最佳化方法與應用
MA5037-*

Chapter 2. Fundamentals of Unconstrained Optimization

§2.1 What Is a Solution?

§2.2 Overview of Algorithms

Introduction

In unconstrained optimization, we minimize an objective function that depends on real variables, with no restrictions at all on the values of these variables. The mathematical formulation is

$$\min_{x \in \mathbb{R}^n} f(x), \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a smooth function.

Usually, we lack a **global perspective** on the function f. All we know are the values of f and maybe some of its derivatives at a set of points x_0, x_1, x_2, $\cdots$. Fortunately, our algorithms get to choose these points, and they try to do so in a way that identifies a solution reliably and without using too much computer time or storage. Often, the information about f does not come cheaply, so we usually prefer algorithms that do not call for this information unnecessarily.

Introduction

In unconstrained optimization, we minimize an objective function that depends on real variables, with no restrictions at all on the values of these variables. The mathematical formulation is

$$\min_{x \in \mathbb{R}^n} f(x), \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a smooth function.

Usually, we lack a **global perspective** on the function f. All we know are the values of f and maybe some of its derivatives at a set of points $x_0, x_1, x_2, \cdots$. Fortunately, our algorithms get to choose these points, and they try to do so in a way that identifies a solution reliably and without using too much computer time or storage. Often, the information about f does not come cheaply, so we usually prefer algorithms that do not call for this information unnecessarily.

Introduction

In unconstrained optimization, we minimize an objective function that depends on real variables, with no restrictions at all on the values of these variables. The mathematical formulation is

$$\min_{x \in \mathbb{R}^n} f(x), \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a smooth function.

Usually, we lack a **global perspective** on the function f. All we know are the values of f and maybe some of its derivatives at a set of points x_0, x_1, x_2, $\cdots$. Fortunately, our algorithms get to choose these points, and they try to do so in a way that identifies a solution reliably and without using too much computer time or storage. Often, the information about f does not come cheaply, so we usually prefer algorithms that do not call for this information unnecessarily.

Introduction

In unconstrained optimization, we minimize an objective function that depends on real variables, with no restrictions at all on the values of these variables. The mathematical formulation is

$$\min_{x \in \mathbb{R}^n} f(x), \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a smooth function.

Usually, we lack a **global perspective** on the function f. All we know are the values of f and maybe some of its derivatives at a set of points x_0, x_1, x_2, $\cdots$. Fortunately, our algorithms get to choose these points, and they try to do so in a way that identifies a solution reliably and without using too much computer time or storage. Often, the information about f does not come cheaply, so we usually prefer algorithms that do not call for this information unnecessarily.

§2.1 What Is a Solution?

Generally, we would be happiest if we found a **global minimizer** of f, a point where the function attains its least value. A formal definition is

A point x_* is a global minimizer if $f(x_*) \leqslant f(x)$ for all $x \in \mathbb{R}^n$.

A global minimizer can be difficult to find, because our knowledge of f is usually only local. Since our algorithm does not visit many points (we hope!), we usually do not have a good picture of the overall shape of f, and we can never be sure that the function does not take a sharp dip in some region that has not been sampled by the algorithm.

§2.1 What Is a Solution?

Generally, we would be happiest if we found a **global minimizer** of f, a point where the function attains its least value. A formal definition is

A point x_* is a global minimizer if $f(x_*) \leqslant f(x)$ for all $x \in \mathbb{R}^n$.

A global minimizer can be difficult to find, because our knowledge of f is usually only local. Since our algorithm does not visit many points (we hope!), we usually do not have a good picture of the overall shape of f, and we can never be sure that the function does not take a sharp dip in some region that has not been sampled by the algorithm.

§2.1 What Is a Solution?

Most algorithms are able to find only a local minimizer, which is a point that achieves the smallest value of f in its neighborhood. Formally, we say:

> A point x_* is a local minimizer if there is a neighborhood $\mathcal{N}$ of x_* such that $f(x_*) \leqslant f(x)$ for all $x \in \mathcal{N}$.

Recall that a neighborhood of x_* is simply an open set that contains x_*. A point that satisfies this definition is sometimes called a **weak** local minimizer. By contrast,

> A point x_* is a **strict** local minimizer (also called a **strong** local minimizer) if there is a neighborhood $\mathcal{N}$ of x_* such that $f(x_*) < f(x)$ for all $x \in \mathcal{N}$ with $x \neq x_*$.

§2.1 What Is a Solution?

Most algorithms are able to find only a local minimizer, which is a point that achieves the smallest value of f in its neighborhood. Formally, we say:

> A point x_* is a local minimizer if there is a neighborhood $\mathcal{N}$ of x_* such that $f(x_*) \leqslant f(x)$ for all $x \in \mathcal{N}$.

Recall that a neighborhood of x_* is simply an open set that contains x_*. A point that satisfies this definition is sometimes called a **weak** local minimizer. By contrast,

> A point x_* is a **strict** local minimizer (also called a **strong** local minimizer) if there is a neighborhood $\mathcal{N}$ of x_* such that $f(x_*) < f(x)$ for all $x \in \mathcal{N}$ with $x \neq x_*$.

§2.1 What Is a Solution?

Most algorithms are able to find only a local minimizer, which is a point that achieves the smallest value of f in its neighborhood. Formally, we say:

> A point x_* is a local minimizer if there is a neighborhood $\mathcal{N}$ of x_* such that $f(x_*) \leqslant f(x)$ for all $x \in \mathcal{N}$.

Recall that a neighborhood of x_* is simply an open set that contains x_*. A point that satisfies this definition is sometimes called a **weak** local minimizer. By contrast,

> A point x_* is a **strict** local minimizer (also called a **strong** local minimizer) if there is a neighborhood $\mathcal{N}$ of x_* such that $f(x_*) < f(x)$ for all $x \in \mathcal{N}$ with $x \neq x_*$.

§2.1 What Is a Solution?

A slightly more exotic type of local minimizer is defined as follows.

> A point x_* is an **isolated** local minimizer if there is a neighborhood $\mathcal{N}$ of x_* such that x_* is the only local minimizer in $\mathcal{N}$.

While strict local minimizers are not always isolated, it is true that all isolated local minimizers are strict.

Sometimes we have additional "global" knowledge about f that may help in identifying global minima. An important special case is that of convex functions, for which every local minimizer is also a global minimizer.

§2.1 What Is a Solution?

A slightly more exotic type of local minimizer is defined as follows.

> A point x_* is an **isolated** local minimizer if there is a neighborhood $\mathcal{N}$ of x_* such that x_* is the only local minimizer in $\mathcal{N}$.

While strict local minimizers are not always isolated, it is true that all isolated local minimizers are strict.

Sometimes we have additional "global" knowledge about f that may help in identifying global minima. An important special case is that of convex functions, for which every local minimizer is also a global minimizer.

§2.1 What Is a Solution?

From the definitions given above, it might seem that the only way to find out whether a point x_* is a local minimum is to examine all the points in its immediate vicinity, to make sure that none of them has a smaller function value. When the function f is smooth; however, there are more efficient and practical ways to identify local minima. In particular, if f is twice continuously differentiable, we may be able to tell that x_* is a local minimizer (and possibly a strict local minimizer) by examining just the gradient $\nabla f(x_*)$ and the Hessian $\nabla^2 f(x_*)$.

§2.1 What Is a Solution?

From the definitions given above, it might seem that the only way to find out whether a point x_* is a local minimum is to examine all the points in its immediate vicinity, to make sure that none of them has a smaller function value. When the function f is smooth; however, there are more efficient and practical ways to identify local minima. In particular, if f is twice continuously differentiable, we may be able to tell that x_* is a local minimizer (and possibly a strict local minimizer) by examining just the gradient $\nabla f(x_*)$ and the Hessian $\nabla^2 f(x_*)$.

§2.1 What Is a Solution?

The mathematical tool used to study minimizers of smooth functions is Taylor's theorem.

Theorem (Taylor)

Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ *is continuously differentiable and that* $p \in \mathbb{R}^n$. *Then we have that*

$$f(x+p) = f(x) + \nabla f(x+tp)^{\mathrm{T}} p$$

for some $t \in (0,1)$. *Moreover, if* f *is twice continuously differentiable, we have that*

$$\nabla f(x+p) = \nabla f(x) + \int_0^1 \nabla^2 f(x+tp) p \, dt$$

and that

$$f(x+p) = f(x) + \nabla f(x)^{\mathrm{T}} p + \frac{1}{2} p^{\mathrm{T}} \nabla^2 f(x+tp) p,$$

for some $t \in (0,1)$.

§2.1 What Is a Solution?

Necessary conditions for optimality are derived by assuming that x_* is a local minimizer and then proving facts about $\nabla f(x_*)$ and $\nabla^2 f(x_*)$.

Theorem

If x_* is a local minimizer and f is continuously differentiable in an open neighborhood of x_*, then $\nabla f(x_*) = 0$.

We call x_* a **stationary point** if $\nabla f(x_*) = 0$. According to the theorem above, any local minimizer must be a stationary point.

§2.1 What Is a Solution?

Recall that a matrix B is positive definite if $p^{\mathrm{T}}Bp > 0$ for all $p \neq 0$, and positive semi-definite if $p^{\mathrm{T}}Bp \geqslant 0$ for all $p \in \mathbb{R}^n$.

Theorem (Second-Order Necessary Conditions)

If x_* is a local minimizer of f and $\nabla^2 f$ exists and is continuous in an open neighborhood of x_*, then $\nabla f(x_*) = 0$ and $\nabla^2 f(x_*)$ is positive semi-definite.

We now describe sufficient conditions on the derivatives of f at the point z^* that guarantee that x_* is a local minimizer.

Theorem (Second-Order Sufficient Conditions)

Suppose that $\nabla^2 f$ is continuous in an open neighborhood of x_* and that $\nabla f(x_*) = 0$ and $\nabla^2 f(x_*)$ is positive definite. Then x_* is a ***strict*** *local minimizer of f.*

§2.1 What Is a Solution?

Recall that a matrix B is positive definite if $p^{\mathrm{T}}Bp > 0$ for all $p \neq 0$, and positive semi-definite if $p^{\mathrm{T}}Bp \geqslant 0$ for all $p \in \mathbb{R}^n$.

Theorem (Second-Order Necessary Conditions)

If x_* is a local minimizer of f and $\nabla^2 f$ exists and is continuous in an open neighborhood of x_*, then $\nabla f(x_*) = 0$ and $\nabla^2 f(x_*)$ is positive semi-definite.

We now describe sufficient conditions on the derivatives of f at the point z^* that guarantee that x_* is a local minimizer.

Theorem (Second-Order Sufficient Conditions)

Suppose that $\nabla^2 f$ is continuous in an open neighborhood of x_* and that $\nabla f(x_*) = 0$ and $\nabla^2 f(x_*)$ is positive definite. Then x_* is a ***strict*** *local minimizer of f.*

§2.1 What Is a Solution?

Note that the second-order sufficient conditions of the theorem in the previous page guarantee something **stronger** than the necessary conditions discussed earlier; namely, that the minimizer is a strict local minimizer. Note too that the second-order sufficient conditions are not necessary: A point x_* may be a strict local minimizer, and yet may fail to satisfy the sufficient conditions. A simple example is given by the function $f(x) = x^4$, for which the point $x_* = 0$ is a strict local minimizer at which the Hessian matrix vanishes (and is therefore not positive definite).

§2.1 What Is a Solution?

Note that the second-order sufficient conditions of the theorem in the previous page guarantee something **stronger** than the necessary conditions discussed earlier; namely, that the minimizer is a strict local minimizer. Note too that the second-order sufficient conditions are not necessary: A point x_* may be a strict local minimizer, and yet may fail to satisfy the sufficient conditions. A simple example is given by the function $f(x) = x^4$, for which the point $x_* = 0$ is a strict local minimizer at which the Hessian matrix vanishes (and is therefore not positive definite).

§2.1 What Is a Solution?

When the objective function is convex, local and global minimizers are simple to characterize.

Theorem

When f is convex, any local minimizer x_* is a global minimizer of f. If in addition f is differentiable, then any stationary point x_* is a global minimizer of f.

Proof.

Suppose that x_* is a local but not a global minimizer. Then there exists a point $z \in \mathbb{R}^n$ with $f(z) < f(x_*)$. For a given neighborhood $\mathcal{N}$ of x_*, let $\lambda \in (0,1)$ be such that $x = \lambda z + (1-\lambda)x_* \in \mathcal{N}$ (such a λ must exist). Then the convexity property for f implies that

$$f(x) \leqslant \lambda f(z) + (1-\lambda) f(x_*) < f(x_*).$$

Hence, x_* is not a local minimizer. □

§2.1 What Is a Solution?

When the objective function is convex, local and global minimizers are simple to characterize.

Theorem

When f is convex, any local minimizer x_* is a global minimizer of f. If in addition f is differentiable, then any stationary point x_* is a global minimizer of f.

Proof.

Suppose that x_* is a local but not a global minimizer. Then there exists a point $z \in \mathbb{R}^n$ with $f(z) < f(x_*)$. For a given neighborhood $\mathcal{N}$ of x_*, let $\lambda \in (0,1)$ be such that $x = \lambda z + (1-\lambda)x_* \in \mathcal{N}$ (such a λ must exist). Then the convexity property for f implies that

$$f(x) \leqslant \lambda f(z) + (1-\lambda) f(x_*) < f(x_*).$$

Hence, x_* is not a local minimizer. □

§2.1 What Is a Solution?

When the objective function is convex, local and global minimizers are simple to characterize.

Theorem

When f is convex, any local minimizer x_* is a global minimizer of f. If in addition f is differentiable, then any stationary point x_* is a global minimizer of f.

Proof.

Suppose that x_* is a local but not a global minimizer. Then there exists a point $z \in \mathbb{R}^n$ with $f(z) < f(x_*)$. For a given neighborhood $\mathcal{N}$ of x_*, let $\lambda \in (0,1)$ be such that $x = \lambda z + (1-\lambda)x_* \in \mathcal{N}$ (such a λ must exist). Then the convexity property for f implies that

$$f(x) \leqslant \lambda f(z) + (1-\lambda) f(x_*) < f(x_*).$$

Hence, x_* is not a local minimizer. □

§2.1 What Is a Solution?

When the objective function is convex, local and global minimizers are simple to characterize.

Theorem

When f is convex, any local minimizer x_* is a global minimizer of f. If in addition f is differentiable, then any stationary point x_* is a global minimizer of f.

Proof.

Suppose that x_* is a local but not a global minimizer. Then there exists a point $z \in \mathbb{R}^n$ with $f(z) < f(x_*)$. For a given neighborhood $\mathcal{N}$ of x_*, let $\lambda \in (0,1)$ be such that $x = \lambda z + (1-\lambda)x_* \in \mathcal{N}$ (such a λ must exist). Then the convexity property for f implies that

$$f(x) \leqslant \lambda f(z) + (1-\lambda) f(x_*) < f(x_*).$$

Hence, x_* is not a local minimizer. □

§2.1 What Is a Solution?

Proof (cont'd).

For the second part of the theorem, suppose that x_* is not a global minimizer and choose z as above. Then, from convexity, we have

$$\begin{aligned}\nabla f(x_*)^{\mathrm{T}}(z - x_*) &= \frac{d}{d\lambda}\Big|_{\lambda=0} f\big(\lambda z + (1-\lambda)x_*\big) \\ &= \lim_{\lambda \to 0^+} \frac{f\big(\lambda z + (1-\lambda)x_*\big) - f(x_*)}{\lambda} \\ &\leqslant \lim_{\lambda \to 0^+} \frac{\lambda f(z) + (1-\lambda) f(x_*) - f(x_*)}{\lambda} \\ &= f(z) - f(x_*) < 0\,.\end{aligned}$$

Therefore, $\nabla f(x_*) \neq 0$, and so x_* is not a stationary point. □

These results, which are based on elementary calculus, provide the foundations for unconstrained optimization algorithms. In one way or another, all algorithms seek a point where $\nabla f(\cdot)$ vanishes.

§2.1 What Is a Solution?

Proof (cont'd).

For the second part of the theorem, suppose that x_* is not a global minimizer and choose z as above. Then, from convexity, we have

$$\begin{aligned}\nabla f(x_*)^{\mathrm{T}}(z - x_*) &= \frac{d}{d\lambda}\Big|_{\lambda=0} f\big(\lambda z + (1-\lambda)x_*\big) \\ &= \lim_{\lambda \to 0^+} \frac{f\big(\lambda z + (1-\lambda)x_*\big) - f(x_*)}{\lambda} \\ &\leqslant \lim_{\lambda \to 0^+} \frac{\lambda f(z) + (1-\lambda) f(x_*) - f(x_*)}{\lambda} \\ &= f(z) - f(x_*) < 0\,.\end{aligned}$$

Therefore, $\nabla f(x_*) \neq 0$, and so x_* is not a stationary point. □

These results, which are based on elementary calculus, provide the foundations for unconstrained optimization algorithms. In one way or another, all algorithms seek a point where $\nabla f(\cdot)$ vanishes.

§2.1 What Is a Solution?

Proof (cont'd).

For the second part of the theorem, suppose that x_* is not a global minimizer and choose z as above. Then, from convexity, we have

$$\begin{aligned}\nabla f(x_*)^{\mathrm{T}}(z - x_*) &= \frac{d}{d\lambda}\Big|_{\lambda=0} f\big(\lambda z + (1-\lambda)x_*\big) \\ &= \lim_{\lambda \to 0^+} \frac{f\big(\lambda z + (1-\lambda)x_*\big) - f(x_*)}{\lambda} \\ &\leqslant \lim_{\lambda \to 0^+} \frac{\lambda f(z) + (1-\lambda) f(x_*) - f(x_*)}{\lambda} \\ &= f(z) - f(x_*) < 0\,.\end{aligned}$$

Therefore, $\nabla f(x_*) \neq 0$, and so x_* is not a stationary point. □

These results, which are based on elementary calculus, provide the foundations for unconstrained optimization algorithms. In one way or another, all algorithms seek a point where $\nabla f(\cdot)$ vanishes.

§2.1 What Is a Solution?

Proof (cont'd).

For the second part of the theorem, suppose that x_* is not a global minimizer and choose z as above. Then, from convexity, we have

$$\begin{aligned}\nabla f(x_*)^{\mathrm{T}}(z - x_*) &= \frac{d}{d\lambda}\Big|_{\lambda=0} f\big(\lambda z + (1-\lambda)x_*\big) \\ &= \lim_{\lambda \to 0^+} \frac{f\big(\lambda z + (1-\lambda)x_*\big) - f(x_*)}{\lambda} \\ &\leqslant \lim_{\lambda \to 0^+} \frac{\lambda f(z) + (1-\lambda) f(x_*) - f(x_*)}{\lambda} \\ &= f(z) - f(x_*) < 0\,.\end{aligned}$$

Therefore, $\nabla f(x_*) \neq 0$, and so x_* is not a stationary point. □

These results, which are based on elementary calculus, provide the foundations for unconstrained optimization algorithms. In one way or another, all algorithms seek a point where $\nabla f(\cdot)$ vanishes.

§2.1 What Is a Solution?

Proof (cont'd).

For the second part of the theorem, suppose that x_* is not a global minimizer and choose z as above. Then, from convexity, we have

$$\begin{aligned}\nabla f(x_*)^{\mathrm{T}}(z - x_*) &= \frac{d}{d\lambda}\Big|_{\lambda=0} f\big(\lambda z + (1-\lambda)x_*\big) \\ &= \lim_{\lambda \to 0^+} \frac{f\big(\lambda z + (1-\lambda)x_*\big) - f(x_*)}{\lambda} \\ &\leqslant \lim_{\lambda \to 0^+} \frac{\lambda f(z) + (1-\lambda) f(x_*) - f(x_*)}{\lambda} \\ &= f(z) - f(x_*) < 0\,.\end{aligned}$$

Therefore, $\nabla f(x_*) \neq 0$, and so x_* is not a stationary point. □

These results, which are based on elementary calculus, provide the foundations for unconstrained optimization algorithms. In one way or another, all algorithms seek a point where $\nabla f(\cdot)$ vanishes.

§2.1 What Is a Solution?

Proof (cont'd).

For the second part of the theorem, suppose that x_* is not a global minimizer and choose z as above. Then, from convexity, we have

$$\begin{aligned}\nabla f(x_*)^{\mathrm{T}}(z - x_*) &= \frac{d}{d\lambda}\Big|_{\lambda=0} f\big(\lambda z + (1-\lambda)x_*\big) \\ &= \lim_{\lambda \to 0^+} \frac{f\big(\lambda z + (1-\lambda)x_*\big) - f(x_*)}{\lambda} \\ &\leqslant \lim_{\lambda \to 0^+} \frac{\lambda f(z) + (1-\lambda) f(x_*) - f(x_*)}{\lambda} \\ &= f(z) - f(x_*) < 0\,.\end{aligned}$$

Therefore, $\nabla f(x_*) \neq 0$, and so x_* is not a stationary point. □

These results, which are based on elementary calculus, provide the foundations for unconstrained optimization algorithms. In one way or another, all algorithms seek a point where $\nabla f(\cdot)$ vanishes.

§2.1 What Is a Solution?

- **Non-smooth Problems**

Our lecture focuses on smooth functions, by which we generally mean functions whose second derivatives exist and are continuous. We note, however, that there are interesting problems in which the functions involved may be non-smooth and even discontinuous. It is not possible **in general** to identify a minimizer of a general discontinuous function. If, however, the function consists of a few smooth pieces, with discontinuities between the pieces, it may be possible to find the minimizer by minimizing each smooth piece individually.

§2.1 What Is a Solution?

- **Non-smooth Problems**

Our lecture focuses on smooth functions, by which we generally mean functions whose second derivatives exist and are continuous. We note, however, that there are interesting problems in which the functions involved may be non-smooth and even discontinuous. It is not possible **in general** to identify a minimizer of a general discontinuous function. If, however, the function consists of a few smooth pieces, with discontinuities between the pieces, it may be possible to find the minimizer by minimizing each smooth piece individually.

§2.1 What Is a Solution?

If the function is continuous everywhere but non-differentiable at certain points, as in Figure 1, we can identify a solution by examing the ***sub-gradient*** or ***generalized gradient***, which are generalizations of the concept of gradient to the non-smooth case. Non-smooth optimization is beyond the scope of our lecture.

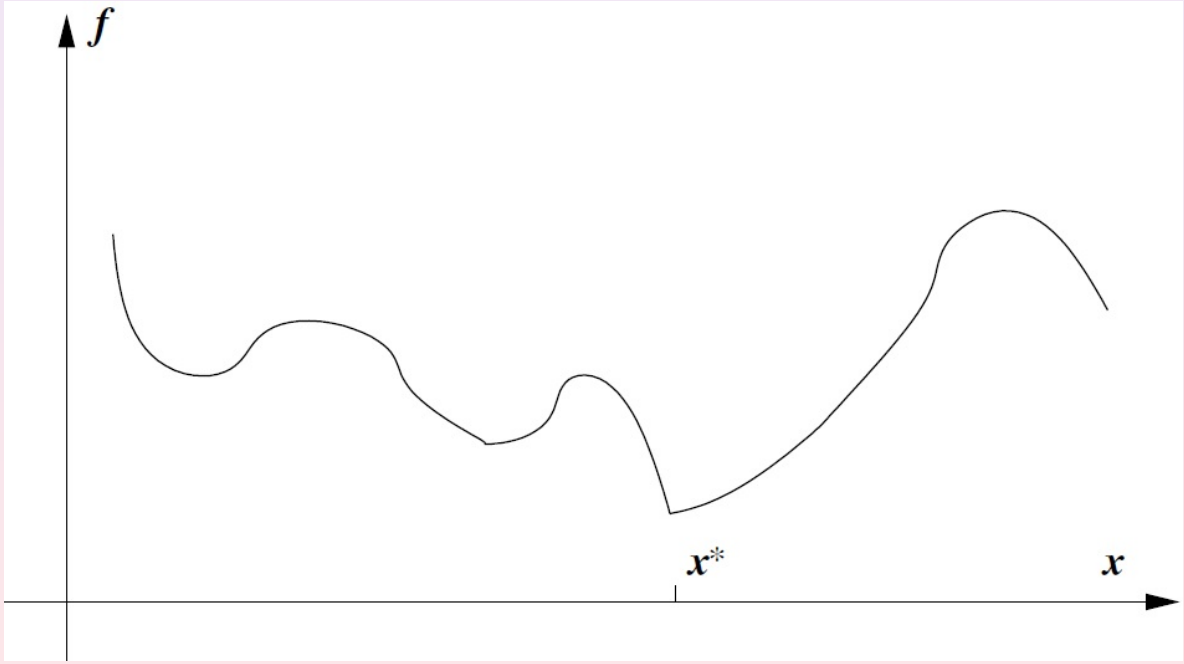

Figure 1: Non-smooth function with minimum at a kink

§2.2 Overview of Algorithms

- **Unconstrained Optimization**:

All algorithms for unconstrained minimization require the user to supply a starting point, which we usually denote by x_0. Beginning at x_0, optimization algorithms generate a sequence of iterates $\{x_k\}_{k=1}^{\infty}$ that terminate when either no more progress can be made or when it seems that a solution point has been approximated with sufficient accuracy. In deciding how to move from one iterate x_k to the next, the algorithms use information about the function f at x_k, and possibly also information from earlier iterates x_0, x_1, $\cdots$, x_{k-1}. They use this information to find a new iterate x_{k+1} with a lower function value than x_k. There exist non-monotone algorithms that do not insist on a decrease in f at every step, but even these algorithms require f to be decreased after some prescribed number m of iterations; that is, $f(x_k) < f(x_{k-m})$.

§2.2 Overview of Algorithms

- **Unconstrained Optimization**:

All algorithms for unconstrained minimization require the user to supply a starting point, which we usually denote by x_0. Beginning at x_0, optimization algorithms generate a sequence of iterates $\{x_k\}_{k=1}^{\infty}$ that terminate when either no more progress can be made or when it seems that a solution point has been approximated with sufficient accuracy. In deciding how to move from one iterate x_k to the next, the algorithms use information about the function f at x_k, and possibly also information from earlier iterates x_0, x_1, $\cdots$, x_{k-1}. They use this information to find a new iterate x_{k+1} with a lower function value than x_k. There exist non-monotone algorithms that do not insist on a decrease in f at every step, but even these algorithms require f to be decreased after some prescribed number m of iterations; that is, $f(x_k) < f(x_{k-m})$.

§2.2 Overview of Algorithms

- **Unconstrained Optimization**:

All algorithms for unconstrained minimization require the user to supply a starting point, which we usually denote by x_0. Beginning at x_0, optimization algorithms generate a sequence of iterates $\{x_k\}_{k=1}^{\infty}$ that terminate when either no more progress can be made or when it seems that a solution point has been approximated with sufficient accuracy. In deciding how to move from one iterate x_k to the next, the algorithms use information about the function f at x_k, and possibly also information from earlier iterates x_0, x_1, $\cdots$, x_{k-1}. They use this information to find a new iterate x_{k+1} with a lower function value than x_k. There exist non-monotone algorithms that do not insist on a decrease in f at every step, but even these algorithms require f to be decreased after some prescribed number m of iterations; that is, $f(x_k) < f(x_{k-m})$.

§2.2 Overview of Algorithms

- **Two Strategies: Line Search and Trust Region**:

In the **line search** strategy, the algorithm chooses a direction p_k and searches along this direction from the current iterate x_k for a new iterate with a lower function value. The distance to move along p_k can be found by approximately solving the following one-dimensional minimization problem to find a step length α:

$$\min_{\alpha>0} f(x_k + \alpha p_k) . \tag{2}$$

By solving (2) exactly, we would derive the maximum benefit from the direction p_k, but an exact minimization may be expensive and is usually unnecessary. Instead, the line search algorithm generates a limited number of trial step lengths until it finds one that loosely approximates the minimum of (2). At the new point, a new search direction and step length are computed, and the process is repeated.

§2.2 Overview of Algorithms

- **Two Strategies: Line Search and Trust Region**:

In the **line search** strategy, the algorithm chooses a direction p_k and searches along this direction from the current iterate x_k for a new iterate with a lower function value. The distance to move along p_k can be found by approximately solving the following one-dimensional minimization problem to find a step length α:

$$\min_{\alpha>0} f(x_k + \alpha p_k). \tag{2}$$

By solving (2) exactly, we would derive the maximum benefit from the direction p_k, but an exact minimization may be expensive and is usually unnecessary. Instead, the line search algorithm generates a limited number of trial step lengths until it finds one that loosely approximates the minimum of (2). At the new point, a new search direction and step length are computed, and the process is repeated.

§2.2 Overview of Algorithms

- **Two Strategies: Line Search and Trust Region**:

In the **line search** strategy, the algorithm chooses a direction p_k and searches along this direction from the current iterate x_k for a new iterate with a lower function value. The distance to move along p_k can be found by approximately solving the following one-dimensional minimization problem to find a step length α:

$$\min_{\alpha>0} f(x_k + \alpha p_k). \tag{2}$$

By solving (2) exactly, we would derive the maximum benefit from the direction p_k, but an exact minimization may be expensive and is usually unnecessary. Instead, the line search algorithm generates a limited number of trial step lengths until it finds one that loosely approximates the minimum of (2). At the new point, a new search direction and step length are computed, and the process is repeated.

§2.2 Overview of Algorithms

- **Two Strategies: Line Search and Trust Region**:

In the **line search** strategy, the algorithm chooses a direction p_k and searches along this direction from the current iterate x_k for a new iterate with a lower function value. The distance to move along p_k can be found by approximately solving the following one-dimensional minimization problem to find a step length α:

$$\min_{\alpha>0} f(x_k + \alpha p_k). \tag{2}$$

By solving (2) exactly, we would derive the maximum benefit from the direction p_k, but an exact minimization may be expensive and is usually unnecessary. Instead, the line search algorithm generates a limited number of trial step lengths until it finds one that loosely approximates the minimum of (2). At the new point, a new search direction and step length are computed, and the process is repeated.

§2.2 Overview of Algorithms

In the second algorithmic strategy, known as **trust region**, the information gathered about f is used to construct a **model function** m_k whose behavior near the current point x_k is similar to that of the actual objective function f. Because the model m_k may not be a good approximation of f when x is far from x_k, we restrict the search for a minimizer of m_k to some region around x_k. In other words, we find the candidate step p by approximately solving the following sub-problem:

$$\min_{p \in \mathbb{R}^n} m_k(x_k + p)\,, \text{where } x_k + p \text{ lies inside the trust region.} \qquad (3)$$

If the candidate solution does not produce a sufficient decrease in f, we conclude that the trust region is too large, and we shrink it and re-solve (3). Usually, the trust region is a ball defined by $\|p\| \leqslant \Delta$, where the scalar $\Delta > 0$ is called the trust-region radius. Elliptical and box-shaped trust regions may also be used.

§2.2 Overview of Algorithms

In the second algorithmic strategy, known as **trust region**, the information gathered about f is used to construct a **model function** m_k whose behavior near the current point x_k is similar to that of the actual objective function f. Because the model m_k may not be a good approximation of f when x is far from x_k, we restrict the search for a minimizer of m_k to some region around x_k. In other words, we find the candidate step p by approximately solving the following sub-problem:

$$\min_{p \in \mathbb{R}^n} m_k(x_k + p)\,, \text{where } x_k + p \text{ lies inside the trust region.} \quad (3)$$

If the candidate solution does not produce a sufficient decrease in f, we conclude that the trust region is too large, and we shrink it and re-solve (3). Usually, the trust region is a ball defined by $\|p\| \leqslant \Delta$, where the scalar $\Delta > 0$ is called the trust-region radius. Elliptical and box-shaped trust regions may also be used.

§2.2 Overview of Algorithms

In the second algorithmic strategy, known as **trust region**, the information gathered about f is used to construct a **model function** m_k whose behavior near the current point x_k is similar to that of the actual objective function f. Because the model m_k may not be a good approximation of f when x is far from x_k, we restrict the search for a minimizer of m_k to some region around x_k. In other words, we find the candidate step p by approximately solving the following sub-problem:

$$\min_{p \in \mathbb{R}^n} m_k(x_k + p)\,, \text{where } x_k + p \text{ lies inside the trust region.} \qquad (3)$$

If the candidate solution does not produce a sufficient decrease in f, we conclude that the trust region is too large, and we shrink it and re-solve (3). Usually, the trust region is a ball defined by $\|p\| \leqslant \Delta$, where the scalar $\Delta > 0$ is called the trust-region radius. Elliptical and box-shaped trust regions may also be used.

§2.2 Overview of Algorithms

The model m_k in (3) is usually defined to be a quadratic function of the form

$$m_k(x_k + p) = f_k + p^{\mathrm{T}} \nabla f_k + \frac{1}{2} p^{\mathrm{T}} B_k p \,, \tag{4}$$

where f_k, ∇f_k, and B_k are a scalar, vector, and matrix, respectively. As the notation indicates, f_k and ∇f_k are chosen to be the function and gradient values at the point x_k, so that m_k and f are in agreement to first order at the current iterate x_k. The matrix B_k is either the Hessian $\nabla^2 f_k$ or some approximation to it.

§2.2 Overview of Algorithms

The model m_k in (3) is usually defined to be a quadratic function of the form

$$m_k(x_k + p) = f_k + p^{\mathrm{T}} \nabla f_k + \frac{1}{2} p^{\mathrm{T}} B_k p \,, \tag{4}$$

where f_k, ∇f_k, and B_k are a scalar, vector, and matrix, respectively. As the notation indicates, f_k and ∇f_k are chosen to be the function and gradient values at the point x_k, so that m_k and f are in agreement to first order at the current iterate x_k. The matrix B_k is either the Hessian $\nabla^2 f_k$ or some approximation to it.

§2.2 Overview of Algorithms

In a sense, the line search and trust-region approaches differ in the order in which they choose the direction and distance of the move to the next iterate. Line search starts by fixing the direction p_k and then identifying an appropriate distance, namely the step length α_k. In trust region, we first choose a maximum distance – the trust-region radius Δ_k – and then seek a direction and step that attain the best improvement possible subject to this distance constraint. If this step proves to be unsatisfactory, we reduce the distance measure Δ_k and try again.

§2.2 Overview of Algorithms

•• **Search Direction for Line Search Methods**:

The steepest descent direction $-\nabla f_k$ is the most obvious choice for search direction for a line search method. It is intuitive; among all the directions we could move from x_k, it is the one along which f **decreases most rapidly**. The steepest descent method is a line search method that moves along $p_k = -\nabla f_k$ at every step. It can choose the step length α_k in a variety of ways, as we discuss in Chapter 3. One advantage of the steepest descent direction is that it requires calculation of the gradient ∇f_k but not of second derivatives. However, it can be excruciatingly slow on difficult problems.

§2.2 Overview of Algorithms

•• **Search Direction for Line Search Methods**:

The steepest descent direction $-\nabla f_k$ is the most obvious choice for search direction for a line search method. It is intuitive; among all the directions we could move from x_k, it is the one along which f **decreases most rapidly**. The steepest descent method is a line search method that moves along $p_k = -\nabla f_k$ at every step. It can choose the step length α_k in a variety of ways, as we discuss in Chapter 3. One advantage of the steepest descent direction is that it requires calculation of the gradient ∇f_k but not of second derivatives. However, it can be excruciatingly slow on difficult problems.

§2.2 Overview of Algorithms

Line search methods may use search directions other than the steepest descent direction. In general, any descent direction – one that makes an angle of strictly less than $\pi/2$ radians with $-\nabla f_k$ – is guaranteed to produce a decrease in f, provided that the step length is sufficiently small.

Another important search direction – perhaps the most important one of all – is the Newton direction. This direction is derived from the second-order Taylor series approximation to $f(x_k + p)$, which is

$$f(x_k + p) \approx f_k + p^{\mathrm{T}} \nabla f_k + \frac{1}{2} p^{\mathrm{T}} \nabla^2 f_k p \equiv m_k(p)\,. \tag{5}$$

Assuming for the moment that $\nabla^2 f_k$ is positive definite, we obtain the Newton direction by finding the vector $p = p_k^{\mathrm{N}}$ that minimizes $m_k(p)$.

§2.2 Overview of Algorithms

Line search methods may use search directions other than the steepest descent direction. In general, any descent direction – one that makes an angle of strictly less than $\pi/2$ radians with $-\nabla f_k$ – is guaranteed to produce a decrease in f, provided that the step length is sufficiently small.

Another important search direction – perhaps the most important one of all – is the Newton direction. This direction is derived from the second-order Taylor series approximation to $f(x_k + p)$, which is

$$f(x_k + p) \approx f_k + p^{\mathrm{T}} \nabla f_k + \frac{1}{2} p^{\mathrm{T}} \nabla^2 f_k p \equiv m_k(p)\,. \tag{5}$$

Assuming for the moment that $\nabla^2 f_k$ is positive definite, we obtain the Newton direction by finding the vector $p = p_k^{\mathrm{N}}$ that minimizes $m_k(p)$.

§2.2 Overview of Algorithms

By simply setting the derivative of $m_k(p)$ to zero, we find that

$$p_k^{\mathrm{N}} = -(\nabla^2 f_k)^{-1} \nabla f_k \,. \tag{6}$$

The Newton direction p_k^{N} is reliable when the difference between the true function $f(x_k + p)$ and its quadratic model $m_k(p)$ is not too large. By comparing (5) with the exact identity

$$f(x+p) = f(x) + \nabla f(x)^{\mathrm{T}} p + \frac{1}{2} p^{\mathrm{T}} \nabla^2 f(x+tp) p \,,$$

we see that the only difference between these functions is that the matrix $\nabla^2 f(x_k + tp)$ in the third term of the expansion has been replaced by $\nabla^2 f_k$. If $\nabla^2 f$ is sufficiently smooth, this difference introduces a perturbation of only $\mathcal{O}(\|p\|^3)$ into the expansion, so that when $\|p\|$ is small, the approximation $f(x_k + p) \approx m_k(p)$ is quite accurate.

§2.2 Overview of Algorithms

By simply setting the derivative of $m_k(p)$ to zero, we find that

$$p_k^{\mathrm{N}} = -(\nabla^2 f_k)^{-1} \nabla f_k \,. \tag{6}$$

The Newton direction p_k^{N} is reliable when the difference between the true function $f(x_k + p)$ and its quadratic model $m_k(p)$ is not too large. By comparing (5) with the exact identity

$$f(x+p) = f(x) + \nabla f(x)^{\mathrm{T}} p + \frac{1}{2} p^{\mathrm{T}} \nabla^2 f(x+tp) p \,,$$

we see that the only difference between these functions is that the matrix $\nabla^2 f(x_k + tp)$ in the third term of the expansion has been replaced by $\nabla^2 f_k$. If $\nabla^2 f$ is sufficiently smooth, this difference introduces a perturbation of only $\mathcal{O}(\|p\|^3)$ into the expansion, so that when $\|p\|$ is small, the approximation $f(x_k + p) \approx m_k(p)$ is quite accurate.

§2.2 Overview of Algorithms

The Newton direction p_k^{N} can be used in a line search method when $\nabla^2 f_k$ is positive definite, for in this case we have

$$\nabla f_k^{\mathrm{T}} p_k^{\mathrm{N}} = -(p_k^{\mathrm{N}})^{\mathrm{T}} \nabla^2 f_k p_k^{\mathrm{N}} \leqslant -\sigma_k \|p_k^{\mathrm{N}}\|^2$$

for some $\sigma_k > 0$ (the minimum eigenvalue of $\nabla^2 f_k$). Unless the gradient ∇f_k (and therefore the step p_k^{N}) is zero, we have that $\nabla f_k^{\mathrm{T}} p_k^{\mathrm{N}} < 0$, so the Newton direction is a descent direction. Unlike the steepest descent direction, there is a "natural" step length of 1 associated with the Newton direction. Most line search implementations of Newton's method use the unit step $\alpha = 1$, where possible and adjust α only when it does not produce a satisfactory reduction in the value of f.

§2.2 Overview of Algorithms

The Newton direction p_k^{N} can be used in a line search method when $\nabla^2 f_k$ is positive definite, for in this case we have

$$\nabla f_k^{\mathrm{T}} p_k^{\mathrm{N}} = -(p_k^{\mathrm{N}})^{\mathrm{T}} \nabla^2 f_k p_k^{\mathrm{N}} \leqslant -\sigma_k \|p_k^{\mathrm{N}}\|^2$$

for some $\sigma_k > 0$ (the minimum eigenvalue of $\nabla^2 f_k$). Unless the gradient ∇f_k (and therefore the step p_k^{N}) is zero, we have that $\nabla f_k^{\mathrm{T}} p_k^{\mathrm{N}} < 0$, so the Newton direction is a descent direction. Unlike the steepest descent direction, there is a "natural" step length of 1 associated with the Newton direction. Most line search implementations of Newton's method use the unit step $\alpha = 1$, where possible and adjust α only when it does not produce a satisfactory reduction in the value of f.

§2.2 Overview of Algorithms

The Newton direction p_k^{N} can be used in a line search method when $\nabla^2 f_k$ is positive definite, for in this case we have

$$\nabla f_k^{\mathrm{T}} p_k^{\mathrm{N}} = -(p_k^{\mathrm{N}})^{\mathrm{T}} \nabla^2 f_k p_k^{\mathrm{N}} \leqslant -\sigma_k \|p_k^{\mathrm{N}}\|^2$$

for some $\sigma_k > 0$ (the minimum eigenvalue of $\nabla^2 f_k$). Unless the gradient ∇f_k (and therefore the step p_k^{N}) is zero, we have that $\nabla f_k^{\mathrm{T}} p_k^{\mathrm{N}} < 0$, so the Newton direction is a descent direction. Unlike the steepest descent direction, there is a "natural" step length of 1 associated with the Newton direction. Most line search implementations of Newton's method use the unit step $\alpha = 1$, where possible and adjust α only when it does not produce a satisfactory reduction in the value of f.

§2.2 Overview of Algorithms

When $\nabla^2 f_k$ is not positive definite, the Newton direction may not even be defined, since $(\nabla^2 f_k)^{-1}$ may not exist. Even when it is defined, it may not satisfy the descent property $\nabla f_k^{\mathrm{T}} p_k^{\mathrm{N}} < 0$, in which case it is unsuitable as a search direction. In these situations, line search methods modify the definition of p_k to make it satisfy the descent condition while retaining the benefit of the second-order information contained in $\nabla^2 f_k$. We describe these modifications in Chapter 3.

§2.2 Overview of Algorithms

Methods that use the Newton direction have a fast rate of local convergence, typically **quadratic**. After a neighborhood of the solution is reached, convergence to high accuracy often occurs in just a few iterations. The main drawback of the Newton direction is the need for the Hessian $\nabla^2 f(x)$. Explicit computation of this matrix of second derivatives can sometimes be a cumbersome, error-prone, and expensive process. Finite-difference and automatic differentiation techniques described in Chapter 8 may be useful in avoiding the need to calculate second derivatives by hand.

Definition

A sequence $\{x_k\}_{k=1}^{\infty}$ is said to converge to x_* **quadratically** if there exists a constant $M > 0$ such that

$$\frac{\|x_{k+1} - x_*\|}{\|x_k - x_*\|^2} \leqslant M \qquad \forall\, k \gg 1\,.$$

§2.2 Overview of Algorithms

Methods that use the Newton direction have a fast rate of local convergence, typically **quadratic**. After a neighborhood of the solution is reached, convergence to high accuracy often occurs in just a few iterations. The main drawback of the Newton direction is the need for the Hessian $\nabla^2 f(x)$. Explicit computation of this matrix of second derivatives can sometimes be a cumbersome, error-prone, and expensive process. Finite-difference and automatic differentiation techniques described in Chapter 8 may be useful in avoiding the need to calculate second derivatives by hand.

Definition

A sequence $\{x_k\}_{k=1}^{\infty}$ is said to converge to x_* **superlinearly** if the following limit holds:

$$\lim_{k\to\infty} \frac{\|x_{k+1} - x_*\|}{\|x_k - x_*\|} = 0\,.$$

§2.2 Overview of Algorithms

Quasi-Newton search directions provide an attractive alternative to Newton's method in that they do not require computation of the Hessian and yet still attain a **superlinear rate of convergence**. In place of the true Hessian $\nabla^2 f_k$, they use an approximation B_k, which is updated after each step to take account of the additional knowledge gained during the step. The updates make use of the fact that changes in the gradient g provide information about the second derivative of f along the search direction. Using the expression

$$\nabla f(x+p) = \nabla f(x) + \int_0^1 \nabla^2 f(x+tp)p\,dt,$$

by adding and subtracting the term $\nabla^2 f(x)p$ we have

$$\nabla f(x+p) = \nabla f(x) + \nabla^2 f(x)p + \int_0^1 \left[\nabla^2 f(x+tp) - \nabla^2 f(x)\right]p\,dt.$$

§2.2 Overview of Algorithms

Quasi-Newton search directions provide an attractive alternative to Newton's method in that they do not require computation of the Hessian and yet still attain a **superlinear rate of convergence**. In place of the true Hessian $\nabla^2 f_k$, they use an approximation B_k, which is updated after each step to take account of the additional knowledge gained during the step. The updates make use of the fact that changes in the gradient g provide information about the second derivative of f along the search direction. Using the expression

$$\nabla f(x+p) = \nabla f(x) + \int_0^1 \nabla^2 f(x+tp)p\,dt,$$

by adding and subtracting the term $\nabla^2 f(x)p$ we have

$$\nabla f(x+p) = \nabla f(x) + \nabla^2 f(x)p + \int_0^1 \left[\nabla^2 f(x+tp) - \nabla^2 f(x)\right]p\,dt.$$

§2.2 Overview of Algorithms

Quasi-Newton search directions provide an attractive alternative to Newton's method in that they do not require computation of the Hessian and yet still attain a **superlinear rate of convergence**. In place of the true Hessian $\nabla^2 f_k$, they use an approximation B_k, which is updated after each step to take account of the additional knowledge gained during the step. The updates make use of the fact that changes in the gradient g provide information about the second derivative of f along the search direction. Using the expression

$$\nabla f(x+p) = \nabla f(x) + \int_0^1 \nabla^2 f(x+tp)p\,dt,$$

by adding and subtracting the term $\nabla^2 f(x)p$ we have

$$\nabla f(x+p) = \nabla f(x) + \nabla^2 f(x)p + \int_0^1 \left[\nabla^2 f(x+tp) - \nabla^2 f(x)\right]p\,dt.$$

§2.2 Overview of Algorithms

Because ∇f is continuous, the size of the final integral term is $o(\|p\|)$. By setting $x = x_k$ and $p = x_{k+1} - x_k$, we obtain

$$\nabla f_{k+1} = \nabla f_k + \nabla^2 f_k (x_{k+1} - x_k) + o(\|x_{k+1} - x_k\|)\,.$$

When x_k and x_{k+1} lie in a region near the solution x_*, within which $\nabla^2 f$ is positive definite, the final term in this expansion is eventually dominated by the $\nabla^2 f_k (x_{k+1} - x_k)$ term, and we can write

$$\nabla^2 f_k (x_{k+1} - x_k) \approx \nabla f_{k+1} - \nabla f_k\,. \tag{7}$$

We choose the new Hessian approximation B_{k+1} so that it mimics the property (7) of the true Hessian; that is, we require it to satisfy the following condition, known as the secant equation:

$$B_{k+1}(x_{k+1} - x_k) = \nabla f_{k+1} - \nabla f_k\,.$$

§2.2 Overview of Algorithms

Because ∇f is continuous, the size of the final integral term is $o(\|p\|)$. By setting $x = x_k$ and $p = x_{k+1} - x_k$, we obtain

$$\nabla f_{k+1} = \nabla f_k + \nabla^2 f_k (x_{k+1} - x_k) + o(\|x_{k+1} - x_k\|)\,.$$

When x_k and x_{k+1} lie in a region near the solution x_*, within which $\nabla^2 f$ is positive definite, the final term in this expansion is eventually dominated by the $\nabla^2 f_k (x_{k+1} - x_k)$ term, and we can write

$$\nabla^2 f_k (x_{k+1} - x_k) \approx \nabla f_{k+1} - \nabla f_k\,. \tag{7}$$

We choose the new Hessian approximation B_{k+1} so that it mimics the property (7) of the true Hessian; that is, we require it to satisfy the following condition, known as the secant equation:

$$B_{k+1}(x_{k+1} - x_k) = \nabla f_{k+1} - \nabla f_k\,.$$

§2.2 Overview of Algorithms

Because ∇f is continuous, the size of the final integral term is $o(\|p\|)$. By setting $x = x_k$ and $p = x_{k+1} - x_k$, we obtain

$$\nabla f_{k+1} = \nabla f_k + \nabla^2 f_k (x_{k+1} - x_k) + o(\|x_{k+1} - x_k\|)\,.$$

When x_k and x_{k+1} lie in a region near the solution x_*, within which $\nabla^2 f$ is positive definite, the final term in this expansion is eventually dominated by the $\nabla^2 f_k (x_{k+1} - x_k)$ term, and we can write

$$\nabla^2 f_k (x_{k+1} - x_k) \approx \nabla f_{k+1} - \nabla f_k\,. \tag{7}$$

We choose the new Hessian approximation B_{k+1} so that it mimics the property (7) of the true Hessian; that is, we require it to satisfy the following condition, known as the secant equation:

$$B_{k+1}(x_{k+1} - x_k) = \nabla f_{k+1} - \nabla f_k\,.$$

§2.2 Overview of Algorithms

Typically, we impose additional conditions on B_{k+1}, such as **symmetry** (motivated by symmetry of the exact Hessian), and a requirement that **the difference** between successive approximations B_k and B_{k+1} **have low rank**.

Two of the most popular formulae for updating the Hessian approximation B_k are the **symmetric-rank-one (SR1) formula**, defined by

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^{\mathrm{T}}}{(y_k - B_k s_k)^{\mathrm{T}} s_k} \tag{8}$$

and the **BFGS formula**, named after its inventors, Broyden, Fletcher, Goldfarb, and Shanno, which is defined by

$$B_{k+1} = B_k - \frac{B_k s_k s_k^{\mathrm{T}} B_k}{s_k^{\mathrm{T}} B_k s_k} + \frac{y_k^{\mathrm{T}} y_k}{y_k^{\mathrm{T}} s_k}, \tag{9}$$

where $s_k = x_{k+1} - x_k$, $y_k = \nabla f_{k+1} - \nabla f_k$.

§2.2 Overview of Algorithms

Typically, we impose additional conditions on B_{k+1}, such as **symmetry** (motivated by symmetry of the exact Hessian), and a requirement that **the difference** between successive approximations B_k and B_{k+1} **have low rank**.

Two of the most popular formulae for updating the Hessian approximation B_k are the **symmetric-rank-one (SR1) formula**, defined by

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^{\mathrm{T}}}{(y_k - B_k s_k)^{\mathrm{T}} s_k} \tag{8}$$

and the **BFGS formula**, named after its inventors, Broyden, Fletcher, Goldfarb, and Shanno, which is defined by

$$B_{k+1} = B_k - \frac{B_k s_k s_k^{\mathrm{T}} B_k}{s_k^{\mathrm{T}} B_k s_k} + \frac{y_k^{\mathrm{T}} y_k}{y_k^{\mathrm{T}} s_k}, \tag{9}$$

where $s_k = x_{k+1} - x_k$, $y_k = \nabla f_{k+1} - \nabla f_k$.

§2.2 Overview of Algorithms

Note that the difference between the matrices B_k and B_{k+1} is a rank-one matrix in the case of (8) and a rank-two matrix in the case of (9). Both updates maintain symmetry and both satisfy the secant equation

$$B_{k+1}(x_{k+1} - x_k) = \nabla f_{k+1} - \nabla f_k$$

or equivalently,

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

One can show that BFGS update (9) generates positive definite approximations whenever the initial approximation B_0 is positive definite and $s_k^{\mathrm{T}} y_k > 0$. We discuss these issues further in Chapter 6.

Two variants of quasi-Newton methods designed to solve large problems – partially separable and limited-memory updating – are described in Chapter 7.

§2.2 Overview of Algorithms

Note that the difference between the matrices B_k and B_{k+1} is a rank-one matrix in the case of (8) and a rank-two matrix in the case of (9). Both updates maintain symmetry and both satisfy the secant equation

$$B_{k+1}(x_{k+1} - x_k) = \nabla f_{k+1} - \nabla f_k$$

or equivalently,

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

One can show that BFGS update (9) generates positive definite approximations whenever the initial approximation B_0 is positive definite and $s_k^{\mathrm{T}} y_k > 0$. We discuss these issues further in Chapter 6.

Two variants of quasi-Newton methods designed to solve large problems – partially separable and limited-memory updating – are described in Chapter 7.

§2.2 Overview of Algorithms

The quasi-Newton search direction is obtained by using B_k in place of the exact Hessian in the Newton direction; that is,

$$p_k = -B_k^{-1} \nabla f_k \,.$$

Some practical implementations of quasi-Newton methods avoid the need to factorize B_k at each iteration by updating the inverse of B_k, instead of B_k itself. In fact, the equivalent formula for (8) and (9), applied to the inverse approximation $H_k \equiv B_k^{-1}$, is

$$H_{k+1} = (\mathrm{I} - \rho_k s_k y_k^{\mathrm{T}}) H_k (\mathrm{I} - \rho_k y_k s_k^{\mathrm{T}}) + \rho_k s_k s_k^{\mathrm{T}} \,, \quad \rho_k = \frac{1}{y_k^{\mathrm{T}} s_k} \,. \quad (10)$$

Calculation of p_k can then be performed by using the formula $p_k = -H_k \nabla f_k$. This matrix–vector multiplication is simpler than the factorization/back-substitution procedure that is needed to implement the quasi-Newton search direction.

§2.2 Overview of Algorithms

The quasi-Newton search direction is obtained by using B_k in place of the exact Hessian in the Newton direction; that is,

$$p_k = -B_k^{-1}\nabla f_k\,.$$

Some practical implementations of quasi-Newton methods avoid the need to factorize B_k at each iteration by updating the inverse of B_k, instead of B_k itself. In fact, the equivalent formula for (8) and (9), applied to the inverse approximation $H_k \equiv B_k^{-1}$, is

$$H_{k+1} = (\mathrm{I} - \rho_k s_k y_k^{\mathrm{T}})H_k(\mathrm{I} - \rho_k y_k s_k^{\mathrm{T}}) + \rho_k s_k s_k^{\mathrm{T}}\,, \quad \rho_k = \frac{1}{y_k^{\mathrm{T}} s_k}\,. \quad (10)$$

Calculation of p_k can then be performed by using the formula $p_k = -H_k\nabla f_k$. This matrix–vector multiplication is simpler than the factorization/back-substitution procedure that is needed to implement the quasi-Newton search direction.

§2.2 Overview of Algorithms

The quasi-Newton search direction is obtained by using B_k in place of the exact Hessian in the Newton direction; that is,

$$p_k = -B_k^{-1}\nabla f_k\,.$$

Some practical implementations of quasi-Newton methods avoid the need to factorize B_k at each iteration by updating the inverse of B_k, instead of B_k itself. In fact, the equivalent formula for (8) and (9), applied to the inverse approximation $H_k \equiv B_k^{-1}$, is

$$H_{k+1} = (\mathrm{I} - \rho_k s_k y_k^{\mathrm{T}})H_k(\mathrm{I} - \rho_k y_k s_k^{\mathrm{T}}) + \rho_k s_k s_k^{\mathrm{T}}\,, \quad \rho_k = \frac{1}{y_k^{\mathrm{T}} s_k}\,. \quad (10)$$

Calculation of p_k can then be performed by using the formula $p_k = -H_k\nabla f_k$. This matrix–vector multiplication is simpler than the factorization/back-substitution procedure that is needed to implement the quasi-Newton search direction.

§2.2 Overview of Algorithms

The last class of search directions we preview here is that generated by nonlinear conjugate gradient methods. They have the form

$$p_k = -(\nabla f)(x_k) + \beta_k p_{k-1}\,,$$

where β_k is a scalar that ensures that p_k and p_{k-1} are conjugate – an important concept in the minimization of quadratic functions that will be defined in Chapter 5. Conjugate gradient methods were originally designed to solve systems of linear equations $Ax = b$, where the coefficient matrix A is symmetric and positive definite. The problem of solving this linear system is equivalent to the problem of minimizing the convex quadratic function defined by

$$\varphi(x) = \frac{1}{2}x^{\mathrm{T}}Ax - -b^{\mathrm{T}}x,$$

so it was natural to investigate extensions of these algorithms to more general types of unconstrained minimization problems.

§2.2 Overview of Algorithms

The last class of search directions we preview here is that generated by nonlinear conjugate gradient methods. They have the form

$$p_k = -(\nabla f)(x_k) + \beta_k p_{k-1}\,,$$

where β_k is a scalar that ensures that p_k and p_{k-1} are conjugate – an important concept in the minimization of quadratic functions that will be defined in Chapter 5. Conjugate gradient methods were originally designed to solve systems of linear equations $Ax = b$, where the coefficient matrix A is symmetric and positive definite. The problem of solving this linear system is equivalent to the problem of minimizing the convex quadratic function defined by

$$\varphi(x) = \frac{1}{2}x^{\mathrm{T}}Ax - -b^{\mathrm{T}}x,$$

so it was natural to investigate extensions of these algorithms to more general types of unconstrained minimization problems.

§2.2 Overview of Algorithms

The last class of search directions we preview here is that generated by nonlinear conjugate gradient methods. They have the form

$$p_k = -(\nabla f)(x_k) + \beta_k p_{k-1}\,,$$

where β_k is a scalar that ensures that p_k and p_{k-1} are conjugate – an important concept in the minimization of quadratic functions that will be defined in Chapter 5. Conjugate gradient methods were originally designed to solve systems of linear equations $Ax = b$, where the coefficient matrix A is symmetric and positive definite. The problem of solving this linear system is equivalent to the problem of minimizing the convex quadratic function defined by

$$\varphi(x) = \frac{1}{2}x^{\mathrm{T}}Ax - -b^{\mathrm{T}}x,$$

so it was natural to investigate extensions of these algorithms to more general types of unconstrained minimization problems.

§2.2 Overview of Algorithms

In general, nonlinear conjugate gradient directions are much more effective than the steepest descent direction and are almost as simple to compute. These methods do not attain the fast convergence rates of Newton or quasi-Newton methods, but they have the advantage of not requiring storage of matrices. An extensive discussion of nonlinear conjugate gradient methods is given in Chapter 5. All of the search directions discussed so far can be used directly in a line search framework. They give rise to the steepest descent, Newton, quasi-Newton, and conjugate gradient line search methods. All except conjugate gradients have an analogue in the trust-region framework, as we now discuss.

§2.2 Overview of Algorithms

In general, nonlinear conjugate gradient directions are much more effective than the steepest descent direction and are almost as simple to compute. These methods do not attain the fast convergence rates of Newton or quasi-Newton methods, but they have the advantage of not requiring storage of matrices. An extensive discussion of nonlinear conjugate gradient methods is given in Chapter 5. All of the search directions discussed so far can be used directly in a line search framework. They give rise to the steepest descent, Newton, quasi-Newton, and conjugate gradient line search methods. All except conjugate gradients have an analogue in the trust-region framework, as we now discuss.

§2.2 Overview of Algorithms

- **Models for Trust-Region Methods**

If we set $B_k = 0$ in

$$m_k(x_k + p) = f_k + p^{\mathrm{T}} \nabla f_k + \frac{1}{2} p^{\mathrm{T}} B_k p \tag{4}$$

and define the trust region using the Euclidean norm, the trust-region sub-problem (3) becomes

$$\min_{p \in \mathbb{R}^n} (f_k + p^{\mathrm{T}} \nabla f_k) \quad \text{subject to} \quad \|p\| \leqslant \Delta_k.$$

The closed form solution to the minimization problem above is

$$p_k = -\frac{\Delta_k \nabla f_k}{\|\nabla f_k\|}.$$

This is simply a **steepest descent step** in which the step length is determined by the trust region radius; the trust-region and line search approaches are essentially the same in this case.

§2.2 Overview of Algorithms

- **Models for Trust-Region Methods**

If we set $B_k = 0$ in

$$m_k(x_k + p) = f_k + p^{\mathrm{T}}\nabla f_k + \frac{1}{2}p^{\mathrm{T}}B_k p \tag{4}$$

and define the trust region using the Euclidean norm, the trust-region sub-problem (3) becomes

$$\min_{p\in\mathbb{R}^n}(f_k + p^{\mathrm{T}}\nabla f_k) \quad \text{subject to} \quad \|p\| \leqslant \Delta_k.$$

The closed form solution to the minimization problem above is

$$p_k = -\frac{\Delta_k \nabla f_k}{\|\nabla f_k\|}.$$

This is simply a **steepest descent step** in which the step length is determined by the trust region radius; the trust-region and line search approaches are essentially the same in this case.

§2.2 Overview of Algorithms

A more interesting trust-region algorithm is obtained by choosing B_k to be the exact Hessian $\nabla^2 f_k$ in the quadratic model (4). Because of the trust-region restriction $\|p\| \leqslant \Delta_k$, the sub-problem (3) is guaranteed to have a solution p_k even when $\nabla^2 f_k$ is not positive definite. The trust-region Newton method has proved to be highly effective in practice, as we discuss in Chapter 7.

If the matrix B_k in the quadratic model function m_k of (4) is defined by means of a quasi-Newton approximation, we obtain a trust-region quasi-Newton method.

§2.2 Overview of Algorithms

A more interesting trust-region algorithm is obtained by choosing B_k to be the exact Hessian $\nabla^2 f_k$ in the quadratic model (4). Because of the trust-region restriction $\|p\| \leqslant \Delta_k$, the sub-problem (3) is guaranteed to have a solution p_k even when $\nabla^2 f_k$ is not positive definite. The trust-region Newton method has proved to be highly effective in practice, as we discuss in Chapter 7.

If the matrix B_k in the quadratic model function m_k of (4) is defined by means of a quasi-Newton approximation, we obtain a trust-region quasi-Newton method.

§2.2 Overview of Algorithms

- **Scaling**

The performance of an algorithm may depend crucially on how the problem is formulated. One important issue in problem formulation is scaling. In unconstrained optimization, a problem is said to be **poorly scaled** if changes to x in a certain direction produce much larger variations in the value of f than do changes to x in another direction.

Scaling is performed (sometimes unintentionally) when the units used to represent variables are changed. During the modeling process, we may decide to change the units of some variables, say from meters to millimeters. By doing so, the range of those variables and their size relative to the other variables will both change.

§2.2 Overview of Algorithms

- **Scaling**

The performance of an algorithm may depend crucially on how the problem is formulated. One important issue in problem formulation is scaling. In unconstrained optimization, a problem is said to be **poorly scaled** if changes to x in a certain direction produce much larger variations in the value of f than do changes to x in another direction.

Scaling is performed (sometimes unintentionally) when the units used to represent variables are changed. During the modeling process, we may decide to change the units of some variables, say from meters to millimeters. By doing so, the range of those variables and their size relative to the other variables will both change.

§2.2 Overview of Algorithms

Some optimization algorithms, such as steepest descent, are sensitive to poor scaling, while others, such as Newton's method, are unaffected by it. Figure 2 shows the contours of two convex nearly quadratic functions, the first of which is poorly scaled, while the second is well scaled.

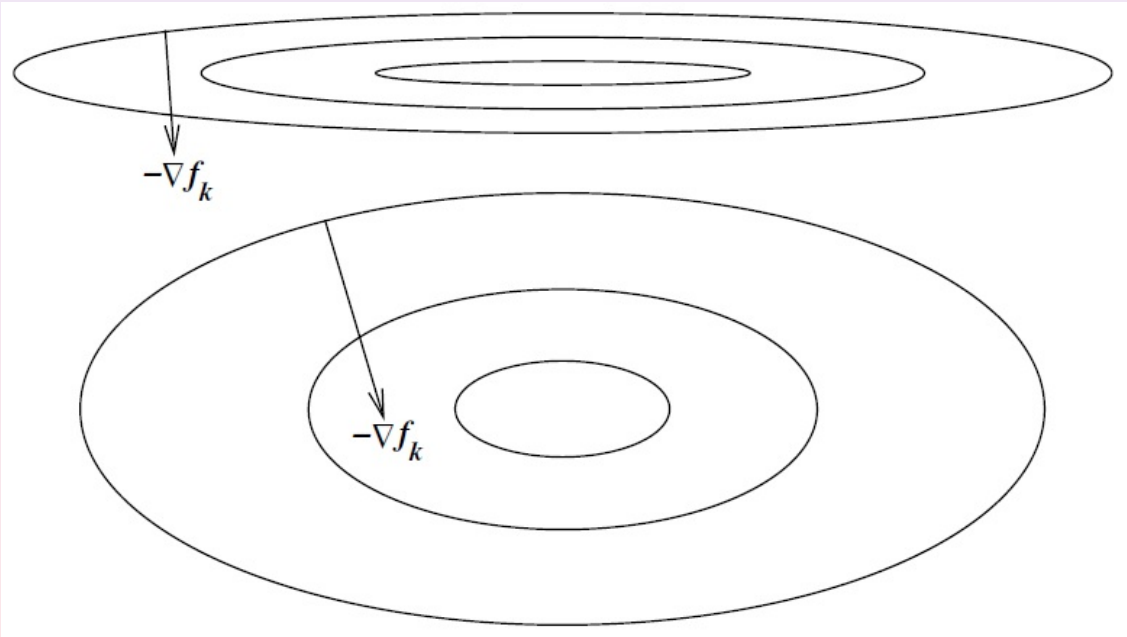

Figure 2: Poorly scaled and well scaled problems, and performance of the steepest descent direction.

§2.2 Overview of Algorithms

For the poorly scaled problem, the one with highly elongated contours, the steepest descent direction does not yield much reduction in the function, while for the well-scaled problem it performs much better. In both cases, Newton's method will produce a much better step, since the second-order quadratic model

$$m_k(p) \equiv f_k + p^{\mathrm{T}} \nabla f_k + \frac{1}{2} p^{\mathrm{T}} \nabla^2 f_k p \tag{5}$$

happens to be a good approximation of f.

Algorithms that are not sensitive to scaling are preferable, because they can handle poor problem formulations in a more robust fashion.