, 2005.

[35] Y.-H. Dai, "Convergence of conjugate gradient methods with constant stepsizes," *Optimization Methods and Software*, vol. 26, no. 6, pp. 895–909, 2011.

[36] Y.-H. Dai, "Nonlinear conjugate gradient methods," *Wiley Encyclopedia of Operations Research and Management Science*, 2011.

[37] G. Zoutendijk, "Nonlinear programming, computational methods," *Integer and nonlinear programming*, pp. 37–86, 1970.

[38] M. Al-Baali, "Descent property and global convergence of the fletcher—reeves method with inexact line search," *IMA Journal of Numerical Analysis*, vol. 5, no. 1, pp. 121–124, 1985.

[39] M. J. D. Powell, "Restart procedures for the conjugate gradient method," *Mathematical programming*, vol. 12, no. 1, pp. 241–254, 1977.

[40] E. Polak and G. Ribiére, "Note sur la convergence de méthodes de directions conjuguées," *Revue française d'informatique et de recherche opérationnelle. Série rouge*, vol. 3, no. 16, pp. 35–43, 1969.

[41] B. T. Polyak, "The conjugate gradient method in extremal problems," *USSR Computational Mathematics and Mathematical Physics*, vol. 9, no. 4, pp. 94–112, 1969.

[42] M. J. Powell, "Nonconvex minimization calculations and the conjugate gradient method," in *Numerical analysis*, pp. 122–141, Springer, 1984.

[43] J. C. Gilbert and J. Nocedal, "Global convergence properties of conjugate gradient methods for optimization," *SIAM Journal on optimization*, vol. 2, no. 1, pp. 21–42, 1992.

[44] Y.-H. Dai and Q. Ni, "Testing different conjugate gradient methods for large-scale unconstrained optimization," *Journal of Computational Mathematics*, pp. 311–320, 2003.

[45] W. W. Hager and H. Zhang, "A survey of nonlinear conjugate gradient methods," *Pacific journal of Optimization*, vol. 2, no. 1, pp. 35–58, 2006.

[46] Y.-H. Dai and Y. Yuan, "Convergence properties of the conjugate descent method," *Advances in Mathematics*, p. 06, 1996.

[47] Y.-H. Dai and Y. Yuan, "A nonlinear conjugate gradient method with a strong global convergence property," *SIAM Journal on optimization*, vol. 10, no. 1, pp. 177–182, 1999.

[48] J. W. Daniel, "The conjugate gradient method for linear and nonlinear operator equations," *SIAM Journal on Numerical Analysis*, vol. 4, no. 1, pp. 10–26, 1967.

[49] D. Touati-Ahmed and C. Storey, "Efficient hybrid conjugate gradient techniques," *Journal of Optimization Theory and Applications*, vol. 64, no. 2, pp. 379–397, 1990.

[50] Y. Hu and C. Storey, "Global convergence result for conjugate gradient methods," *Journal of Optimization Theory and Applications*, vol. 71, no. 2, pp. 399–405, 1991.

[51] Y. Dai and Y. Yuan, "An efficient hybrid conjugate gradient method for unconstrained optimization," *Annals of Operations Research*, vol. 103, no. 1-4, pp. 33–47, 2001.

[52] A. Cohen, "Rate of convergence of several conjugate gradient algorithms," *SIAM Journal on Numerical Analysis*, vol. 9, no. 2, pp. 248–259, 1972.

[53] E. G. Birgin and J. M. Martínez, "A spectral conjugate gradient method for unconstrained optimization," *Applied Mathematics and optimization*, vol. 43, no. 2, pp. 117–128, 2001.

[54] A. Perry, “A modified conjugate gradient algorithm,” *Operations Research*, vol. 26, no. 6, pp. 1073–1078, 1978.

[55] Y. Narushima and H. Yabe, “A survey of sufficient descent conjugate gradient methods for unconstrained optimization,” *SUT journal of Mathematics*, vol. 50, no. 2, pp. 167–203, 2014.

[56] Y.-H. Dai and L.-Z. Liao, “New conjugacy conditions and related nonlinear conjugate gradient methods,” *Applied Mathematics and Optimization*, vol. 43, no. 1, pp. 87–101, 2001.

[57] N. I. Gould, D. Orban, and P. L. Toint, “Cutest: a constrained and unconstrained testing environment with safe threads for mathematical optimization,” *Computational Optimization and Applications*, vol. 60, no. 3, pp. 545–557, 2015.

[58] L. Lukšan, C. Matonoha, and J. Vlcek, “Modified cute problems for sparse unconstrained optimization,” tech. rep., 2010.

[59] E. D. Dolan and J. J. Moré, “Benchmarking optimization software with performance profiles,” *Mathematical programming*, vol. 91, no. 2, pp. 201–213, 2002.

[60] N. Andrei, “A hybrid conjugate gradient algorithm for unconstrained optimization as a convex combination of HS and DY,” *Studies in Informatics and Control*, vol. 17, no. 1, p. 57, 2008.

[61] S. S. Djordjević, “New hybrid conjugate gradient method as a convex combination of LS and CD methods,” *Filomat*, vol. 31, no. 6, pp. 1813–1825, 2017.

Appendix A

CUTEst environment

CUTEst[7] is a widely used testing environment for mathematical optimization. It contains a set of problems for constrained and unconstrained optimization. The problems are believed to reflect the qualities of many practical problems and are frequently used as the only benchmark to evaluate the performance of new methods in numerical optimization research.

For any given problem, CUTEst provides an interface obtain the initial guess point, evaluate the objective function at any given point and other, problem-dependent facilities, such as gradient, Hessian and constraint evaluation.

The CUTEst environment is implemented in the Fortran language. Testing problems are stored in the special format `.SIF` and can be decoded by the special utility `sifdecode`. Decoding procedure produces problem-dependent Fortran subroutines (e.g. for function evaluation, etc.) and data files that can be accessed from a Fortran interface. Hence, each new problem must first be decoded and compiled.

CUTEst environment can be used directly from the Fortran and also provides interfaces for C and MATLAB languages. In our implementation we used the Julia language wrapper that is discussed in Appendix B. It is much more convenient to setup and use the Julia interface than the mentioned interfaces provided by CUTEst.

[7] `https://github.com/ralna/CUTEst`

Appendix B

Implementation details

Julia language

The methods where implemented in Julia language[8], version 0.6.2. Julia is a modern language, aimed primarily on numerical computations. Julia aims to combine the expressiveness of scripting languages like Python or R with the speed of C or Fortran. For this reasons, the Julia language is the perfect choice for our purposes.

Line search algorithm

We used the Moré-Thuente line search algorithm and its implementation `MoreThuente` in the `LineSearches.jl`[9] package, version 6.0.1.

Random number generation and seeding

For random number generation, which is required in the probabilistic version of Algorithm 4.2 with weights update (4.5), we used the the `Distributions.jl`[10] package, version 0.15. Numbers are drawn from `Categorical` distribution that implements the discrete distribution that is required for (4.5).

In the tests, the result of the probabilistic method is a mean value of results of 10 independent iterations, where the internal random generator, `GLOBAL_RNG`, is initialized by calling `srand(i)`, where seed `i` is the iteration number. Hence, by repeating the same procedure on the same version of the package, one should be able to get exactly the same output, as the output of the pseudo-random generator is exactly defined by the seed value.

Interaction with CUTEst

The CUTEst environment was initially implemented in Fortran language. Its installation and setup is a highly nontrivial task itself, however, we used the wrapper for Julia, `CUTEst.jl`[11] package, version 0.3.3. The package does the installation and setup automatically and provides a convenient interface for accessing the testing environment.

[8] https://julialang.org/

[9] https://github.com/JuliaNLSolvers/LineSearches.jl

[10] https://github.com/JuliaStats/Distributions.jl

[11] https://github.com/JuliaSmoothOptimizers/CUTEst.jl

Appendix C

Contents of enclosed CD

readme.txt	file with instructions on launching the tests
src	the directory of source codes
impl	implementation and tests
thesis	source codes of the thesis
text	the thesis text directory
thesis.pdf	the thesis text in PDF format