APPM2007A/APPM2020A Optimisation II

Krupa Prag
Anani Ngowi

2022-11-07

Contents

List of Tables

List of Figures

Course Outline

Course Structure and Details

- **Office: UG04 MSB**
- **Tutorials: Alternate Wednesdays - 12:30 - 13:15**
- **Lectures: Monday - 8:00-9:45 MSL004/005 or MS Team (Blended)**

Note: The course will follow a blended format. Online interactions will take place on MS Teams and face-to-face interactions will take place at MSL (or an alternative venue). The format per lesson will be communicated.

Course Information

Course mark:

- Lab assignments.
- Test.
- Final assesment/exam.

Please see the course outline for further details.

Course Topics

We will be covering the following topics throughout the course:

- Generalised concept of optimisation
- One dimensional unconstrained and bound constrained problems

- Numerical differentiation and solutions to nonlinear equations
- Numerical optimisation of univariate functions
- Multivariate unconstrained optimisation
- Gradient methods for unconstrained optimisation
- Newton and Quasi-Newton methods
- Direct search method for unconstrained problems
- Lagrangian multipliers for constraint optimisation

These notes are authored by Prof Montaz Ali and edited by Dr Matthew Woolway and Krupa Prag.

Hardware Requirements

The course will be very computational in nature, however, you do not need your own personal machine. MSL already has **Python** installed. The labs will be running the IDEs for Python (along with Jupyter) while I will be using Jupyter for easier presentation and explanation in lectures. You will at some point need to become familiar with Jupyter as the tests will be conducted in the Maths Science Labs (MSL) utilising this platform for autograding purposes.

If you do have your own machine and would prefer to work from that you are more than welcome. Since all the notes and code will be presented through Jupyter please follow the following steps:

- Install Anaconda from here: https://repo.continuum.io/archive/Anaconda3-5.2.0-Windows-x86_64.exe
  - Make sure when installing Anaconda to set the installation to PATH when prompted (it will be deselected by default)
- To launch a Jupyter notebook, open the command promt (cmd) and type jupyter notebook. This should launch the browser and jupyter. If you see any proxy issues while on campus, then you will need to set the proxy to exclude the localhost.

If you are not running Windows but rather Linux, then you can get Anaconda at: https://repo.continuum.io/archive/Anaconda3-5.2.0-Linux-x86_64.sh

Chapter 1

Definition and General Concepts

In industry, commerce, government, indeed in all walks of life, one frequently needs answers to questions concerning operational efficiency. Thus an architect may need to know how to lay out a factory floor so that the article being manufactured does not have to be moved about too much as it goes from one machine tool to another; the manager of a shipping company needs to plan the itineraries of his ships so as to increase the amount of goods handled, while avoiding costly waiting-around in docks. A telecommunications engineer may want to know how best to transmit signals so as to minimise the possibility of error on reception. Further examples of problems of this sort are provided by the planning of a railroad time-table to ensure that trains are available as and when needed, the synchronisation of traffic lights, and many other real-life situations. Formerly such problems would usually be 'solved' by imprecise methods giving results that were both unreliable and costly. Today, they are increasingly being subjected to rigid mathematical analysis, designed to provide methods for finding exact solutions or highly reliable estimates rather than vague approximations. Optimisation provide many of the mathematical tools used for solving such problems.

Optimisation, or mathematical programming, is the study of maximising and minimising functions subject to specified boundary conditions or constraints. The functions to be optimised arise in engineering, the physical and management sciences, and in various branches of mathematics. With the emergence of the computer age, optimisation experienced a dramatic growth as a mathematical theory, enhancing both combinatorics and classical analysis. In its applications to engineering and management science, optimisation (linear programming) forms an important part of the discipline of operations research.

1.1 Nonlinear Optimisation

Linear programming has proved an extremely powerful tool, both in modelling real-world problems and as a widely applicable mathematical theory. However, many interesting but practical optimisation problems are nonlinear. The study of such problems involves a diverse blend of linear algebra, multivariate calculus, quadratic form, numerical analysis and computing techniques. Important special areas include the design of computational algorithms, the geometry and analysis of convex sets and functions, and the study of specially structured problems such as unconstrained and constrained nonlinear optimisation problems. Nonlinear optimisation provides fundamental insights into mathematical analysis, and is widely used in the applied sciences, in areas such as engineering design, regression analysis, inventory control, and geophysical exploration among others. General nonlinear optimisation problems and various computational algorithm to addressing such problems will be taught in this course.

1.2 General Statement of an Optimisation Problem

Optimisation problems are made of three primary ingredients; (i) an objective function, (ii) variables (unknowns) and (iii) the constraints.

- An **objective function** which we want to minimize or maximize. For instance, in a manufacturing process, we might want to *maximize the profit* or *minimize the cost.* In fitting experimental data to a user-defined model, we might minimize the total deviation of observed data from predictions based on the model. In designing an automobile panel, we might want to *maximize the strength.*
- A set of **unknowns/variables** which affect the value of the objective function. In the manufacturing problem, the variables might include the *amounts of different resources used* or the *time spent on each activity.* In fitting-the-data problem, the unknowns are the *parameters* that define the model. In the panel design problem, the variables used define the *shape and dimensions* of the panel.
- A set of **constraints** that allow the unknowns to take on certain values but exclude others. For the manufacturing problem, it does not make sense to spend a negative amount of time on any activity, so we constrain all the 'time' variables to be non-negative. In the panel design problem, we would probably want to limit the *weight* of the product and to constrain its *shape.* The optimisation problem is then :

Find values of the variables that minimize or maximize the objective function while satisfying the constraints.

The general statement being:

$$\underset{\text{w.r.t } \mathbf{x}}{\text{minimise}}\ f(\mathbf{x}),\ \ \mathbf{x} = [x_1, x_2, \ldots, x_n]^T \in \mathbb{R}^n \tag{1.1}$$

subject to:

$$g_j(\mathbf{x}) \leq 0,\ \ j = 1, 2, \ldots, m,$$
$$h_j(\mathbf{x}) = 0,\ \ j = 1, 2, \ldots, r.$$

where $f(\mathbf{x})$, $g_j(\mathbf{x})$ and $h_j(\mathbf{x})$ are scalar functions of the real vector $\mathbf{x}$.

The continuous components x_i of $\mathbf{x}$ are called the the *variables*, $f(\mathbf{x})$ is the objective function, $g_j(\mathbf{x})$ denotes the respective *inequality constraint functions* and $h_j(\mathbf{x})$ the *equality constraint functions*. The optimum vector $\mathbf{x}$ that solves Equation (1.1) is denoted by $\mathbf{x}^*$ with a corresponding optimum function value $f(\mathbf{x}^*)$. If there are no constraints specified, then the problem is aptly named an *unconstrained* minimisation problem. A large quantity of progress has been made when solving different classes of the general problem introduced in Equation (1.1). On occasion these solutions can be attained analytically yielding a closed-form solution. However, most real world problems exist where $n > 2$ and as a result need to be solved numerically through suitable computational algorithms.

1.3 Important Optimisation Concepts

1.3.1 Definitions

1.3.1.1 Neighbourhoods

Definition 1.1 (Neighbourhood)**.** δ-neighbourhood of a point, say $\mathbf{y}$, where $\mathbf{y} \in \mathbb{R}^n$:

A δ-neighbourhood of a point $\mathbf{y}$ is the set of all points within a neighbourhood of $\mathbf{y}$, it is denoted by $N_\delta(\mathbf{y})$. It is the set of all points $\mathbf{x}$ such that:

$$N_\delta(\mathbf{y}) : \|\mathbf{x} - \mathbf{y}\| \leq \delta, \tag{1.2}$$

that is $\mathbf{x} \in N_\delta(\mathbf{y})$.

1.3.1.2 Local and Global Minimisers

Definition 1.2 (Global Minimiser/Maximiser)**.** If a point $\mathbf{x} \in S$, where S is the **feasible set**, then $\mathbf{x}$ is a feasible solution. A feasible solution $\mathbf{x_g}$ of some problem P is the **global** minimiser of $f(\mathbf{x})$ if:

$$f(\mathbf{x_g}) \leq f(\mathbf{x}), \tag{1.3}$$

for all feasible points $\mathbf{x}$ of the problem P. The value $f(\mathbf{x_g})$ is called the global minimum. The converse applies for the global maximum.

Definition 1.3 (Local Minimiser/Maximiser)**.** A point $\mathbf{x}^*$ is called a **local** minimiser of $f(\mathbf{x})$ if there exists a suitable $\delta > 0$ such that for all feasible $\mathbf{x} \in N_\delta(\mathbf{x}^*)$:

$$f(\mathbf{x}^*) \leq f(\mathbf{x}). \tag{1.4}$$

In other words, $\mathbf{x}^*$ is a local minimiser of $f(\mathbf{x})$ if there exists a neighbourhood $N_\delta(\mathbf{x}^*)$ of $\mathbf{x}^*$ containing feasible $\mathbf{x}$ such that:

$$f(\mathbf{x}^*) \leq f(\mathbf{x}), \ \ \forall \in N_\delta(\mathbf{x}^*). \tag{1.5}$$

Again the converse applies for local maximisers.

1.3.1.3 Infimum and Supremum of a Function

Definition 1.4 (Infimum)**.** If f is a function on S, the greatest lower bound or **infimum** of f on S is the largest number m (possibly $m = -\infty$) such that $f(\mathbf{x}) \geq m \ \ \forall \mathbf{x} \in S$. The infimum is denoted by:

$$\inf_{\mathbf{x} \in S} \ f(\mathbf{x}). \tag{1.6}$$

Definition 1.5 (Supremum)**.** If f is a function on S, the least upper bound or **supremum** of f on S is the smallest number m (possibly $m = +\infty$) such that $f(\mathbf{x}) \leq m \ \ \forall \mathbf{x} \in S$. The supremum is denoted by:

$$\sup_{\mathbf{x} \in S} \ f(\mathbf{x}). \tag{1.7}$$

1.3.2 Convexity

1.3.2.1 Affine Set

Definition 1.6 (Affine Set)**.** A line though the points $\mathbf{x}_1$ and $\mathbf{x}_2$ in $\mathbb{R}^n$ is the set:

$$L = \{\mathbf{x}| \ \ \theta\mathbf{x}_1 + (1-\theta)\mathbf{x}_2, \ \ \ \forall \ \theta \in \mathbb{R}\}. \tag{1.8}$$

This is known as an **Affine Set**. An example of this is the solution of linear equations $Ax = b$.

1.3.2.2 Convex Set

Definition 1.7 (Convex Set)**.** A set $S \subset R^n$ is a **Convex Set** if for all $\mathbf{x}_1, \mathbf{x}_2 \in S$, the line segment between x_1 and x_2 is in S. The *line segment* between the points $\mathbf{x}_1$ and $\mathbf{x}_2$, can be represented as:

$$\mathbf{x} = \theta\mathbf{x}_1 + (1-\theta)\mathbf{x}_2, \ \ \text{where } 0 \leq \theta \leq 1. \tag{1.9}$$

If this condition does not hold then the set is non-convex. Think of this as *line of sight*. Some are examples are considered in the Figure below:

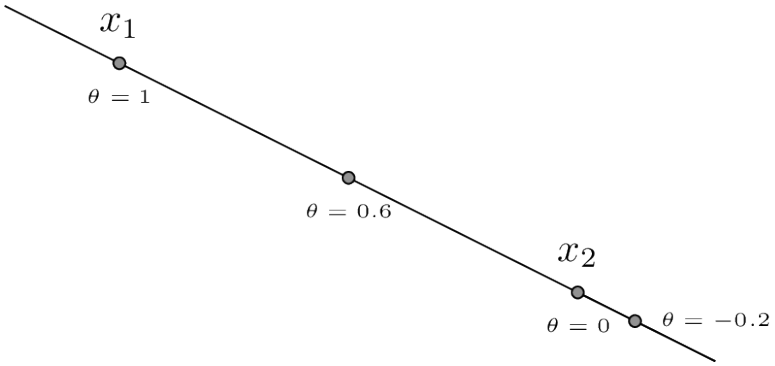


Figure 1.1: A Line Segement.

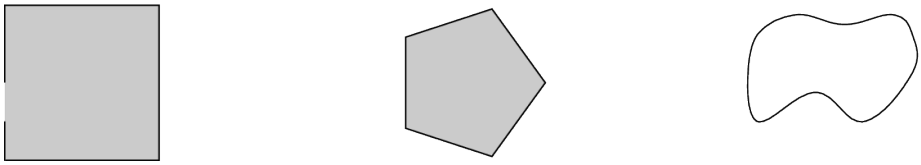


Figure 1.2: Left and Right are Non-Convex, Center Convex.

1.3.2.3 Convex Combination and Convex Hull

Definition 1.8 (Convex Combination)**.** We can define the **Convex Combination** of $\mathbf{x}_1,\ldots,\mathbf{x}_n$, as any point $\mathbf{x}$ which satisfies:

$$\mathbf{x} = \theta_1\mathbf{x}_1 + \theta_2\mathbf{x}_2 + \ldots + \theta_n\mathbf{x}_n, \tag{1.10}$$

where $\theta_1 + \ldots + \theta_n = 1,\ \theta_i \geq 0$.

The **Convex Hull** (**conv L**) is the set of all convex combinations of the points in **L**. This can be thought of as the tightest bound across all points in the set, as can be seen in the Figure below:

1.3.2.4 Hyperplanes and Halfspaces

Definition 1.9 (Hyperplane/Halfspace)**.** We can define the **Hyperplane** as the set of the form:

$$\{x \mid a^T x = b\}\ (a \neq 0), \tag{1.11}$$

we can define the **Halfspace** as the set of the form:

$$\{x \mid a^T x \leq b\}\ (a \neq 0), \tag{1.12}$$

where a is the normal vector.

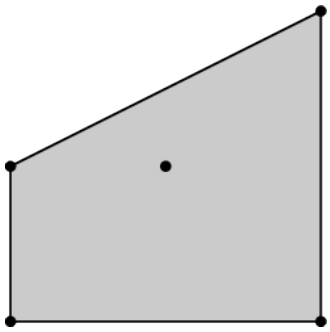


Figure 1.3: An Example of a Convex Hull.

Note: Hyperplanes are both affine and convex, while halfspaces are only convex. These are illustrated in the Figures below:

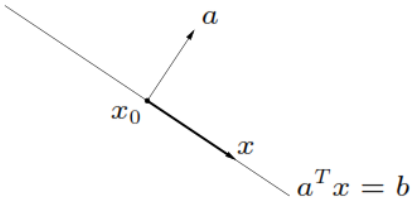


Figure 1.4: A Hyperplane.

1.3.2.5 Level Sets and Level Surfaces

Definition 1.10 (Level Set)**.** Consider the real valued function f on L. Let a be in $\mathbf{R}^1$, then we denote L_a to be the set:

$$L_a = \{x \in L \mid f(x) = a\}. \tag{1.13}$$

The level set is the set of points that have a corresponding function value euqal to that of the constant value a.

Definition 1.11 (Level Surface)**.** Consider the real valued function f on L. Let a be in $\mathbf{R}^3$, then we denote C_a to be the set:

$$C_a = \{\mathbf{x} \in L \mid f(\mathbf{x}) = a\}. \tag{1.14}$$

These sets are known as level surfaces of f on L and can be thought of as the cross section taken at some point $\mathbf{x}_0 \in L$.

See the example of $x^2y + xy^2$ below:

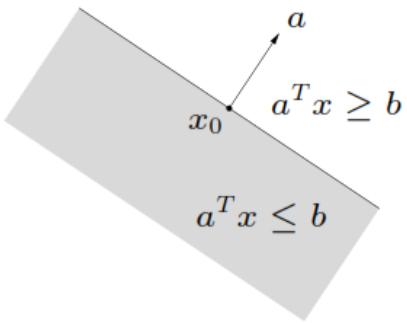


Figure 1.5: A Halfspace.

1.3.3 Exercises

1. Find the *convex hull* of the following sets:

$$\begin{aligned} S_1 &= \{(x_1, x_2) \mid x_1^2 + x_2^2 \leq 1\} \\ S_2 &= \{(x_1, x_2) \mid x_1^2 + x_2^2 > 1\} \\ S_3 &= \{(0,0), (1,0), (1,1), (0,1)\} \\ S_4 &= \{(x_1, x_2) \mid |x_1| + |x_2| < 1\} \end{aligned}$$

2. Find the minimum (if any) of the following:
   - min $\frac{1}{x}$ for $x \in [1, \infty)$
   - min x for $x \in (0, 1]$
3. Determine the supremum, infimum, maximum and minimum of the following functions:
   - $f(x) = e^x, \ x \in \mathbb{R}$
   - Let $f : L \subset \mathbb{R}^2 \to \mathbb{R}$. The set L is defined by the disc $x_1^2 + x_2^2 \leq 1 \in \mathbb{R}^2$ and:

$$f(x_1, x_2) = e^{x_1^2 + x_2^2}.$$

4. In each of the following cases, sketch the level sets L_b of the function f:
   - $f(x_1, x_2) = x_1 + x_2, \ b = 1, 2$
   - $f(x_1, x_2) = x_1 x_2, \ b = 1, 2$
   - $f(x) = e^x, \ b = 10, 0$
5. Let L be a convex set in $\mathbb{R}^n$, A be an $m \times n$ matrix and α a scalar. Show that the following sets are convex.
   - $\{y : y = Ax, \ x \in L\}$

- $\{\alpha x : x \in L\}$

6. If you have two points that solve a system of linear equations $Ax = b$, i.e. points x_1 and x_2, where $x_1 \neq x_2$. Then prove that the line that passes through these two points is in the affine set.
7. Prove that a halfspace is convex.

Chapter 2

One Dimensional Unconstrained and Bound Constrained Problems

The one dimensional *unconstrained problem* is defined by:

$$\underset{x \in L}{\text{minimise}}\ f(x),\ \ L \subseteq \mathbb{R}, \tag{2.1}$$

where f is a continuous and twice differentiable function. If L is an interval, then x is bound constrained. If L is indeed $\mathbb{R}$, then the problem is unconstrained.

2.1 Unimodal and Multimodal

A function is said to be *monotonic increasing* along a given path when $f(x_2) > f(x_1)$, and *monotonic decreasing* when $f(x_2) < f(x_1)$ for all points in the domain for which $x_2 > x_1$. For the situation in which $f(x_2) \geq f(x_1)$ and $f(x_2) \leq f(x_1)$ the functions are respectively called *monotonic non-decreasing* and *monotonic non-increasing*.

For example if we consider $f(x) = x^2$, where $x > 0$, then $f(x)$ is monotonic increasing. For $x < 0$, $f(x)$ is monotonic decreasing. A function that has a single minimum or a single maximum (single peak) is known as unimodal function. Functions with two peaks (two minima or two maxima) are called bimodal and functions with many peaks are known as multimodal functions.

2.2 Convex Functions

Definition 2.1 (Convex Functions)**.** A function $f : L \subset \mathbb{R}^n \to \mathbb{R}$ defined on the convex set L is convex if for all $\mathbf{x}_1, \mathbf{x}_2 \in L$, for all $\theta \in [0,1]$:

$$f(\theta \mathbf{x}_2 + (1-\theta)\mathbf{x}_1) \leq \theta f(\mathbf{x}_2) + (1-\theta) f(\mathbf{x}_1). \tag{2.2}$$

Consider the function of a single variable. We may think of this easily then by saying; *The function f is convex if the chord connecting x_1 and x_2 lays above the graph.*

See the Figure below:

Note:

> The function is strictly convex if only $<$ applies f is concave if $-f$ is convex

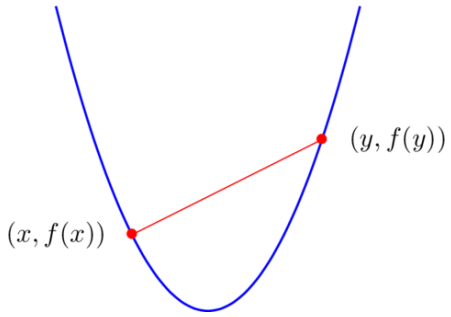


Figure 2.1: An Example of a Convex Function.

2.3 Global Extrema

We shall summarise some definitions that were previously mentioned. We begin with the concept of global optimisation (maximisation or minimisation). In the context of optimisation, relative optima (maximum or minimum) are normally referred to as local optima.

Global Optima

A point $f(x)$ attains its greatest (or least) value on an interval $[a,b]$ is called a point of **global** maximum (or minimum). In general, however, a function $f(x)$ takes on its absolute (global) maximum (minimum) at a point x^* if $f(x) < f(x^*)$ ($f(x) > f(x^*)$) for all x over which the function $f(x)$ is defined.

Local Optima

$f(x)$ has a **strong** local (relative) maximum (minimum) at an interior point $x^* \in (a,b)$ if $f(x) < f(x^*)$ ($f(x) > f(x^*)$) for all x in some neighbourhood of x^*. The maximum (minimum) is weak if $\leq$ replaces $<$. Strong local are illustrated in the Figure below. If a function $f(x)$ has a strong relative maximum at some point x^*, then there is an interval including x^*, no matter how small, such that for all x in this interval, $f(x)$ is strictly less than $f(x^*)$, i.e. $f(x) < f(x^*)$. It is the 'strictly less' that makes this a stronger relative maximum. If however, the strictly less sign is replaced by a $\leq$ sign, i.e. $f(x) \leq f(x^*)$, then the minimum value at x^* is a weak minimum and x^* is a weak minimiser.

2.4 Necessary and Sufficient Conditions

A necessary condition for $f(x)$ to have a maximum or a minimum at an interior point $x^* \in (a,b)$ is that the slope at this point must be zero, i.e. where:

$$f'(x) = \frac{df(x)}{dx} = 0, \tag{2.3}$$

which corresponds to the **first order necessary condition** (FONC). The FONC may be necessary but it is not sufficient. For example, consider the function $f(x) = x^3$ as seen in the Figure below. At $x = 0$, $f'(x) = 0$ but there is no maximum or minimum point on the interval $(-\infty, \infty)$. At $x = 0$ there is a point of *inflection*, where $f'' = 0$. Therefore, the point $x = 0$ is a stationary point but not a local optima.

Thus in addition to the FONC, **non-negative curvature** is also necessary at x^*, i.e. it is required that the second order condition:

$$f''(x) = \frac{d^2 f(x)}{dx^2} > 0, \tag{2.4}$$

must hold at x^* for a strong local minimum. This is known as the **second order sufficient condition** (SOSC).

2.4.1 Exercises

1. If the convexity condition for any real valued function $f : \mathbb{R} \to \mathbb{R}$ is given by:

   $$\lambda f(b) + (1-\lambda) f(a) \geq f(\lambda b + (1-\lambda) a), \quad [a,b] \in \mathbb{R},$$

   then using the above, prove that the following one dimensional functions are convex;

   - $f_1(x) = 1 + x^2$
   - $f_2(x) = x^2 - 1$

2. Find all stationary points of:

$$f(x) = x^3(3x^2 - 5) + 1,$$

and decide the maximiser, minimiser and the point of inflection, if any.

3. Using the FONC of optimality, determine the optimiser of the following functions:
   - $f(x) = \frac{1}{3}x^3 - \frac{7}{2}x^2 + 12x + 3$
   - $f(x) = 2(x-1)^2 + x^2$
4. Using necessary and sufficient conditions for optimality, investigate the maximiser/minimiser of:

$$f(x) = -(x-1)^4.$$

5. The function $f(x) = \max\{0.5, x, x^2\}$ is convex. True or false?
6. The function $f(x) = \min\{0.5, x, x^2\}$ is concave. True or false?
7. The function $f(x) = \dfrac{x^2+2}{x+2}$ is concave. True or false?
8. Determine the maximum and minimum values of the function:

$$f(x) = 12x^5 - 45x^4 + 40x^3 + 5$$

Chapter 3

Numerical Solutions to Nonlinear Equations

It is often necessary to find the stationary point(s) of a given function $f(x)$. This would mean finding the root of a nonlinear function $g(x)$ if we consider $g(x) = f'(x) = 0$. In other words, solving $g(x) = 0$. Here, we introduce the Newton method. This method is important because when we cannot solve $f'(x) = 0$ analytically we will be able to solve numerically.

3.1 Newton's Method

Newton's method is one of the more powerful and well known numerical methods for finding a root of $g(x)$ i.e. for solving for x such that $g(x) = 0$. So we can use it to find the turning point i.e., when $f'(x) = 0$. In the context of optimization we want an x^* such that $f'(x^*) = 0$. Consider the figure below:

Suppose at some stage we have obtained the point x_n as an approximation to the root at x^* (initially this is a guess). Newton observed that if $g(x)$ was a straight line through $(x_n, g(x_n))$ with slope $= g'(x_n) = g'(x) =$ constant for all x then the equation of the straight line could be found and the root read off. Obviously there would be no problem if $g(x)$ was actually a straight line; however the tangent might still be a good approximation (as seen in the Figure above). If we regard the tangent as a model of the function $g(x)$ and we have an approximation x_n then we can produce a better approximation x_{n+1}. The method can be applied again and again, to give a sequence of values, each approximating x^* with more and more certainty.

The general equation of the tangent to the curve of $g(x)$ at $(x_n, g(x_n))$ has slope

$$g'(x_n) = \frac{(y - g(x_n))}{(x - x_n)}. \quad \text{When} \quad y = 0, \quad \text{let} \tag{3.1}$$

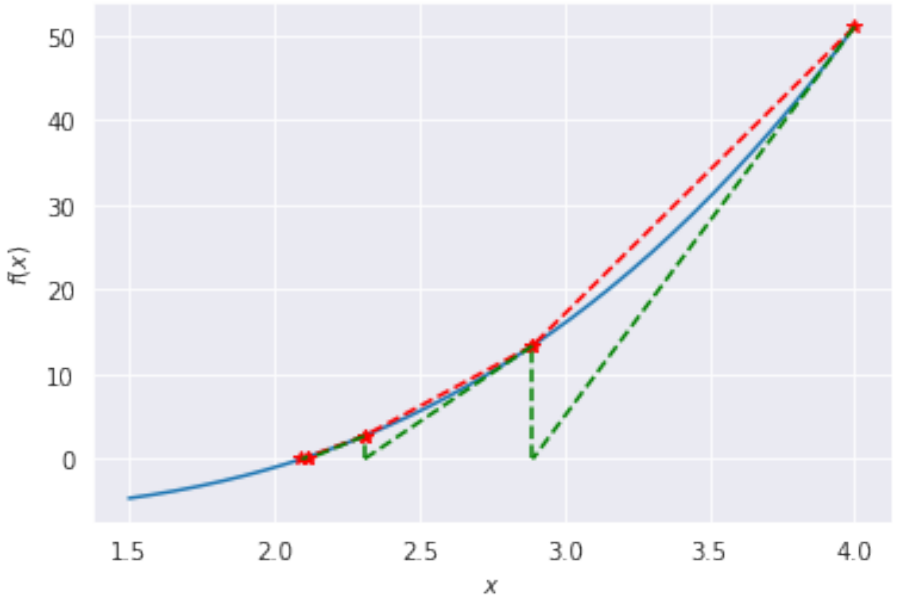


Figure 3.1: Newton's Method in Action.

$x = x_{n+1}$ (the point where the tangent cuts the x-axis).

Thus the Newton formula for root finding is :

$$x_{n+1} = x_n - \frac{g(x_n)}{g'(x_n)} \qquad g'(x_n) \neq 0 \tag{3.2}$$

Hence the Newton method can be described be the following two steps:
1. Get an initial 'guess' x_0.
2. Iterate by $x_{n+1} = x_n - f'(x_n)/f''(x_n)$.
x_n converges to a turning point for suitable choice of x_0.

3.1.0.1 Example

Find the root of $f(x) = \frac{x^4}{4} + \frac{x^2}{2} - 3x$ near $x = 2$.

We can see that to find the root of the deritive yields the minimum value of $f(x)$ in this case (approximately 1.21341). **Check:** perform a few iterations of Equation (3.2) to check the solution.

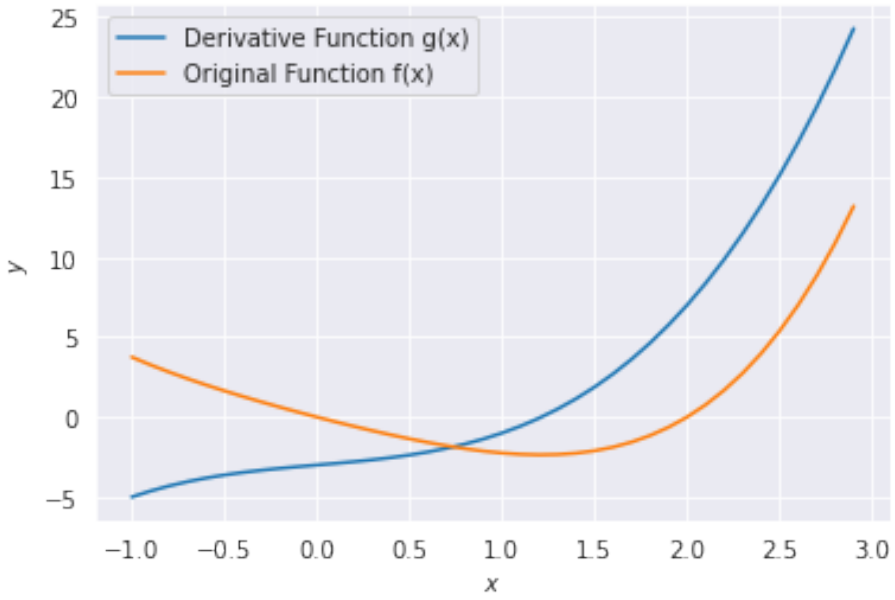


Figure 3.2: Function $f(x)$ and it's derivative $g(x)$.

3.1.1 Advantages and Disadvantages of Newton's Method

Advantages

- The method is really fast when it works (quadratic convergence)

Disadvantages:

- Unknown number of steps needed for required accuracy, unlike Bisection for example.
- f must be at least twice differentiable
- Run into problems when $g'(x^*) = 0$.
- Potentially could be difficult to compute $g(x)$ and $g'(x)$ even if they do exist.

In general Newton's Method is fast, reliable and trouble-free, but one has to be mindful of the potential problems.

3.2 Secant Method

Here we try to avoid the problems of computing $f''(x)$ and approximate it by $\frac{f'(x_n)-f'(x_{n-1})}{x_n-x_{n-1}}$. This given the updating formula:

$$x_{n+1} = x_n - f'(x_n)\frac{(x_n - x_{n-1})}{\left(f'(x_n) - f'(x_{n-1})\right)}. \tag{3.3}$$

The other disadvantages of Newton's method still apply.

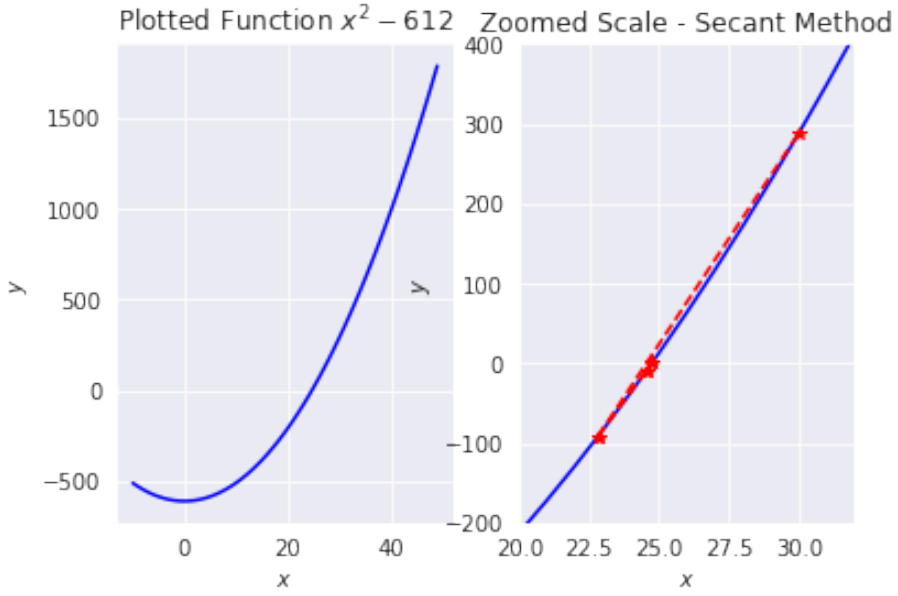


Figure 3.3: Secant Method

3.2.1 Exercises

1. Beginning with $x = 0$, apply Newton's Method to find the solution of

$$3x - \sin(x) - \exp(x) = 0,$$

   up to four iterations. Here x is in radians.
2. Find the critical point(s) of the function

$$f(x) = \frac{1}{4}x^4 + \frac{1}{2}x^2 - 2x + 1,$$

using both Newton's Method and the Secant Method. If the critical point is a minimiser then obtain the minimum value. You may assume $x = 2$ as an initial guess.

Chapter 4

Numerical Optimisation of Univariate Functions

The simplest functions with which to begin a study of non-linear optimisation methods are those with a single independent variable. Although the minimisation of univariate functions is in itself of some practical importance, the main area of application for these techniques is as a subproblem of multivariate minimisation.

There are functions to be minimised where the variable x is unrestricted (say, $x \in \mathbb{R}$); there are also functions to be optimised over a finite interval (in n-dimension it is a box). Single variable optimization in a finite interval is important because of its application is in multi-variable optimisation. In this chapter we will consider one dimensional optimisation.

If one needs to find the maximum or minimum (i.e. the optimal) value of a function $f(x)$ on the interval $[a,b]$ the procedure would be:

1. Find all turning (stationary) points of $f(x)$ (assuming $f(x)$ is differentiable) on $[a,b]$ and then decide the optimum.
2. Find the optimal turning point of $f(x)$ on $[a,b]$.

Generally it may be difficult/impossible/tiresome to implement (i) analytically, so we resort to the computer and an appropriate numerical method to find an optimal (hopefully the best estimate!) solution of an univariate function. In the next section we introduce some numerical techniques. The numerical approach is mandatory when the function $f(x)$ is not given explicitly.

In many cases when one would like to find the minimiser of a function $f(x)$ but neither $f(x)$ nor $f'(x)$ are given (or known) explicitly, then the numerical approaches viz : polynomial interpolations or function comparison methods are used. These are the univariate optimisation used as **line search** in multivariate optimisation.

4.1 Techniques Using Function Evaluations

4.1.1 Bisection Method

We assume that an interval $[a,b]$ is given and that a local minimum $x^* \in [a,b]$. When the first derivative of the objective function $f(x)$ is known at a and b, it is necessary to evaluate function information at only one interior point in order to reduce this interval. This is because it is possible to decide if any interval brackets a minimum simply by looking at the function values $f(a), f(b)$ and $f'(a), f'(b)$ at extreme points a and b. The conditions that to be satisfied are:

- $f'(a) < 0$ and $f'(b) > 0$.
- $f'(a) < 0$ and $f(b) > f(a)$.
- $f'(a) > 0$ and $f'(b) > 0$ and $f(b) < f(a)$.

These three situations are illustrated in the Figure below. The next step of the bisection method is to reduce the interval. At the k-th iteration we have an interval $[a_k, b_k]$ and the mid-point $c_k = \frac{1}{2}(a_k + b_k)$ is computed. The next interval will be called $[a_{k+1}, b_{k+1}]$ which is either $[a_k, c_k]$ or $[c_k, b_k]$ depending on which interval brackets the minimum. The process continues until two consecutive interval produces minima which are within an acceptable tolerance.

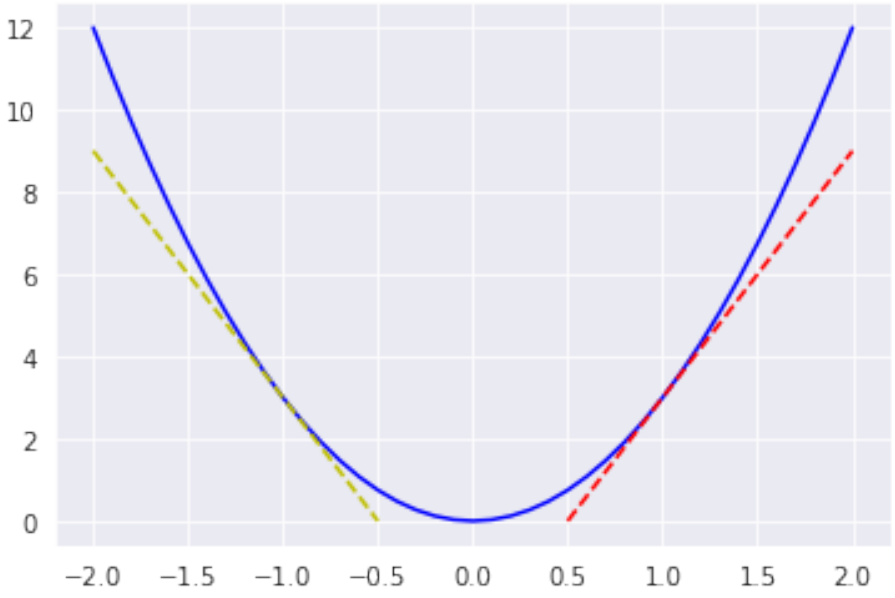


Figure 4.1: Condition 1

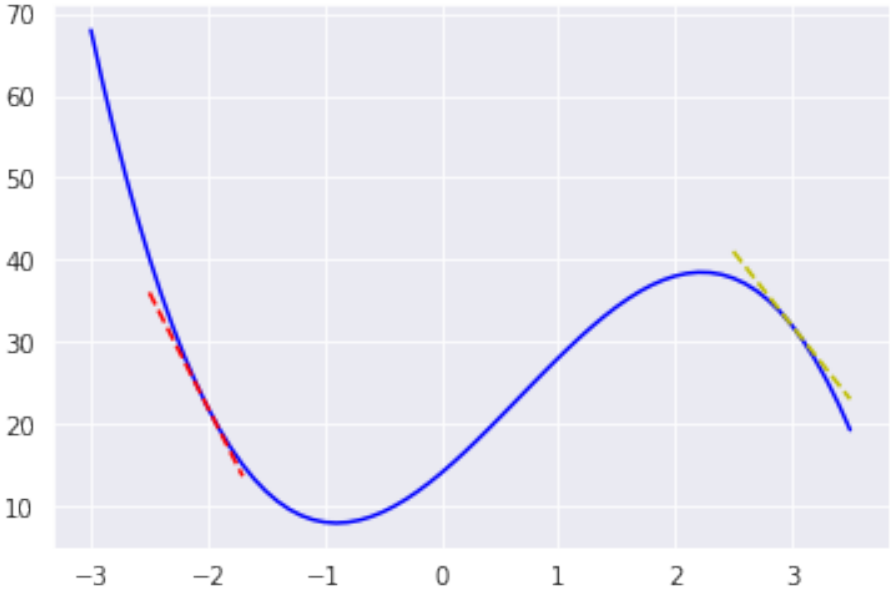


Figure 4.2: Condition 2

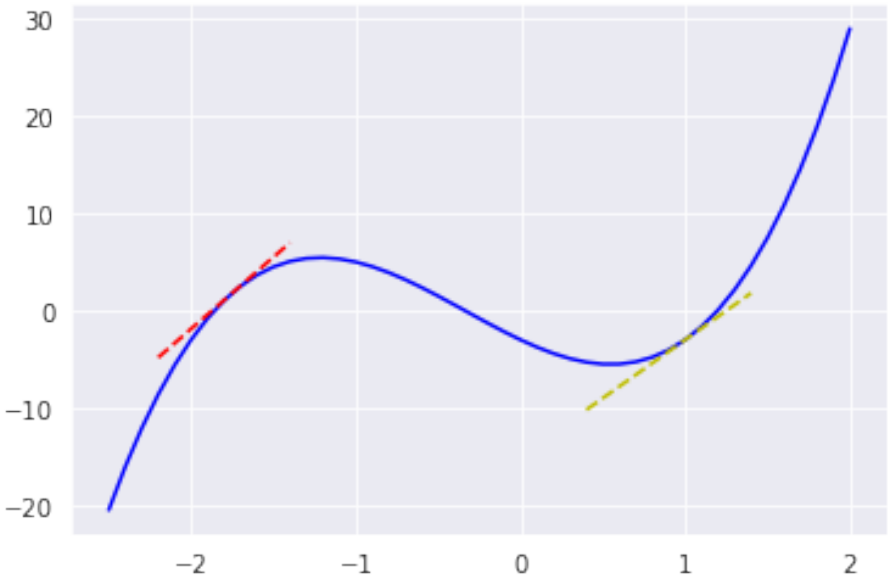


Figure 4.3: Condition 3

4.1.1.1 Exercise

Find the minimum value of:

$$f(x) = -\frac{1}{3}x^3 - \frac{1}{2}x^2 + 2x - 5,$$

over the domain $[-3,\ 0]$ using the bisection method. The problem has a minimum of value of -8.33 at $x = 2$.

4.1.2 Golden Search Method

Suppose $f : \mathbb{R} \to \mathbb{R}$ on the interval $[a, b]$ and f has only one minimum (we say f is *unimodal* at x^*. The problem is to locate x^*. The method we now discuss is based on evaluating the objective function at different points in the interval $[a, b]$. We choose these points in such a way that an approximation to the minimiser of f may be achieved in as few evaluation as possible. Our goal is to progressively narrow down the range of the subinterval containing x^*. If we evaluate f at only one intermediate point of the interval $[a, b]$, we cannot narrow the range within which we know the minimiser is located. We have to evaluate f at two intermediate points in such a way that the reduction in the range is symmetrical, such that $\rho = \dfrac{x_1 - a}{b - a}$ and $\rho = \dfrac{b - x_2}{b - a}$. We then evaluate f at the intermediate points.
Case I:
If $f(x_1) > f(x_2)$, then the minimiser located in the range $[a, x_1]$. Then we need to update the interval and calculate an update the interior points at the next iteration.
Case II:
If, on the other hand, $f(x_1) < f(x_2)$, then the minimiser must lie in the range $[x_2, b]$. Then we need to update the interval and calculate and the updated interior points at the next iteration.

Starting with the reduced range of uncertainty we can repeat the process and similarly find two new interior point, respectively. We would like to minimise the number of function evaluations while reducing the width of the interval of uncertainty. Suppose that $f(x_1) < f(x_2)$. Then we know that $x^* \in [x_2, b]$. Because x_1 is already in the uncertainty interval and $f(x_1)$ is known, we can use these information. We can make x_2 coincide with x_1. Thus, only one new evaluation of f at x_1 would be necessary.

If $f(x_1) > f(x_2)$, then the minimiser must lie in the range $[a, x_1]$. Because x_2 is already in the uncertainty interval and $f(x_2)$ is known, we can use these information. We can make x_1 coincide with x_2. Thus, only one new evaluation of x_2 and the corresponding function value $f(x_1)$ would be necessary.

We let the L_0 represent the span of the interval, that is $L_0 = b - a$, $L_1 = x_1 - a$, and $L_2 = b - x_1$. Then L_0 can also be seen as the sum of L_1 and L_2.

Figure 4.4: Golden Search Interval and Interior Points

Using the two conditions that $L_0 = L_1 + L_2$ and that $\frac{L_1}{L_0} = \frac{L_2}{L_1}$, using the quadratic formula we can deduce the ratio ρ to be equal to $\frac{\sqrt{5}-1}{2} = 0.681...$.

This forms the basis of a search algorithm since the technique is applied again on the reduced interval.

4.1.2.1 Example

Use the four iterations Golden Section search to find the value of x that minimizes:

$$f(x) = x^2 - 6x + 15,$$

on the domain $[0,\ 10]$.

Answer:

Iteration 1:

We evaluate f in two intermediate points x_1 and x_2. We have:

$$x_1 = a + \rho(b-a) = 6.18,$$
$$x_2 = b - \rho(b-a) = 3.82.$$

We compute

$$f(x_1) = 16.12,$$
$$f(x_2) = 6.67.$$

Thus we have $f(x_1) > f(x_2)$, and so the uncertainty interval is reduced to $[a, x_1] = [0, 6.18]$.

Iteration 2:

We choose x_1 to coincide with x_2, and f need only to be evaluated at one new point

$$x_2 = a + \rho(b-a) = 2.36.$$

Now we have:

$$f(x_1) = 6.67,$$
$$f(x_2) = 6.41.$$

Now, $f(x_1) > f(x_2)$, and so the uncertainty interval is reduced to $[a, x_1] = [0, 3.82]$.

Iteration 3:

We set $x_1 = x_2$, and compute x_2:

$$x_2 = 1.46.$$

We have:

$$f(x_1) = 6.41,$$
$$f(x_2) = 8.37.$$

So we have $f(x_1) < f(x_2)$. Hence, the new interval is $[x_2, b] = [1.46, 3.82]$.

Iteration 4:

We set $x_2 = x_1$, and compute x_1:

$$x_1 = 2.92.$$

We have:

$$f(x_1) = 6.01,$$
$$f(x_2) = 6.41.$$

Since $f(x_1) < f(x_2)$. Thus the value of x that minimizes f is located in the interval

$$[2.36, 3.82].$$

Golden Search Pseudocode

```
Inputs [a, b], iterations and R = (sqrt(5)-1)/2
  x1 = a + R(b-a)
  x2 = b - R(b-a)
  for i in range(iterations):
      f_x1 = f(x1)
      f_x2 = f(x2)
      if f_x1>f_x2:
          b = x1
          x1 = x2
          x2 = b - R(b-a)
          f_x1 = f(x1)
          f_x2 = f(x2)
      else if f_x1<f_x2:
          a = x2
          x2 = x1
          x1 = a + R(b-a)
          f_x1 = f(x1)
          f_x2 = f(x2)
      end
  end
Return (b+a)/2
```

4.1.3 Exercises

1. Find the minimum value of the one dimensional function $f(x) = x^2 - 3x\exp(-x)$, over $[0, 1]$, using:
   - Bisection Method
   - Golden Search Method

Chapter 5

Multivariate Unconstrained Optimisation

Unconstrained optimisation is optimisation when we know we do not have to worry about the boundaries of the feasible set.

$$\min_{s.t.\ x \in S} \quad f(x) \tag{5.1}$$

where S is the feasible set. It should then be possible to find local minima and maxima just by looking at the behaviour of the objective function; and indeed sufficient and necessary conditions. In this chapter these conditions will be derived. The idea of a line in a particular direction is important for any unconstrained optimization methods, we discuss this and derive the slope and curvature of the function f at a point on the line.

5.1 Terminology for Functions of Several Variables

For a function $f(\mathbf{x}) \in \mathbb{R}^n$ there exists, at any point $\mathbf{x}$ a vector of first order partial derivatives, or gradient vector:

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \dfrac{\partial f}{\partial x_1}(\mathbf{x}) \\ \dfrac{\partial f}{\partial x_2}(\mathbf{x}) \\ \vdots \\ \dfrac{\partial f}{\partial x_n}(\mathbf{x}) \end{bmatrix} = \mathbf{g}(\mathbf{x}). \tag{5.2}$$

It can be shown that if the function $f(\mathbf{x})$ is smooth, then at the point $\mathbf{x}$ the gradient vector $\nabla f(\mathbf{x})$ (denoted by $g(\mathbf{x})$) is always perpendicular to the contours (or surfaces

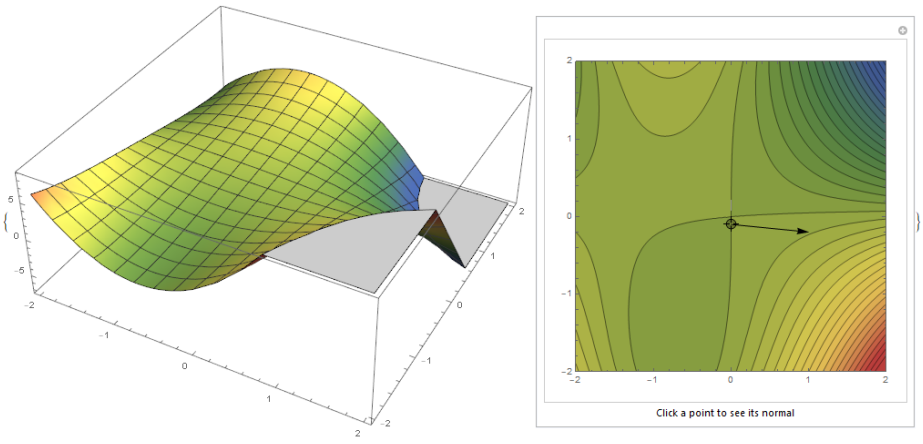


Figure 5.1: Mathematica Demo of Gradient.

of constant function value) and is the **direction of maximum increase** of $f(\mathbf{x})$ as seen in the Figure above.

If $f(\mathbf{x})$ is twice continuously differentiable then at the point $\mathbf{x}$ there exists a matrix of second order partial derivatives called the **Hessian matrix**:

$$\mathbf{H}(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2}(\mathbf{x}) & \frac{\partial^2 f}{\partial x_1 \partial x_2}(\mathbf{x}) & \dots & \frac{\partial^2 f}{\partial x_1 \partial x_n}(\mathbf{x}) \\ \frac{\partial^2 f}{\partial x_2 \partial x_1}(\mathbf{x}) & \ddots & & \\ \frac{\partial^2 f}{\partial x_n \partial x_1}(\mathbf{x}) & \dots & & \frac{\partial^2 f}{\partial x_n^2}(\mathbf{x}) \end{bmatrix} = \nabla^2 f(\mathbf{x}) \tag{5.3}$$

5.1.0.1 Example

Let $f(x_1, x_2) = 5x_1 + 8x_2 + x_1x_2 - x_1^2 - 2x_2^2$. Then:

$$\nabla f(\mathbf{x}) = \begin{bmatrix} 5 + x_2 - 2x_1 \\ 8 + x_1 - 4x_2 \end{bmatrix},$$

and

$$\nabla^2 f(\mathbf{x}) = \begin{bmatrix} -2 & 1 \\ 1 & -4 \end{bmatrix}.$$

Definition 5.1 (Feasible Direction)**.** A vector $d \in \mathbb{R}^n$, $d \neq 0$, is a **feasible direction** at $\mathbf{x} \in S$ if there exists $\alpha_0 > 0$ such that $x + \alpha d \in S$ for all $\alpha \in [0, \alpha_0]$.

Definition 5.2 (Directional Derivative)**.** Let $f : \mathbb{R}^n \to \mathbb{R}$ be a real-valued function and let d be a feasible direction at $\mathbf{x} \in S$. The **directional derivative** of f in the direction of d, denoted by $d^T \nabla f(\mathbf{x})$, is given by:

$$\nabla f^T d = \lim_{\alpha \to 0} \frac{f(\mathbf{x} + \alpha d) - f(\mathbf{x})}{\alpha} \tag{5.4}$$

If $\|d\| = 1$, then $d^T \nabla f(\mathbf{x})$ is the rate of increase of f at $\mathbf{x}$ in the direction d. To compute the above directional derivative, suppose that $\mathbf{x}$ and d are given. Then, $f(\mathbf{x}+\alpha d)$ is a function of α, and:

$$d^T \nabla f(\mathbf{x}) = \frac{d}{d\alpha} f(\mathbf{x} + \alpha d)\big|_{\alpha=0}. \tag{5.5}$$

5.2 A Line in a Particular Direction in the Context of Optimisation

A line is a set of points $\mathbf{x}$ such that:

$$\mathbf{x} = \mathbf{x}' + \alpha \mathbf{d}, \quad \forall\ \alpha, \tag{5.6}$$

where $\mathbf{d}$ and $\mathbf{x}'$ are given. For $\alpha \geq 0$ Equation (5.6) is a half-line. The point $\mathbf{x}'$ is a fixed point (corresponding to $\alpha = 0$) along the line, $\mathbf{d}$ is the direction of the line. For instance, if we take the fixed point $\mathbf{x}'$ to be $(2,2)^T$ and the direction $\mathbf{d} = (3,1)^T$ then the Figure below shows the line in the direction of $\mathbf{d}$.

The vector $\mathbf{d}$ in indicated by the arrow. If we normalise the vector $\mathbf{d}$ so that $\mathbf{d}^T\mathbf{d} = \sum_i d_i^2 = 1$. This does not change the line, but only the value of α associated with any point along the line. For Example:

```
d = [3, 1]
alpha = norm(d)
print('Alpha is:')
3.1622776601683795
norm_d = d/alpha
print('The normalised vector d is:')
[0.9486833  0.31622777]
print('The normalised d^Td gives:f', 0.9999999999999999)
print('So alpha x normalised d returns d:')
[3. 1.]
```

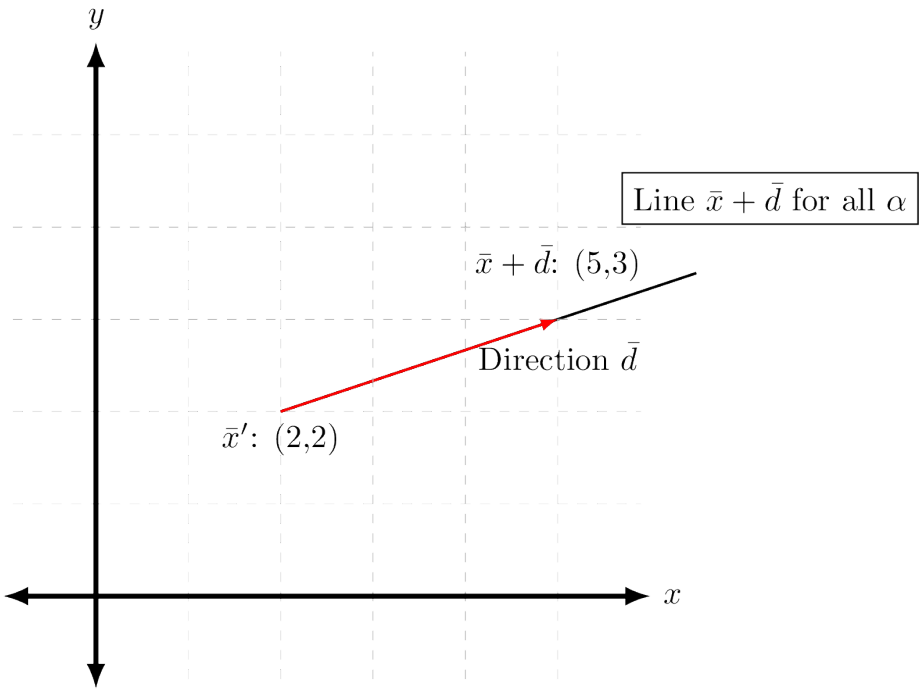


Figure 5.2: An Example of a Line in a Particular Direction.

We now use the gradient and the Hessian of $f(\mathbf{x})$ to derive the derivative of $f(\mathbf{x})$ along a line of any direction. For a fixed line of a given direction like Equation (5.6) we see that the points on the line is a function of α only. Hence a change in α causes change in all coordinates of $\mathbf{x}(\alpha)$. The derivative of $f(\mathbf{x})$ with respect to α :

$$\frac{df(\mathbf{x}(\alpha))}{d\alpha} = \frac{\partial f(\mathbf{x}(\alpha))}{\partial x_1}\frac{dx_1(\alpha)}{d\alpha} + \frac{\partial f(\mathbf{x}(\alpha))}{\partial x_2}\frac{dx_2(\alpha)}{d\alpha} + \cdots + \frac{\partial f(\mathbf{x}(\alpha))}{\partial x_n}\frac{dx_n(\alpha)}{d\alpha} \tag{5.7}$$

The Equation (5.7) represents the derivative of $f(\mathbf{x})$ at any point $\mathbf{x}(\alpha)$ along the line. The operator $\frac{d}{d\alpha}$ can be expressed as:

$$\frac{d}{d\alpha} = \frac{\partial}{\partial x_1}\frac{dx_1}{d\alpha} + \frac{\partial}{\partial x_2}\frac{dx_2}{d\alpha} + \cdots + \frac{\partial}{\partial x_n}\frac{dx_n}{d\alpha} = \mathbf{d}^T\nabla \tag{5.8}$$

The slope of $f(\mathbf{x})$ at $\mathbf{x}(\alpha)$ can be written as:

$$\frac{df}{d\alpha} = \mathbf{d}^T\nabla f(\mathbf{x}(\alpha)) = \nabla f(\mathbf{x}(\alpha))^T\mathbf{d}. \tag{5.9}$$

Likewise, the curvature of $f(\mathbf{x}(\alpha))$ along the line:

$$\frac{d^2f}{d\alpha^2} = \frac{d}{d\alpha}\left(\frac{df(\mathbf{x}(\alpha))}{d\alpha}\right) = \mathbf{d}^T\nabla\left(\nabla f^T\mathbf{d}\right) = \mathbf{d}^T\nabla^2 f\mathbf{d}, \tag{5.10}$$

where ∇f and $\nabla^2 f$ are evaluated at $\mathbf{x}(\alpha)$. These (slope and curvature) when evaluated at α=0 are respectively known as derivative (also called slope since $f = f(\alpha)$ is now a function of the single variable α) and curvature of f at x' in the direction of d.

5.2.0.1 Example

Let us consider the Rosenbrock's function:

$$f(\mathbf{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \tag{5.11}$$

If $\mathbf{x}' = \binom{0}{0}$ then show that the slope of $f(\mathbf{x})$ along the line generated by $\mathbf{d} = \binom{1}{0}$ is $\mathbf{d}^T\nabla f = -2$ and the curvature is $\mathbf{d}^T G\mathbf{d} = 2$ where $G = \nabla^2 f(\mathbf{x}')$.

Solution:

$$\nabla f = \begin{bmatrix} -400x_1(x_2 - x_1^2) - 2(1 - x_1) \\ 200(x_2 - x_1^2) \end{bmatrix} = \begin{bmatrix} -2 \\ 0 \end{bmatrix}$$

Therefore $\mathbf{d}\nabla f = [1 \quad 0] \times [-2 \quad 0]^T = -2$. Next:

$$\nabla^2 f = \begin{bmatrix} -400(x_2 - x_1^2) + 800x_1^2 + 2 & -400x_1 \\ -400x_1 & 200 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 200 \end{bmatrix}$$

Thus $\mathbf{d}^T G\mathbf{d} = [1 \quad 0] \times \begin{bmatrix} 2 & 0 \\ 0 & 200 \end{bmatrix} \times [1 \quad 0]^T = 2$

These definitions of slope and curvature depend on the size of $\mathbf{d}$, and this ambiguity can be resolved by requiring that $\|\mathbf{d}\| = 1$. Hence Equation (5.9) is the directional derivative in the direction of a unit vector $\mathbf{d}$ and this given by $\nabla f(\mathbf{x})\mathbf{d}^T$. Likewise the curvature along the line in the direction of the unit vector is given by $\mathbf{d}^T\nabla^2 f(\mathbf{x})\mathbf{d}$.

Since $\mathbf{x}(\alpha) = \mathbf{x}' + \alpha\mathbf{d}$, at $\alpha = 0$ we have $\mathbf{x}(0) = \mathbf{x}'$. Therefore, the function value $f(\mathbf{x}(0)) = f(\mathbf{x}')$, the slope at $\alpha = 0$ in the direction of $\mathbf{d}$ is $f'(0) = \mathbf{d}^T\nabla f(\mathbf{x}')$ and the curvature at $\alpha = 0$ is $f''(0) = \mathbf{d}^T G(\mathbf{x}')\mathbf{d}$.

5.3 Taylor Series for Multivariate Function

In the context of optimization involving smooth function $f(\mathbf{x})$ the Taylor series is indispensable. Since $\mathbf{x} = \mathbf{x}(\alpha) = \mathbf{x}' + \alpha\mathbf{d}$ for a fixed point $\mathbf{x}'$ and a given direction $\mathbf{d}$, the $f(\mathbf{x})$ at $\mathbf{x}(\alpha)$ becomes a function of the single variable α. Hence, $f(\mathbf{x}) = f(\mathbf{x}(\alpha)) = f(\alpha)$. Therefore, expanding the Taylor series around zero we have:

$$f(\alpha) = f(0+\alpha) = f(0) + \alpha f'(0) + \frac{1}{2}\alpha^2 f''(0) + \cdots \tag{5.12}$$

But $f(\alpha) = f(\mathbf{x}' + \alpha\mathbf{d})$ is the value of the function $f(\mathbf{x})$ of many variable along the line $\mathbf{x}(\alpha)$. Hence, we can re-write Equation (5.12) as:

$$f(\mathbf{x}' + \alpha\mathbf{d}) = f(\mathbf{x}') + \alpha\mathbf{d}^T\nabla f(\mathbf{x}') + \frac{1}{2}\alpha^2\mathbf{d}^T\left[\nabla^2 f(\mathbf{x}')\right]\mathbf{d} + \cdots \tag{5.13}$$

5.4 Quadratic Forms

The quadratic function in n variables may be written as:

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{b}^T\mathbf{x} + c, \tag{5.14}$$

where $c \in \mathbb{R}$, $\mathbf{b}$ is a real n vector and $\mathbf{A}$ is a $n \times n$ real matrix that can be chosen in a non-unique manner. It is usually chosen symmetrical in which case it follows that:

$$a_{11}{x_1}^2 + 2a_{12}x_1x_2 + a_{22}{x_2}^2 + 2a_{13}x_1x_3 + \cdots + a_{nn}{x_n}^2 = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_ix_j, \tag{5.15}$$

and:

$$\nabla f(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}; \quad \mathbf{H}(\mathbf{x}) = \mathbf{A}. \tag{5.16}$$

The form A is said to be positive definite if $A \geq 0$ for all $\mathbf{x}$ with $A = 0$ iff $\mathbf{x} = 0$. The form A is said to be positive semi-definite if $A \geq 0$ for all $\mathbf{x}$. Similar definitions apply to negative definite and negative semi-definite with the inequalities reversed.

5.4.0.1 Example

Write $A(\mathbf{x}) = {x_1}^2 + 5x_1x_2 + 4{x_2}^2$ in the matrix form.

Solution: $A(\mathbf{x}) = (x_1, x_2)\begin{pmatrix} 1 & \frac{5}{2} \\ \frac{5}{2} & 4 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$

5.5 Stationary Points

In the following chapters we will be concerned with gradient based minimization methods. Therefore, we only consider the minimization of smooth functions. We will not consider the non-smooth minima as they do not satisfy the same conditions as smooth minima. We, however, will consider the case of saddle point. Hence, we assume that the first and the second derivative exist.

We can classify definiteness by looking at the eigenvalues of $\nabla^2 f(\mathbf{x})$. Specifically:

- If $\nabla^2 f(\mathbf{x}^*)$ is indefinite, i.e. all λ_i are mixed sign, then $\mathbf{x}^*$ is a saddle point.
- If $\nabla^2 f(\mathbf{x}^*)$ is positive definite, i.e. all $\lambda_i > 0$, then $\mathbf{x}^*$ is a minimum.
- If $\nabla^2 f(\mathbf{x}^*)$ is negative definite, i.e. all $\lambda_i < 0$, then $\mathbf{x}^*$ is a maximum.
- If $\nabla^2 f(\mathbf{x}^*)$ is positive semi-definite, i.e. all $\lambda_i \geq 0$, then $\mathbf{x}^*$ is a half cylinder.

These can be seen in the Figure below:

In summary:

Let $G = \nabla^2 f(\mathbf{x})$, i.e. the Hessian.

- $G(x)$ is positive semi-definite if $x^T Gx \geq 0, \quad \forall x$
- $G(x)$ is negative semi-definite if $x^T Gx \leq 0, \quad \forall x$
- $G(x)$ is positive definite iff $x^T Gx > 0, \quad \forall x \neq 0$
- $G(x)$ is negative definite iff $x^T Gx < 0, \quad \forall x \neq 0$
- $G(x)$ is indefinite iff $x^T Gx$ is mixed negative and positive

and:

- $f(x)$ is concave iff $G(x)$ is negative semi-definite
- $f(x)$ is strictly concave iff $G(x)$ is negative definite
- $f(x)$ is convex iff $G(x)$ is positive semi-definite
- $f(x)$ is convex iff $G(x)$ is positive definite

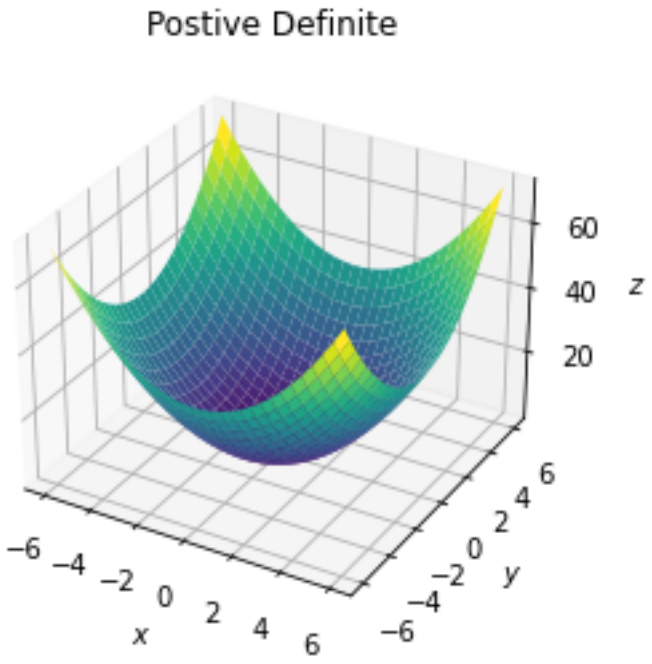


Figure 5.3: Positive Definite.

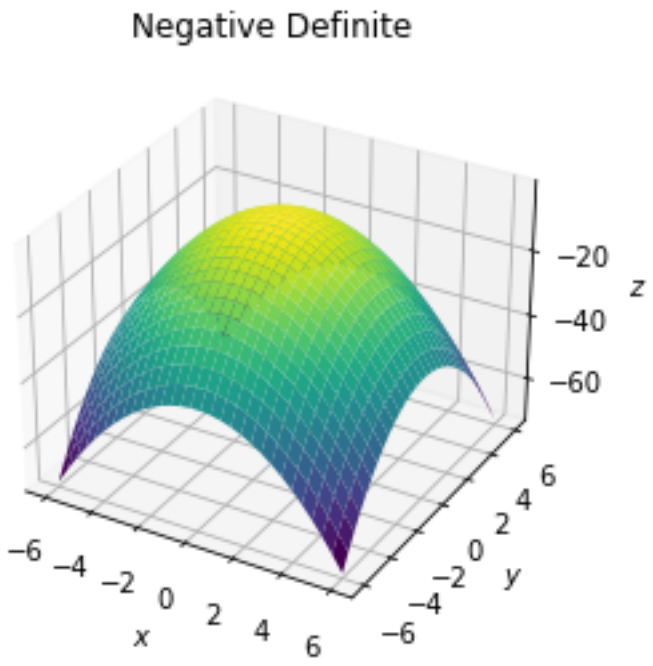


Figure 5.4: Negative Definite.

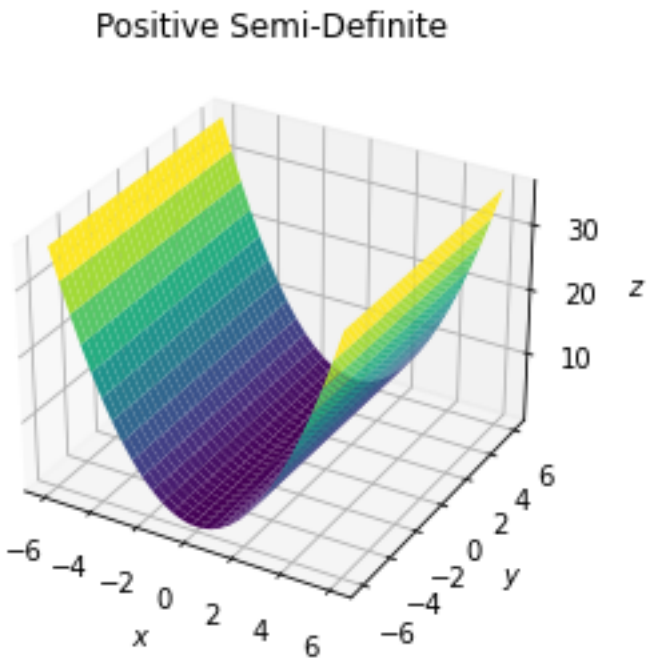


Figure 5.5: Positive Semi Definite.

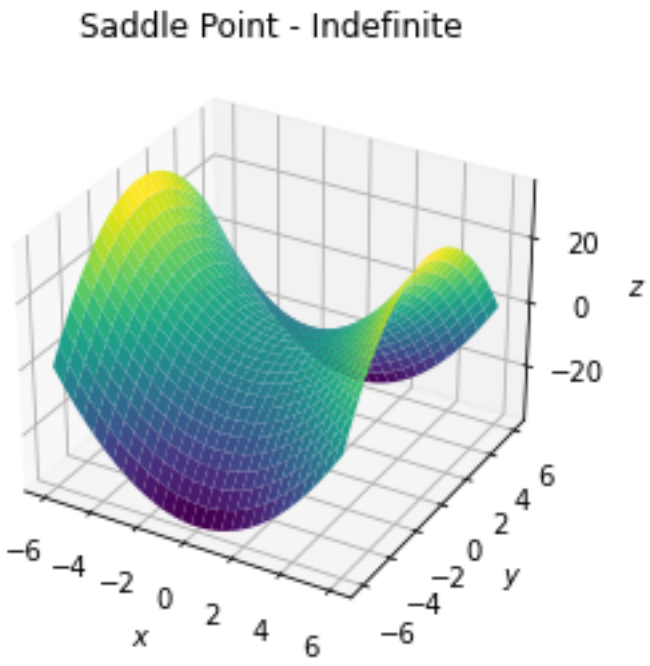


Figure 5.6: Indefinite.

5.5.1 Tests for Positive Definiteness

There are a number of ways for us to test for positive or negative definiteness. Namely;

5.5.1.1 Compute the Eigenvalues

5.5.1.1.1 Example Classify the stationary points of the function

$$f(\mathbf{x}) = 2x_1^2 + x_1x_2^2 + x_2^2.$$

Solution:

The stationary points are the solutions of

$$\begin{aligned} \frac{\partial f}{\partial x_1} &= 4x_1 + {x_2}^2 = 0 \\ \frac{\partial f}{\partial x_2} &= 2x_1x_2 + 2x_2 = 0 \end{aligned}$$

which gives $\mathbf{x}_1 = (0,0)^T, \mathbf{x}_2 = (-1,2)^T$ and $\mathbf{x}_3 = (-1,-2)^T$. The Hessian matrix is:

$$G = \begin{pmatrix} 4 & 2x_2 \\ 2x_2 & 2x_1 + 2 \end{pmatrix}$$

Thus:

$$G_1 = \begin{pmatrix} 4 & 0 \\ 0 & 2 \end{pmatrix}$$

The eigenvalues are the solution of

$$(4-\lambda)(2-\lambda) = 0$$

which gives $\lambda = 4, 2$. Thus $\mathbf{x}_1$ correspond to a minimum. Similarly:

$$G_2 = \begin{pmatrix} 4 & 4 \\ 4 & 0 \end{pmatrix}$$

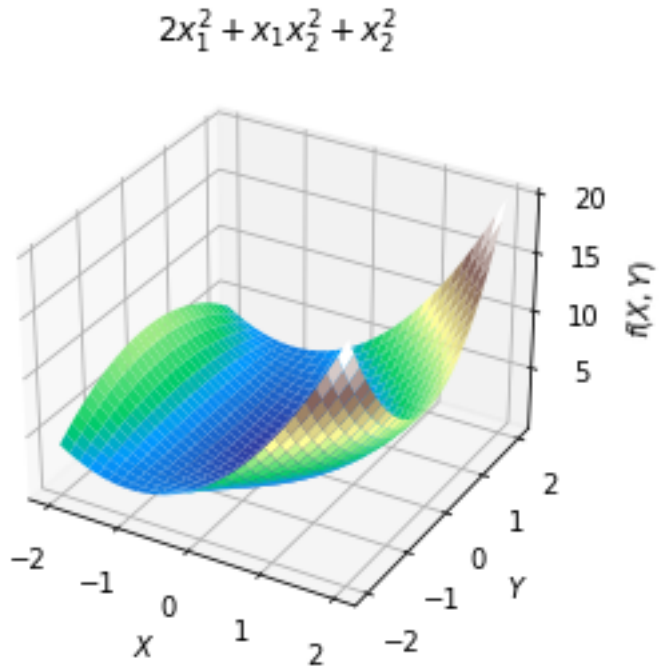
has eigenvalues:

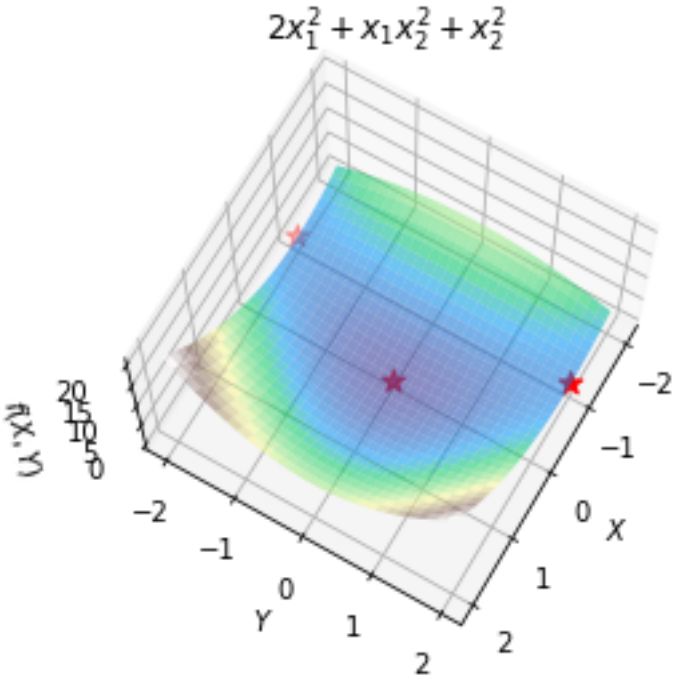
$$\lambda = 2 + \sqrt{20}, 2 - \sqrt{20}$$

Thus $\mathbf{x}_2$ correspond to a saddle point. Finally

$$G_3 = \begin{pmatrix} 4 & -4 \\ -4 & 0 \end{pmatrix}$$

has the same eigenvalues as G_2 and therefore $\mathbf{x}_3$ corresponds to a saddle point.


$2x_1^2 + x_1x_2^2 + x_2^2$
20
15
10
5
f(X, Y)
−2
−1
0
1
2
X
Y


$2x_1^2 + x_1x_2^2 + x_2^2$
f(X, Y)
20
15
10
5
0
−2
−1
0
1
2
Y
X

5.5.1.2 Principle Minors

From the Hessian we can compute the determinant of all subminors. If these are all greater than zero, then the Hessian is positive definite. Utilising the example above. If:

$$G_1 = \begin{pmatrix} 4 & 0 \\ 0 & 2 \end{pmatrix}$$

Then the first subminor is just det|4| which is > 0. The second and final subminor is the entire matrix, so:

$$\det \begin{vmatrix} 4 & 0 \\ 0 & 2 \end{vmatrix} = 8 - 0 > 0.$$

Therefore G_1 is positive definite. G_2 and G_3 are dealt with similarly. However, to prove negative definiteness we need to prove $(-1)^k D_k > 0$, where D is the determinant of the k-th principle minor.

This approach would be preferable when dealing with the case of large matrices.

5.6 Necessary and Sufficient Conditions

Theorem 5.1 (First Order Necessary Condition (FONC) for Local Maxima/Minima)**.** *If $f(\mathbf{x})$ has continuous first partial derivatives at all points of $S \subset R^n$ and if $\mathbf{x}^*$ is an interior point of the feasible set S then $\mathbf{x}^*$ is a local minimum or maximum of $f(\mathbf{x})$:*

$$\nabla f(x^*) = 0. \tag{5.17}$$

Alternatively, if $\mathbf{x}^* \in S$ is a local minimum or maximum, then at the point $\mathbf{x}^*$:

$$\frac{\partial f(x^*)}{\partial x_i} = 0; \quad i = 1, 2, \ldots n. \tag{5.18}$$

Theorem 5.2 (Second Order Necessary Condition (SONC) for Local Maxima/Minima)**.** *Let f be twice continuously differentiable on the feasible set S, $\mathbf{x}^*$ is a local minimiser of $f(\mathbf{x})$, and $\mathbf{d}$ is a feasible direction at $\mathbf{x}^*$. If $\mathbf{d}^T \nabla f(\mathbf{x}^*) = 0$, then:*

$$\mathbf{d}^T \nabla^2 f(\mathbf{x}^*) \mathbf{d} \geq 0. \tag{5.19}$$

Theorem 5.3 (Second Order Sufficient Condition (SOSC) for Strong Local Maxima/Minima)**.** *Let $\mathbf{x}^*$ be an interior of S. If $\mathbf{x}^*$ is a local minimiser of $f(\mathbf{x})$ then (i) $\nabla f(\mathbf{x}^*) = 0$ and (ii) $\mathbf{d}^T \nabla^2 f(\mathbf{x}^*) \mathbf{d} > 0$. That is the hessian is positive definite.*

5.6.0.1 Example

Let $f(\mathbf{x}) = x_1^2 + x_2^2$. Show that $\mathbf{x} = (0,0)^T$ satisfies the FONC, the SONC and SOSC hence $(0,0)^T$ is a strict local minimiser. We see that $\nabla f(\mathbf{x}) = (2x_1, 2x_2) = 0$ if and only if $x_1 = x_2 = 0$. It also can be easily shown that for all $\mathbf{d} \neq 0$, $\mathbf{d}^T \nabla^2 f(x)\mathbf{d} = 2d_1^2 + 2d_2^2 > 0$. Hence $\nabla^2 f(\mathbf{x})$ is positive definite.

5.6.0.2 Example

$$f(x_1, x_2) = x_1^4 + x_2^4$$

$\nabla f(\mathbf{x}) = \begin{pmatrix} 4x_1^3 \\ 4x_2^3 \end{pmatrix}$. The only stationary point is $(0\ 0)^T$. Now the Hessian $\nabla^2 f = \begin{pmatrix} 12x_1^2 & 0 \\ 0 & 12x_2^2 \end{pmatrix}$. At the origin the Hessian is $\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$ and so there is no prediction of the minimum from the test although it is easy to see that the origin is a minimum.

5.6.0.3 Example

$$f(x_1, x_2) = \frac{1}{2c}\left(\frac{x_1^2}{a^2} + \frac{x_2^2}{b^2}\right),$$

where a, b, and c are constants. $\nabla f(\mathbf{x}) = \begin{pmatrix} \frac{x_1}{ca^2} \\ \frac{-x_2}{cb^2} \end{pmatrix}$. So the only stationary point is $(0\ 0)^T$. The Hessian is $\nabla^2 f(\mathbf{x}) = \begin{pmatrix} \frac{1}{ca^2} & 0 \\ 0 & -\frac{1}{cb^2} \end{pmatrix}$. This is clearly indefinite and hence $(0\ 0)^T$ is a saddle point.

Thus in summary, the *necessary and sufficent condition* for $\mathbf{x}^*$ to be a **strong** local minimum are:

- $\nabla f(\mathbf{x}^*) = 0$
- Hessian is positive definite

5.6.1 Exercises

1. Find the gradient vectors of the following functions (where $\mathbf{x} \in \mathbb{R}^n$):
   - $f(\mathbf{x}) = \mathbf{c}^T\mathbf{x}, \quad \mathbf{c} \in \mathbb{R}^n$
   - $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{x}$
   - $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T G\mathbf{x}$ where G is symmetric
2. Find the slope and the curvature of the following function:
   - $f(\mathbf{x}) = 100(x_2 - x_1^2) + (1 - x_1)^2$ at $(0,0)^T$ in the direction of $(1,0)^T$.
3. Use the *necessary condition* of optimality to determine the optimiser of the following function
$$f(x_1, x_2) = (x_1 - 1)^2 + (x_2 - 1)^2 + x_1 x_2$$
4. For the following function, find the points where the gradients vanish, and investigate which of these are local minima, maxima or saddle.
   - $f(x_1, x_2) = x_1(1 + x_1) + x_2(1 + x_2) - 1$.
5. Consider the function $f : R^2 \to R$ determined by
$$f(x) = x^T \begin{bmatrix} 1 & 2 \\ 4 & 8 \end{bmatrix} x + x^T \begin{bmatrix} 3 \\ 4 \end{bmatrix} + 6$$
   - Find the gradient and Hessian of f at the point $(1,1)$.
   - Find the directional derivative of f at the point $(1,1)$ in the direction of the maximal rate of increase.
   - Find a point that satisfies the first order necessary condition (FONC). Does the point also satisfy the second order necessary condition (SONC) for a minimum?
6. Find the stationary points of the function
$$f(x_1, x_2) = \left({x_1}^2 - 4\right)^2 + {x_2}^2$$
   Show that f has an absolute minimum at each of the points $(x_1, x_2) = (\pm 2, 0)$. Show that the point $(0,0)$ is a saddle point.
7. Show that the point x^* on the line $x_2 - 2x_1 = 0$ is a weak global minimiser of
$$f(x) = 4{x_1}^2 - 4x_1 x_2 + {x_2}^2$$
8. Show that
$$f(x) = 3{x_1}^2 - {x_2}^2 + {x_1}^3$$
   has a strong local maximiser at $(-2,0)^T$ and a saddle point at $(0,0)^T$, but has no minimisers.
9. Prove that for a general quadratic function $f(\mathbf{x}) = c + \mathbf{b}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T G\mathbf{x}$, the Hessian G of f maps differences in position into differences in gradient, i.e., $\mathbf{g}^1 - \mathbf{g}^2 = G(\mathbf{x}^1 - \mathbf{x}^2)$.

Chapter 6

Gradient Methods for Unconstrained Optimisation

In this chapter we will study the methods for solving nonlinear unconstrained optimisation problems. The non-linear minimisation algorithms to be described here are iterative methods which generate a sequence of points, $\mathbf{x}^0, \mathbf{x}^1 \ldots$. say, or $\{\mathbf{x}^k\}$ (superscripts denoting iteration number), hopefully converging to a minimiser $\mathbf{x}^*$ of $f(\mathbf{x})$. Univariate minimisation along the line in a particular direction is known as the line search technique. One dimensional minimisation is known as line search subproblem in many variable unconstrained non-linear minimisation.

6.1 General Line Search Techniques used in Unconstrained Multivariate Minimisation

The algorithms for multivariate minimisation are all iterative processes which fit into the same general framework:

> At the beginning of the k-th iteration the current estimate of minimum is $f(\mathbf{x}^k)$, and a search is made in $\mathbb{R}^n$ from $\mathbf{x}^k$ along a given vector direction $\mathbf{d}^k$ ($\mathbf{d}^k$ is different for different minimization methods) in an attempt to find a new point $\mathbf{x}^{k+1}$ such that $f(\mathbf{x}^{k+1})$ is sufficiently smaller than $f(\mathbf{x}^k)$. This process is called line (or linear) search.

Line-search methods, therefore, generate the iterates by setting:

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha^k \mathbf{d}^k \tag{6.1}$$

where $\mathbf{d}^k$ is a search direction and $\alpha_k > 0$ is chosen so that:

$$f(\mathbf{x}^k + \alpha^k \mathbf{d}^k) = f(\mathbf{x}^{k+1}) < f(\mathbf{x}^k), \tag{6.2}$$

Therefore, for a given $\mathbf{d}^k$, a line-search procedure is used to choose an $\alpha_k > 0$ that approximately minimises f along the ray $x^k + \alpha^k d^k : \alpha^k > 0$. Hence, the line search is the univariate minimisation involving the single variable α^k (since both the $\mathbf{x}^k$ and $\mathbf{d}^k$) are known $f(\mathbf{x}^k + \alpha^k \mathbf{d}^k)$ becomes a function of α^k only) such that:

$$f(\alpha^k) = f(\mathbf{x}^k + \alpha^k \mathbf{d}^k). \tag{6.3}$$

Bear in mind that this single variable minimiser cannot always be obtained analytically and hence some numerical techniques may be necessary.

6.1.1 Challenges in Computing Step Length α^k

The challenges in finding a good α^k are both in avoiding a step length that is too long or too short. Consider the Figures below:

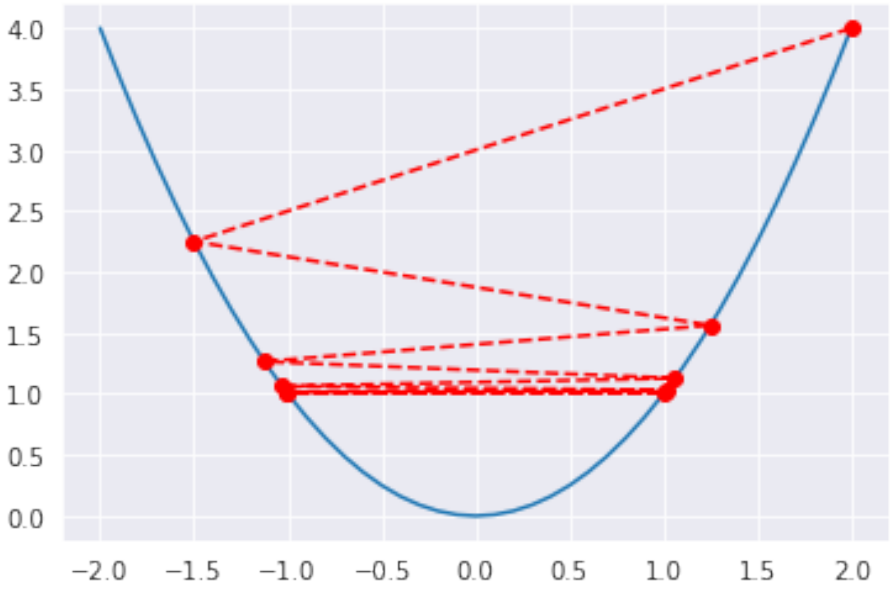


Figure 6.1: Step-size too big

Here the objective function is $f(x) = x^2$ and the iterates, $x^{k+1} = x^k + \alpha^k d^k$ are generated by the descent directions $d^k = (-1)^{k+1}$ with steps $\alpha^k = 2 + \frac{3}{2^{k+1}}$ with an initial starting point of $x_0 = 2$.

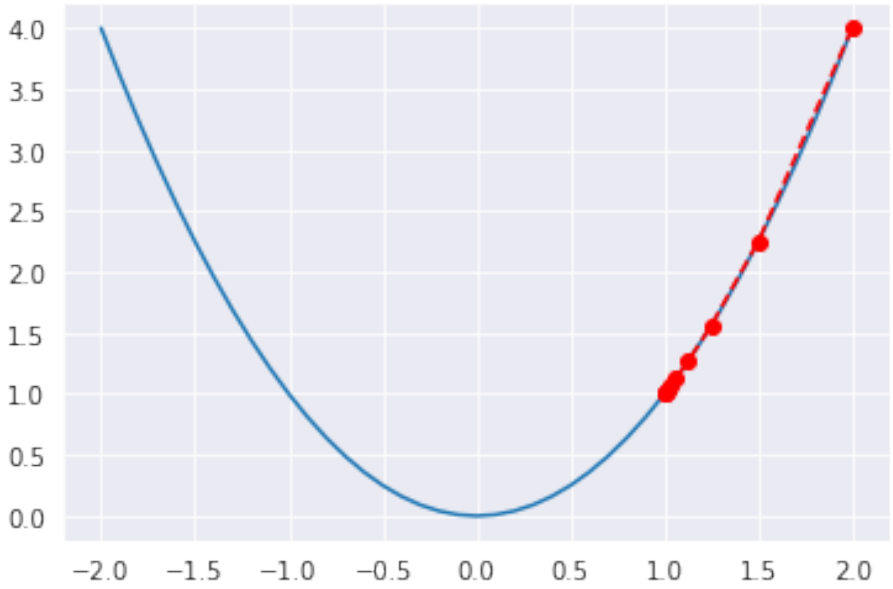


Figure 6.2: Step-size too small

Here the objective function is $f(x) = x^2$ and the iterates, $x^{k+1} = x^k + \alpha^k d^k$ are generated by the descent directions $d^k = (-1)$ with steps $\alpha^k = \frac{1}{2^{k+1}}$ with an initial starting point of $x_0 = 2$.

6.2 Exact and Inexact Line Search

Given the direction $\mathbf{d}^k$ and the point $\mathbf{x}^k$, $f(\mathbf{x}^k + \alpha\mathbf{d}^k)$ becomes a function of α. Hence it is simply a one dimensional minimisation with respect to α. The solution of $\frac{df(\alpha)}{d\alpha} = 0$ will determine the exact location of the minimiser α^k. However, it may not be possible to locate the exact location of α^k for which $\frac{df(\alpha)}{d\alpha} = 0$. It may even require very large number of iterations to locate the minimiser α^k. Nonetheless, the idea is conceptually useful. Notice that for exact line search the slope $\frac{df}{d\alpha}$ at α^k must be zero. Therefore, we get:

$$\frac{df(\mathbf{x}^{k+1})}{d\alpha} = \nabla f(\mathbf{x}^{k+1})^T \frac{d\mathbf{x}^{k+1}}{d\alpha} = \mathbf{g}(x^{k+1})^T \mathbf{d}^k = 0. \tag{6.4}$$

Line search algorithms used in practice are much more involved than the one dimensional search methods (optimisation methods) presented in the previous chapter. The reason for this stems from several practical considerations. First, determining the value of α_k that exactly minimises $f(\alpha)$ may be computationally demanding;

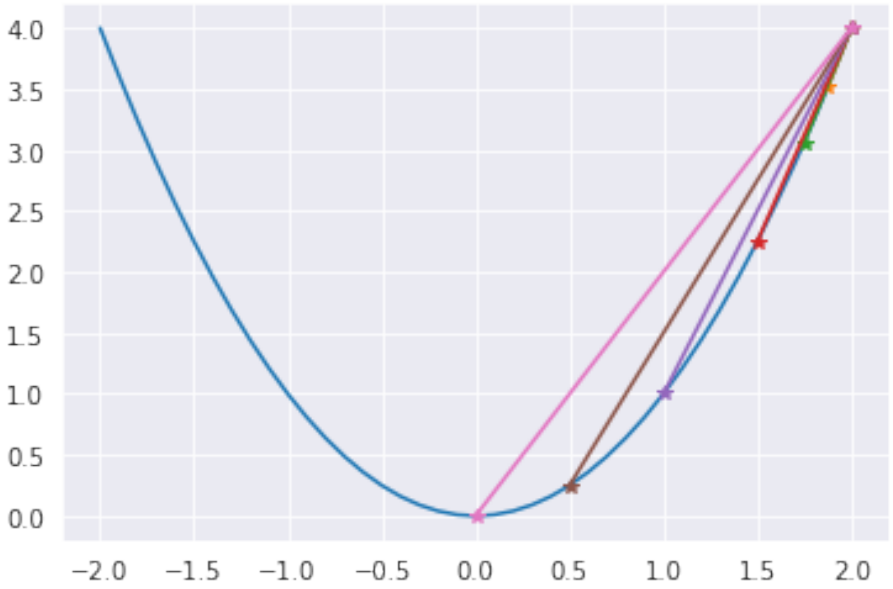


Figure 6.3: Varying Alpha

even worse, the minimiser of $f(\alpha)$ may not even exist. Second, practical experience suggests that it is better to allocate more computational time on iterating the optimisation algorithm rather than performing exact line searches. These considerations led to the development of conditions for terminating line search algorithms that would result in low-accuracy line searches while still securing a decrease in the value of f from one iteration to the next.

In practice, the line search is terminated when some descent conditions along the line $\mathbf{x}^k + \alpha\mathbf{d}^k$ are satisfied. Hence, it is no longer necessary to go for the exact line search. The line search carried out in this way is known as the inexact line search. A further justification for the inexact line search is that it is not efficient to determine the line search minima to a high accuracy when $\mathbf{x}^k$ is far from the minimiser $\mathbf{x}^*$. Under these circumstances, nonlinear minimisation algorithms employ an inexact or approximate line search. To sum up, exact line search relates to theoretical concept and the inexact is its practical implementation.

Remark:

Each iteration of a line search method computes a search direction $\mathbf{d}^k$ and then decides how far to move along that direction. The iteration is given by

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha^k\mathbf{d}^k,$$

where the positive scalar α^k is called the *step length*. The success of a line search method depends on effective choices of both the direction $\mathbf{d}^k$ and and the step length α^k. Most line search algorithms require $\mathbf{d}^k$ to be a descent direction.

6.2.1 Algorithmic Structure

The typical behaviour of a minimisation algorithm is that it repeatedly generates points $\mathbf{x}^k$ such that as k increases $\mathbf{x}^k$ moves close to $\mathbf{x}^*$. Features of a minimisation algorithm is that $f(\mathbf{x}^k)$ is always reduced on each iteration, which imply that the stationary point turns out to be a local minimiser. In a minimisation algorithm it is required to supply an initial estimate, say $\mathbf{x}^0$. At each iteration the algorithm finds a descent direction along which the function is minimised. This minimisation algorithm in a particular direction is known as *the line search.* The basic structure of the general algorithm is:

1. Initialise the algorithm with estimate $\mathbf{x_k}$. Initialise $k = 0$.
2. Determine a search direction $\mathbf{d}^k$ at $\mathbf{x}^k$.
3. Find α^k to minimise $f(\mathbf{x}^k + \alpha\mathbf{d}^k)$ with respect to α.
4. Set $\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha^k\mathbf{d}^k$.
5. Line search is stopped when $f(\mathbf{x}^{k+1}) < f(\mathbf{x}^k)$
6. If algorithm meets stopping criteria then **STOP**, **ELSE** set $k = k+1$ and go back to (2).

Different minimisation methods select $\mathbf{d}^k$ in different ways in (2). Steps (3&4) is the one dimensional *sub-problem* carried out along the line $\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha^k\mathbf{d}^k$ for $\alpha \in [0,1]$. The direction $\mathbf{d}^k$ at $\mathbf{x}^k$ must satisfy the descent condition.

6.3 The Descent Condition

Central to the development of the gradient based minimisation methods is the idea of a descend direction. Conditions for the descent direction can be obtained using Taylor series around the point $\mathbf{x}^k$. Using two terms of Taylor series we have:

$$f(\mathbf{x}^k + \alpha\mathbf{d}^k) - f(\mathbf{x}^k) = \alpha\mathbf{d^k}^T\nabla f(\mathbf{x}^k) + \cdots \tag{6.5}$$

Clearly the descent condition can easily be seen as:

$$\mathbf{d}^{k^T}\nabla f(\mathbf{x}^k) < 0, \tag{6.6}$$

since we require the left hand side of Equation (6.5) to be negative.

6.4 The Direction of Greatest Reduction

A simple line search descent method is the *steepest descent method* in which:

$$\mathbf{d}^k = -\nabla f(\mathbf{x}^k) = -\mathbf{g}^k, \quad \forall k \tag{6.7}$$

From Equation (6.5) we see that:

$$f^{k+1} - f^k = \alpha^k \mathbf{d}^{k^T} \mathbf{g}^k \tag{6.8}$$
$$= \alpha^k \|\mathbf{d}^k\| \|\mathbf{g}^k\| \cos\theta, \tag{6.9}$$

where θ can be interpreted geometrically as the angle between $\mathbf{d}^k$ and $\mathbf{g}^k$. If we allow θ to vary holding $\alpha^k, \|\mathbf{d}^k\|$ and $\|\mathbf{g}^k\|$ constant, then the right hand side of Equation (6.9) is most negative when $\theta = \pi$. Thus when α^k is sufficiently small, the greatest reduction in function is obtained in the direction:

$$\mathbf{d}^k = -\mathbf{g}^k \tag{6.10}$$

This negative gradient direction which satisfy the descent condition (6.10) gives rise to *the method of steepest descent.*

6.5 The Method of Steepest Descent

Here the the search direction is taken as the negative gradient and the step size, α_k, is chosen to achieve the maximum decrease in the objective function f at each step. Specifically we solve the problem:

$$\text{Minimise } f\left(\mathbf{x}^{(k)} - \alpha \nabla f\left(\mathbf{x}^{(k)}\right)\right) \text{ w.r.t. } \alpha \tag{6.11}$$

This is now a one-dimensional optimisation problem.

6.5.1 Steepest Descent Algorithm

Given $\mathbf{x}_0$, for all iterations $k = 1, 2, \ldots$ until stopping criterion is met, do:

- Compute gradient $\mathbf{g}(\mathbf{x}^k) = \nabla f(\mathbf{x}^k)$.
- Compute α^k such that $f(\mathbf{x}^k - \alpha^k \mathbf{g}^k) = \min_{\alpha} f(\mathbf{x}^k - \alpha \mathbf{g}^k)$.
- Compute $\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha^k \mathbf{g}^k$
- If stopping criterion met **STOP**, **Else** set $k = k + 1$ and go to (1).

6.5.2 Convergence Criteria

In practice the algorithm is terminated if some convergence criterion is satisfied. Usually termination is enforced at iteration k if one, or a combination of the following is met:

- $\|\mathbf{x}^k - \mathbf{x}^{k-1}\| < \epsilon_1$.

- $\|\nabla f(\mathbf{x}^k)\| < \epsilon_2$
- $|f(\mathbf{x}^k) - f(\mathbf{x}^{k-1})| < \epsilon_3$

Here ϵ_1, ϵ_2 and ϵ_3 are designated some small positive tolerances.

6.5.2.1 Example

Consider $f(\mathbf{x}) = 2x_1^2 + 3x_2^2$, where $\mathbf{x}_0 = (1,1)$. Use two iterations of Steepest Descent.

Solution:

Compute $\nabla f(\mathbf{x}) = \begin{bmatrix} 4x_1 \\ 6x_2 \end{bmatrix} = \mathbf{g}$.

First Iteration:

We know that:

$$\mathbf{x}_1 = \mathbf{x}_0 - \alpha_0 g(\mathbf{x}_0),$$

so:

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 4\alpha \\ 6\alpha \end{bmatrix} = \begin{bmatrix} 1-4\alpha \\ 1-6\alpha \end{bmatrix}.$$

Therefore:

$$\begin{aligned} f(\mathbf{x}_0 - \alpha_0 g(\mathbf{x}_0)) &= 2(1-4\alpha)^2 + 3(1-6\alpha)^2 \\ &= 2 - 16\alpha + 32\alpha^2 + 3 - 36\alpha + 108\alpha^2 \\ \Rightarrow \nabla f(\mathbf{x}_0 - \alpha_0 g(\mathbf{x}_0)) &= 280\alpha - 52 = 0 \\ \Rightarrow \alpha &= 52/280 = 13/70. \end{aligned}$$

Finally:

$$\mathbf{x}_1 = \begin{bmatrix} 1 - 4\frac{13}{70} \\ 1 - 6\frac{13}{70} \end{bmatrix} = \begin{bmatrix} \frac{9}{35} \\ \frac{-4}{35} \end{bmatrix}.$$

Second Iteration:

We have:

$$\mathbf{x}_2 = \mathbf{x}_1 - \alpha_1 g(\mathbf{x}_1)$$

Compute (Simplified here):

$$f(\mathbf{x}_1 - \alpha_1 g(\mathbf{x}_1)) = \frac{1}{35^2}\left(2(9-36\alpha)^2 + 3(-4+24\alpha)^2\right).$$

We get:

$$\begin{aligned} \nabla f(\mathbf{x}_1 - \alpha_1 g(\mathbf{x}_1)) &= 0 \\ \Rightarrow 60\alpha &= 13 \\ \alpha &= \frac{13}{60} \end{aligned}$$

Therefore:

$$\mathbf{x}_2 = \mathbf{x}_1 - \alpha_1 g(\mathbf{x}_1) = \begin{bmatrix} \frac{9}{35} \\ \frac{-4}{35} \end{bmatrix} - \frac{13}{60} \begin{bmatrix} \frac{9}{35} \\ \frac{-4}{35} \end{bmatrix} = \begin{bmatrix} \frac{6}{175} \\ \frac{6}{175} \end{bmatrix}.$$

The process continues in the same manner above. We can see from inspection that the function should achieve a minimum at (0, 0). We can see this as a sanity check in the Python code below.

It is also worth noting that since this is a quadratic function, we can actually use another technique. We will redo the first iteration as illustration. Specifically, the quadratic functions allow α to be solved using:

$$\alpha_k = \frac{-g^{k^T} d^k}{d^{k^T} Q d^k}.$$

Thus:

First Iteration:

Compute $f(\mathbf{x}^0) = 5$, $\mathbf{g}(\bar{x}^0)^T = (4,6)$ and $Q = \begin{bmatrix} 4 & 0 \\ 0 & 6 \end{bmatrix}$

Therefore:

$$\alpha_1 = -\frac{(g^k)^T d^k}{(d^k)^T Q d^k} = \frac{52}{(4,6)\begin{bmatrix} 4 & 0 \\ 0 & 6 \end{bmatrix}\begin{bmatrix} 4 \\ 6 \end{bmatrix}} = \frac{13}{70}$$

Thus:

$$\mathbf{x}^1 = (1,1) - \frac{13}{70}(4,6) = \left(-\frac{9}{35}, \frac{4}{35}\right)$$

Similarly, the process repeats.

6.5.3 Inexact Line Search

Although you will only cover inexact line search techniques in the third year syllabus, we will quickly introduce a very simply inexact technique to use for the purpose of your labs.

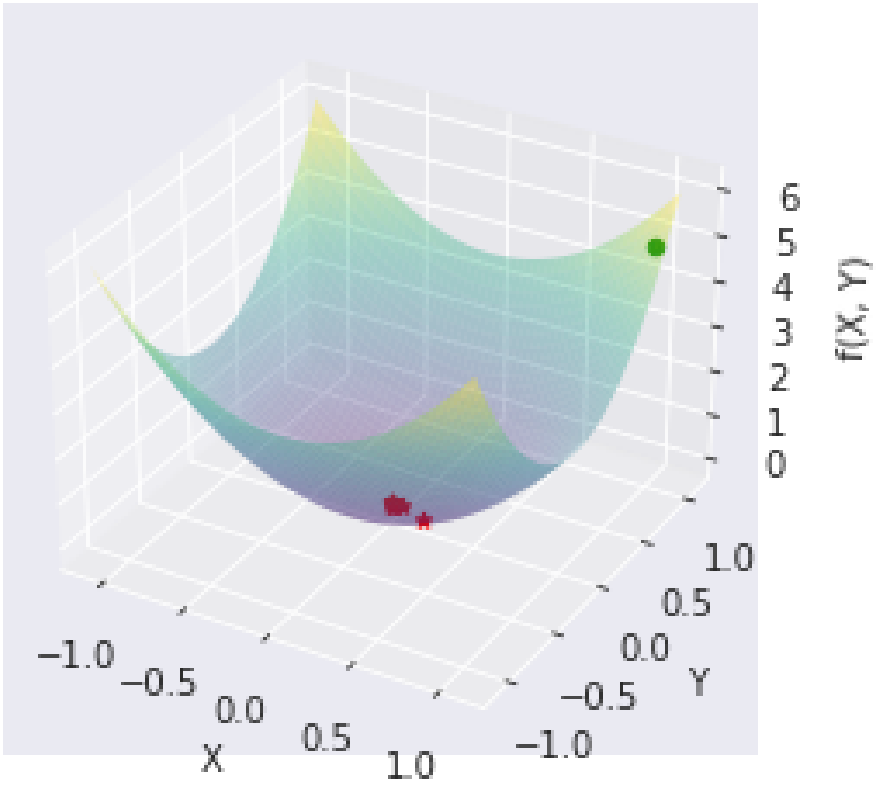


Figure 6.4: $2x_1^2 + 3x_2^2$

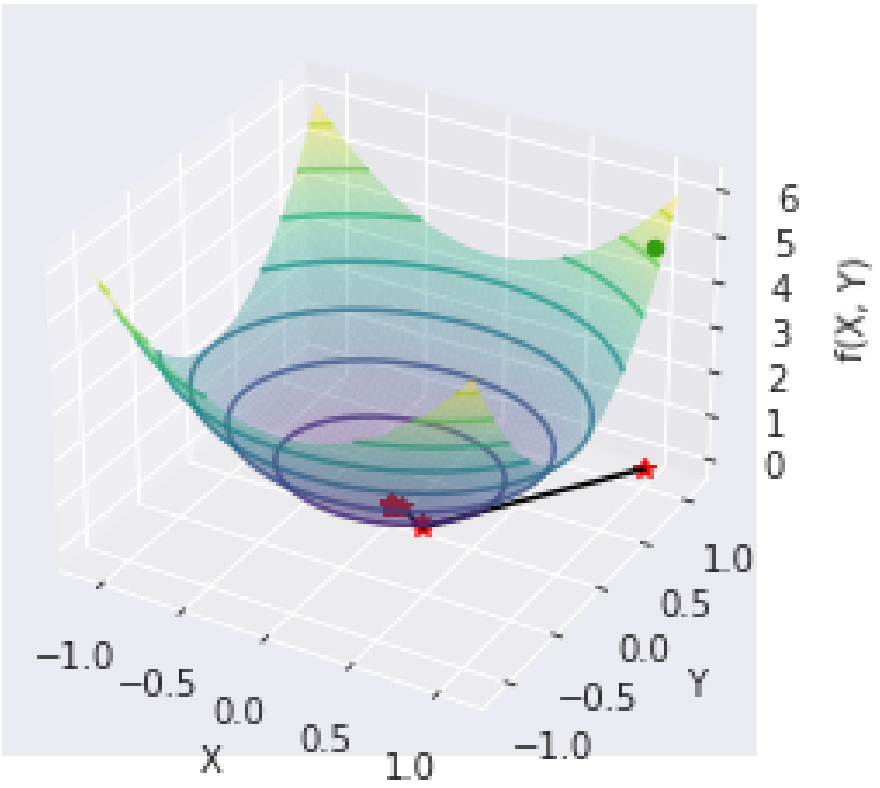


Figure 6.5: Corresponding contour plot

6.5.3.1 Backtracking Line Search

One way to adaptively choose the step size is to do the following:

- First fix a parameter $0 < \beta < 1$
- Then at each iteration, start with $t = 1$ and while

$$f(x - t\nabla f(x)) > f(x) - \frac{t}{2}\|\nabla f(x)\|^2,$$

  and update $t = \beta t$.

This is a simple technique and tends to work quite well in practice. For further reading you can consult *Convex Optimisation* by Boyd.

Steepest Descent Inexact Method

```
Inputs: f, g, x0, beta and tol
  from numpy import linalg as LA
  it    = 0
  xvals = np.array([x0])
  while LA.norm(g(x0)) > tol and it < 1000:
      alpha = 1
      while(f(x0 - alpha*g(x0))) > (f(x0) - (alpha/2)*LA.norm(g(x0))**2):
          alpha = beta*alpha

      x0 = x0 - alpha*g(x0)
      it += 1
      xvals = np.append(xvals, x0)
  Return x0, it, xvals
```

6.5.4 Exercises

1. Show that the value of the function

$$ax_1^2 + bx_2^2 + cx_3^2$$

   reached after taking a single of the steepest descent method from the point $(1,1,1)^T$ is:

$$\frac{ab(b-a)^2 + bc(c-b)^2 + ca(a-c)^2}{a^3 + b^3 + c^3}.$$

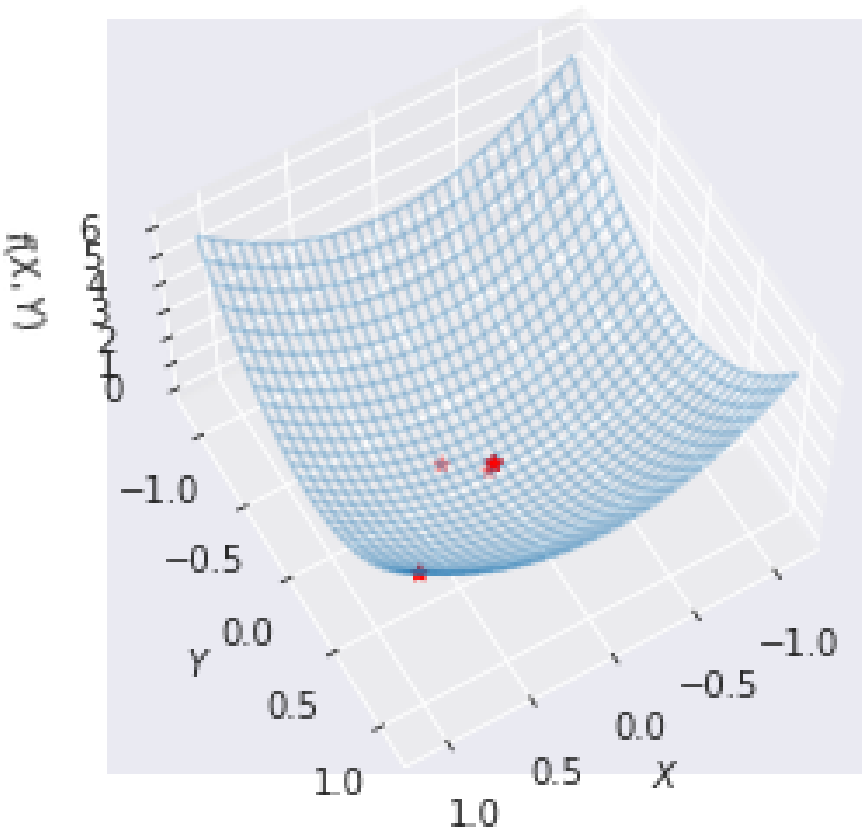


Figure 6.6: $2x_1^2 + 3x_2^2$

2. Show that if exact line search is carried out on the quadratic

$$\frac{1}{2}x^T Qx + b^T x + c$$

using the iteration

$$x^{k+1} = x^k + \alpha^k d^k,$$

then:

$$\alpha^k = -\frac{g^{(k)T} d^k}{d^{(k)T} Q d^k}.$$

3. Compute the first two iterations of the method of steepest descent applied to the objective function

$$f(\mathbf{x}) = 4{x_1}^2 + {x_2}^2 - {x_1}^2 x_2$$

with $x^0 = [1,1]^T$. Use exact line search.

4. Use three iterations of the steepest descent method on the function

$$f(\mathbf{x}) = 3{x_1}^2 + 2{x_2}^2$$

with initial point $(1,1)^T$.

6.6 The Gradient Descent Algorithm and Machine Learning

We will briefly look at the context of what we have learnt from the machine learning perspective. This is to emphasize the power of this chapter. In machine learning, you will find the gradient descent algorithm everywhere. While the literature may seem to allude to this method being new, powerful and cool, it is really nothing more than the method of steepest descent introduced above.

6.6.1 Basic Example

Let's try find a local minimum for the function $f(x) = x^3 - 2x^2 + 2$:

```
## (-1.0, 2.5)
```

```
## (0.0, 3.0)
```

So from the above plot we can see that there is a local minimum somewhere around 1.3 - 1.4 according to the x-axis. Of course, we normally won't be afforded the luxury of information such as this *a priori*, so let's just assume we arbitrarily set our starting point to be $x_0 = 2$. Implementing the gradient descent with a fixed stepsize, or learning rate (in the context of ML) we have:

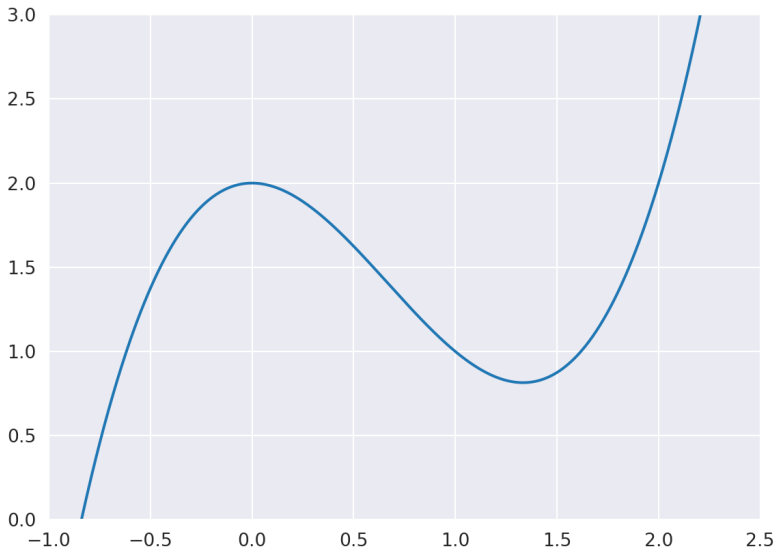


Figure 6.7: $f(x) = x^3 - 2x^2 + 2$

```
x_old = 0
x_new = 2 # The algorithm starts at x=2
n_k = 0.1 # step size fixed at 0.1
precision = 0.0001 # tolerance value

x_list, y_list = [x_new], [f(x_new)]

# returns the value of the derivative of our function
def f_prime(x):
    return 3*x**2-4*x

while abs(x_new - x_old) > precision:
    x_old = x_new
    s_k = -f_prime(x_old)
    x_new = x_old + n_k * s_k
    x_list.append(x_new)
    y_list.append(f(x_new))
print("Local minimum occurs at:", x_new)
```

```
## Local minimum occurs at: 1.3334253508453249
```

```
print("Number of steps:", len(x_list))
```

```
## Number of steps: 17
```

How did the algorithm look step by step?

```
## (-1.0, 2.5)
```

```
## (0.0, 3.0)
```

```
## (1.2, 2.1)
```

```
## (0.0, 3.0)
```

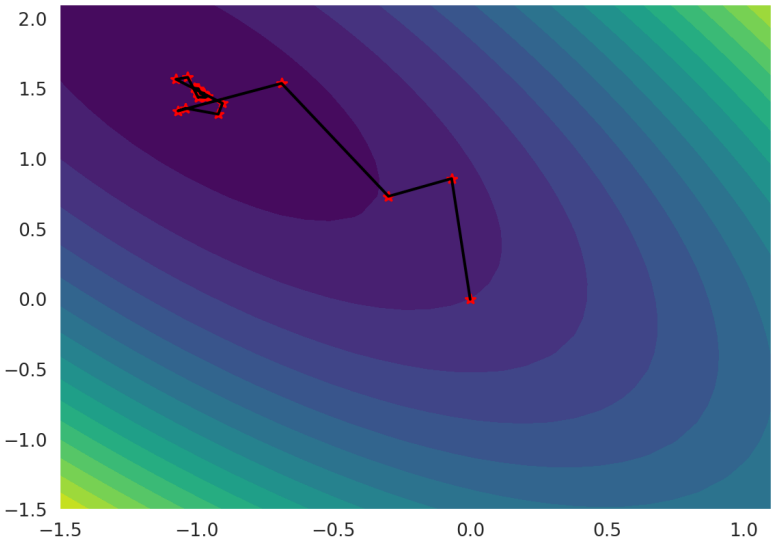


Figure 6.8: Iterative steps

In our above implementation we had a fixed step-size n_k. In machine learning, this is called the **learning rate**. You'll notice this is contrary to the algorithm in the aforementioned pseudocode. Making the assumption of the fixed learning rate made the implementation easier but could yield the issues mentioned in the beginning of the chapter.

6.6.2 Adaptive Step-Size

One means of overcoming this issue is to use adaptive step-sizes. This can be done using scipy's fmin function to find the optimal step-size at each iteration.

```
from scipy import stats
from scipy.optimize import fmin

# we setup this function to pass into the fmin algorithm
def f2(n,x,s):
    x = x + n*s
    return f(x)

x_old = 0
x_new = 2 # The algorithm starts at x=2
precision = 0.0001

x_list, y_list = [x_new], [f(x_new)]

# returns the value of the derivative of our function
def f_prime(x):
    return 3*x**2-4*x

while abs(x_new - x_old) > precision:
    x_old = x_new
    s_k = -f_prime(x_old)

    # use scipy fmin function to find ideal step size.
    # Uses the downhill simplex algorithm which is a zero-order method
    n_k = fmin(f2,0.1,(x_old,s_k), full_output = False, disp = False)

    x_new = x_old + n_k * s_k
    x_list.append(x_new)
    y_list.append(f(x_new))

print("Local minimum occurs at ", float(x_new))
```

```
## Local minimum occurs at  1.3333333284505209
```

```
print("Number of steps:", len(x_list))
```

```
## Number of steps: 4
```

So we can see that using the adaptive step-sizes, we've reduced the number of iterations to convergence from 17 to 4. This is a substantial reduction, however, it must

be noted that it takes time to compute the appropriate step-size at each iterations. This highlights a major issue in the decision making for optimisation: *trying to find the balance between speed and accuracy.*

How did the modified algorithm look step by step?

Well we can see that it converges rapidly and after the first two iterations, we need to zoom in to see further improvements.

```
## (-1.0, 2.5)

## (1.2, 2.1)

## (0.0, 3.0)

## (1.3333, 1.3335)

## (0.0, 3.0)
```

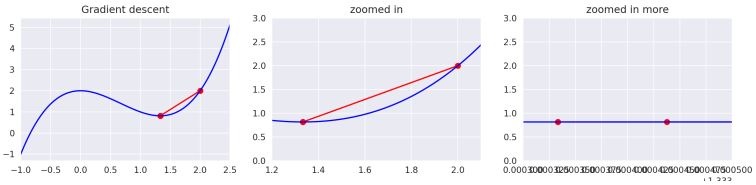


Figure 6.9: Iterative steps

6.6.3 Decreasing Step-Size

Instead of using computational resources having to find an optimal step-size at each iteration, we could apply an dampening factor at each step to reduce the step-size over time. For example:

$$\eta(t+1) = \frac{\eta(t)}{1 + t \times d}$$

```
x_old = 0
x_new = 2 # The algorithm starts at x=2
n_k = 0.17 # step size
precision = 0.0001
t, d = 0, 1

x_list, y_list = [x_new], [f(x_new)]
```

```
# returns the value of the derivative of our function
def f_prime(x):
    return 3*x**2-4*x

while abs(x_new - x_old) > precision:
    x_old = x_new
    s_k = -f_prime(x_old)
    x_new = x_old + n_k * s_k
    x_list.append(x_new)
    y_list.append(f(x_new))
    n_k = n_k / (1 + t * d)
    t += 1

print("Local minimum occurs at:", x_new)
```

```
## Local minimum occurs at: 1.3308506740900838
```

```
print("Number of steps:", len(x_list))
```

```
## Number of steps: 6
```

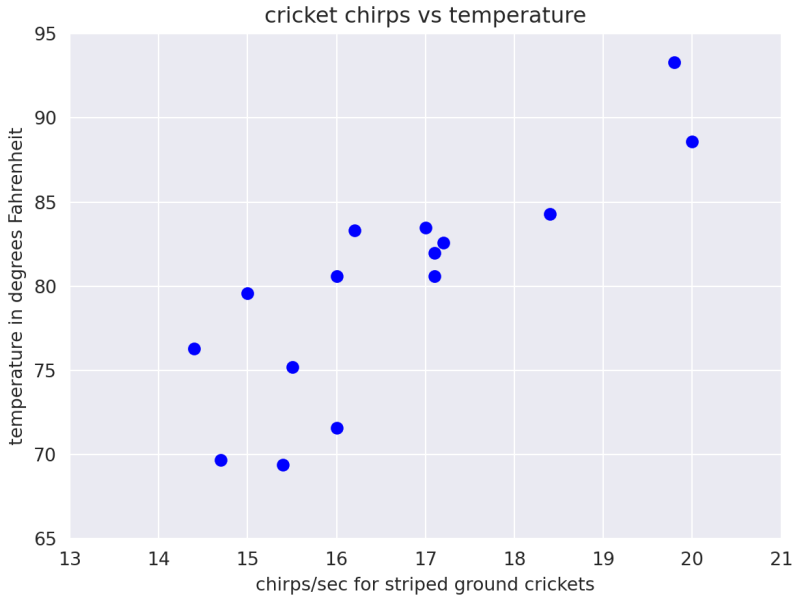
We can now see that we've still reduced the number of iterations required substantially but are not bounding to finding an optimal step-size at each iteration. This highlights that trade-off of finding cheap improvements that improve convergence at **minimal cost**.

How Do We Use the Gradient Descent in Linear Regression?

While using these line methods to find the minima of basic functions is interesting, one might wonder how this relates to some of the regressions we are interested in performing. Let us consider a slightly more complicated example. In this data set, we have data relating to how temperature affects the noise produced by crickets. Specifically, the data is a number of observations or samples of cricket chirp rates at various temperatures.

```
## (13.0, 21.0)
```

```
## (65.0, 95.0)
```



What can we deduce from the plotted data?

We can see that the data set is exhibiting a linear relationship. Therefore, our aim is to find the equation of the straight line given by:

$$h_\theta(x) = \theta_0 + \theta_1 x,$$

that best fits all of our data points, i.e. minimise the residual error.

The function that we are trying to minimize in this case is:

$$J(\theta_0,\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$$

In this case, our gradient will be defined in two dimensions:

$$\frac{\partial}{\partial\theta_0}J(\theta_0,\theta_1) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)$$

$$\frac{\partial}{\partial\theta_1}J(\theta_0,\theta_1) = \frac{1}{m}\sum_{i=1}^{m}((h_\theta(x_i) - y_i)\cdot x_i)$$

Below, we set up our function for h, J and the gradient:

```
h = lambda theta_0,theta_1,x: theta_0 + theta_1*x

def J(x,y,m,theta_0,theta_1):
    returnValue = 0
    for i in range(m):
```

```
        returnValue += (h(theta_0,theta_1,x[i])-y[i])**2
    returnValue = returnValue/(2*m)
    return returnValue

def grad_J(x,y,m,theta_0,theta_1):
    returnValue = np.array([0.,0.])
    for i in range(m):
        returnValue[0] += (h(theta_0,theta_1,x[i])-y[i])
        returnValue[1] += (h(theta_0,theta_1,x[i])-y[i])*x[i]
    returnValue = returnValue/(m)
    return returnValue
```

```
import time
start = time.time()
theta_old = np.array([0.,0.])
theta_new = np.array([1.,1.]) # The algorithm starts at [1,1]
n_k = 0.001 # step size
precision = 0.001
num_steps = 0
s_k = float("inf")

while np.linalg.norm(s_k) > precision:
    num_steps += 1
    theta_old = theta_new
    s_k = -grad_J(x,y,m,theta_old[0],theta_old[1])
    theta_new = theta_old + n_k * s_k

print("Local minimum occurs where:")
```

```
## Local minimum occurs where:
```

```
print("theta_0 =", theta_new[0])
```

```
## theta_0 = 25.128552558595363
```

```
print("theta_1 =", theta_new[1])
```

```
## theta_1 = 3.297264756251897
```

```
print("This took",num_steps,"steps to converge")
```

```
## This took 565859 steps to converge
```

```
end = time.time()
print(str(end - start) + 'seconds')
```

```
## 19.64289903640747seconds
```

It's clear that the algorithm seems to take quite a long time for such a trivial example. Let's check that the values we've obtained from the gradient descent are any good. We can get the true values for θ_0 and θ_1 with the following:

```
from scipy import stats as sp
start = time.time()
actualvalues = sp.stats.linregress(x,y)
print("Actual values for theta are:")
```

```
## Actual values for theta are:
```

```
print("theta_0 =", actualvalues.intercept)
```

```
## theta_0 = 25.232304983426026
```

```
print("theta_1 =", actualvalues.slope)
```

```
## theta_1 = 3.2910945679475647
```

```
end = time.time()
print(str(end - start) + 'seconds')
```

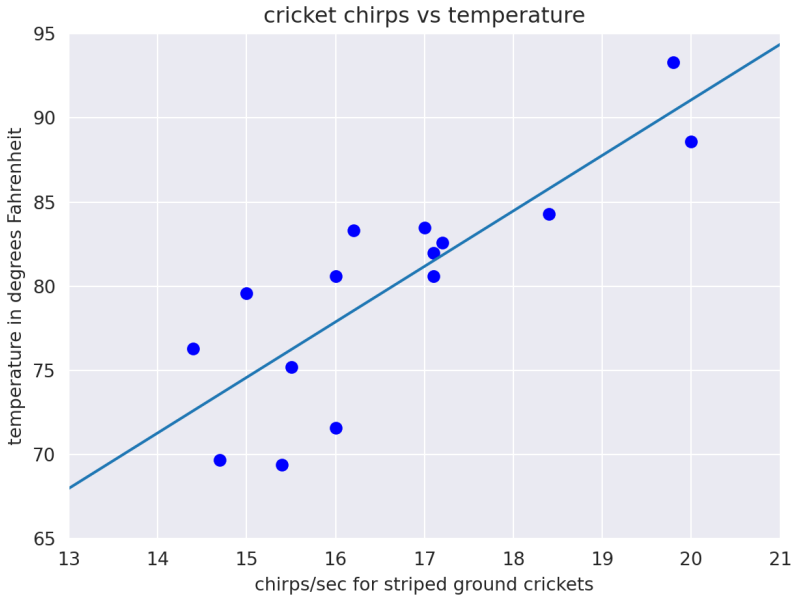
```
## 0.012906551361083984seconds
```

One thing this highlights is how much effort goes into optimising the functions found in these libraries. If one looks at the code inside linregress, clever exploitations to speed up the computation can be found.

Now, let's plot our obtained results on the original data set:

```
## (13.0, 21.0)
```

```
## (65.0, 95.0)
```



So in our implementation above, we needed to compute the gradient at each step. While this might not seem important, it is! In this toy example, we only have 15 data points, however, imagine the computational intractability when millions of data points are involved.

6.6.4 Stochastic Gradient Descent

What we implemented above is often called Vanilla/Batch gradient descent. As pointed out, this implementation means that we need to sum the cost of each sample in order to calculate the gradient of the cost function. This means given 3 million samples, we would have to loop through 3 million times!

> So to move a single step towards the minimum, one would need to calculate each cost 3 million times.

So what can we do to overcome this? Well, we can use the stochastic gradient descent. In this idea, we use the cost gradient of 1 sample at each iteration rather than the sum of the cost gradient of all samples. So recall our gradient equations from above:

$$\frac{\partial}{\partial\theta_0}J(\theta_0,\theta_1)=\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i)-y_i),$$

$$\frac{\partial}{\partial\theta_1}J(\theta_0,\theta_1)=\frac{1}{m}\sum_{i=1}^{m}((h_\theta(x_i)-y_i)\cdot x_i),$$

where:

$$h_\theta(x)=\theta_0+\theta_1 x.$$

We now want to update our values at each item in the training set instead of all so that we can begin improvement straight away.

We can redefine our algorithm into the stochastic gradient descent for the simple linear regression as follows:

```
Randomly shuffle the data set
 for k = 0, 1, 2, ... do
     for i = 1 to m do
```

$$\begin{bmatrix}\theta_0\\ \theta_1\end{bmatrix}=\begin{bmatrix}\theta_0\\ \theta_1\end{bmatrix}-\alpha\begin{bmatrix}2(h_\theta(x_i)-y_i)\\ 2x_i(h_\theta(x_i)-y_i)\end{bmatrix}$$

```
     end for
 end for
```

Depending on the size of the data set, we run the entire data set 1 to k times.

So the key advantage here is that unlike batch gradient descent where we have to go through the entire data set before initiating any progress, we can now make process straight away as we move through the data set. This is the primary reason why stochastic gradient descent is used when dealing with large data sets.

Chapter 7

Newton and Quasi-Newton Methods

The steepest descent method uses information based only on the first partial derivatives in selecting a suitable search direction. This strategy is not always the most effective. A faster method may be obtained by approximating the objective function $f(\mathbf{x})$ as a quadratic $q(\mathbf{x})$ and making use of a knowledge of the second partial derivatives. This is the basis of Newton's method. The idea behind this method is as follows. Given a starting point, we construct a quadratic approximation to the objective function that matches the first and the second derivative of the original objective function at that point. We then minimise the approximate (quadratic) function instead of the original objective function. We then use the minimiser of the quadratic function to obtain the next iterate and repeat the procedure iteratively. If the objective function is quadratic then the approximation is exact and and the method yields the true minimiser in one step. If, on the other hand, the objective function is not quadratic, then the approximation will provide only an estimate of the position of the true minimiser.

We can obtain a quadratic approximation to the given twice continuously differentiable objective function using the Taylor series expansion of f about the current x^k, neglecting terms of order three and the higher. Using the Taylor series expansion:

$$f(\mathbf{x}) \approx f(\mathbf{x}^{(k)}) + (\mathbf{x}-\mathbf{x}^{(k)})^T \mathbf{g}^{(k)} + (\mathbf{x}-\mathbf{x}^{(k)})^T H(\mathbf{x}^{(k)})(\mathbf{x}-\mathbf{x}^{(k)}) = q(\mathbf{x}),$$

where $\mathbf{g} = \nabla f$ and H is the Hessian matrix. The minimum of the quadratic $q(\mathbf{x})$ satisfies:

$$0 = \nabla q(\mathbf{x}) = \mathbf{g}^{(k)} + H(\mathbf{x}^{(k)})(\mathbf{x}-\mathbf{x}^{(k)}),$$

or inverting:

$$\mathbf{x} = \mathbf{x}^{(k)} - H^{-1}(\mathbf{x}^{(k)})\mathbf{g}^{(k)}.$$

Newton's formula is:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - H^{-1}(\mathbf{x}^{(k)})\mathbf{g}^{(k)}. \tag{7.1}$$

This can be rewritten as

$$H^{(k)}\mathbf{d}^{(k)} = -\mathbf{g}^{(k)} \tag{7.2}$$

where $\mathbf{d}^{(k)} = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}$

Note to solve in 1-dimension $g(x) = 0$, we iterate $x_{k+1} = x_k - \frac{g(x_k)}{g'(x_k)}$. The above formula is the multidimensional extension of Newton's method.

The Method requires that f^k, $\mathbf{g}^k$ and H^k i.e., the function value, the gradient and the Hessian to be made available at each iterate $\mathbf{x}^k$. Most importantly the Newton method is only well defined if the Hessian H^k is positive definite. This is because only then $q(\mathbf{x})$ will have a unique minimiser. The positive definiteness of the Hessian can only be guaranteed if the starting iterate $\mathbf{x}^0$ is very near to the minimizer $\mathbf{x}^*$ of $f(\mathbf{x})$

The Newton method is fast to converge when it is applied close to the minimiser. If the starting point (the initial point) is further from the minimiser then the Algorithm may not converge.

7.0.0.1 Example

For example let us take the following example

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2.$$

Let us take $\mathbf{x}^0 = \binom{0}{0}$. The gradient vector and the Hessian at $\mathbf{x}^0$ are respectively given by:

$$\nabla f(\mathbf{x}) = \begin{pmatrix} -400x_1\left(x_2 - x_1^2\right) - 2(1 - x_1) \\ 200\left(x_2 - x_1^2\right) \end{pmatrix} = \mathbf{g},$$

and:

$$H(\mathbf{x}) = \begin{pmatrix} 800x_1^2 - 400\left(x_2 - x_1^2\right) + 2 & -400x_1 \\ -400x_1 & 200 \end{pmatrix}.$$

So substituting x^0 gives:

$$\mathbf{g}^0 = (2,0)^T\,; \quad H^0 = \begin{pmatrix} 2 & 0 \\ 0 & 200 \end{pmatrix}.$$

Now using

$$H^0\mathbf{d}^0 = -\mathbf{g}^0,$$

recall that:

$$H^k d^k = -g^k,$$

so:

$$H^0\mathbf{d}^0 = -\mathbf{g}^0 \Rightarrow \mathbf{d}^0 = \mathbf{g}^0(H^0)^{-1} = \begin{pmatrix}2\\0\end{pmatrix}\begin{pmatrix}1/2 & 0\\0 & 1/200\end{pmatrix} = \begin{pmatrix}1\\0\end{pmatrix}.$$

Recall:

$$\mathbf{d}^k = \mathbf{x}^{k+1} - \mathbf{x}^k \Rightarrow \mathbf{d}^0 = \mathbf{x}^1 - \mathbf{x}^0 \Rightarrow \mathbf{x}^1 = \mathbf{d}^0 + \mathbf{x}^0$$

Thus:

$$\mathbf{x}^1 = \begin{pmatrix}1\\0\end{pmatrix} + \begin{pmatrix}0\\0\end{pmatrix} = \begin{pmatrix}1\\0\end{pmatrix}$$

Calculating the function value we have:

$$f(\mathbf{x}^1) = 100 > f(\mathbf{x}^0) = 1$$

which shows that the algorithm is diverging!

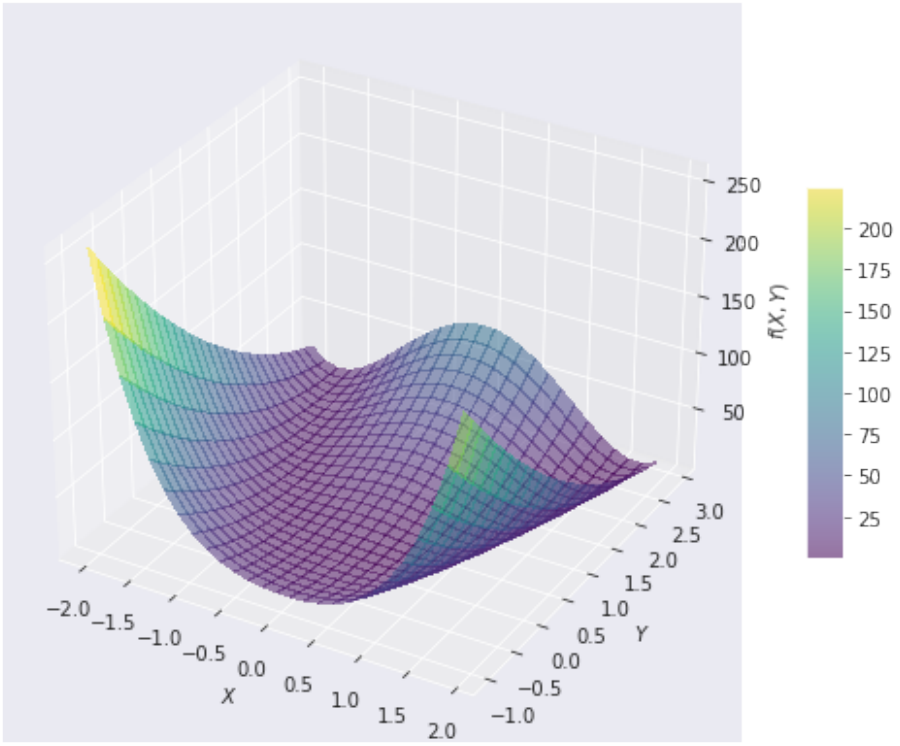


Figure 7.1: Rosenbrock Function

‘‘‘

7.1 The Modified Newton Method

The modified Newton method is

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha^k H^{-1}(\mathbf{x}^{(k)})\mathbf{g}^{(k)}.$$

The step length parameter α^k modifies the step taken in the search direction, usually to minimize $f(\mathbf{x}^{(k+1)})$. Newton's method applied without this modification does not necessarily produce a decrease in $f(\mathbf{x}^{(k+1)})$, as described by the above example.

To address the drawbacks of Newton method line search is introduced where $f^{k+1} < f^k$ is sought. As with the other gradient based methods the new iterate $\mathbf{x}^{k+1}$ is found by minimizing f along the search direction $\mathbf{d}^k$ such that:

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha_k \mathbf{d}^k$$

where α^k is the value of α which minimizes $f(\mathbf{x}^k + \alpha \mathbf{d}^k)$.

Although Newton method without this modification may generate points where the function may increase (see example above), the directions generated by Newton method are initially downhill if H^k is positive definite.

Remarks:

- Newton's method always goes in a descent direction provided we do not go too far but sometimes Newton over-steps the mark and does not work.
- The drawback to the method is that evaluating H^{-1} can be expensive in computational time.

7.2 Convergence of Newton's Method for Quadratic Functions

If $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T Q\mathbf{x} + \mathbf{x}^T\mathbf{b} + c$ is a quadratic function with positive definite symmetric Q, then Newton's method reaches the minimum in 1 step irrespective of the initial starting point.

Proof:

The gradient vector $\mathbf{g}(\mathbf{x}) = \nabla f(\mathbf{x}) = Q\mathbf{x} + \mathbf{b}$. The Hessian $H(\mathbf{x}) = Q$ and is a constant. Hence given $\mathbf{x}^{(0)}$,

$$\begin{aligned} \mathbf{x}^{(1)} &= \mathbf{x}^{(0)} - H^{-1}\mathbf{g}^{(0)} \\ \mathbf{x}^{(1)} &= \mathbf{x}^{(0)} - Q^{-1}\left(Q\mathbf{x}^{(0)} + \mathbf{b}\right) \\ \mathbf{x}^{(1)} &= -Q^{-1}\mathbf{b} = \mathbf{x}^*. \end{aligned}$$

The result also works if Q is negative definite resulting in a strong local maximum or Q is symmetric indefinite giving $\mathbf{x}^*$ as a saddle point.

7.3 Quasi-Newton Methods

The basic Newton method as it stands is not suitable for a general purpose algorithm since H^k may not be positive definite when $\mathbf{x}^k$ is remote from the solution. Furthermore, as we have shown in the previous example, even if H^k is positive definite the convergence may not occur. To address these issues Quasi-Newton algorithms were developed. We start by describing the drawbacks of the Newton method. At each iteration (say, at the k-th iteration) of the Newton's method a new matrix H^k has to be calculated (even if the method uses line search) and then either the inverse of this matrix has to found or a system of equation has to be solved before the new point $\mathbf{x}^{(k+1)}$ is found using $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{d}^{(k)}$. Quasi-Newton methods avoid the calculation of a new matrix at each iteration, rather they only update the matrix (positive definite) of the previous iteration. This matrix remains also positive definite. This method also does not need to solve a system of equation. First it finds its direction using the positive definite matrix and it finds the step length using line search.

Introduction of the quasi-Newton method largely increased the range of problems which could be solved. *This type of method is like Newton method with line search, except that $H^{k^{-1}}$ at each iteration is approximated by a symmetric positive definite matrix G^k, which is updated from iteration to iteration.* Thus the kth iteration has the basic structure.

1. Set $\mathbf{d}^k = -G^k\mathbf{g}^k$
2. Line search along $\mathbf{d}^k$ giving $\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha^k\mathbf{d}^k$
3. Update G^k giving G^{k+1}

The initial positive definite matrix is chosen as $G^0 = I$. Potential advantages of the method (as against Newton's method) are:

- Only first derivative required (Second derivative required in Newton method)
- G^k positive definite implies the descent property (H^k may be indefinite in Newton method)

Much of the interest lies in the updating formula which enables G^{k+1} to be calculated from G^k. We know that for any quadratic function:

$$q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T H\mathbf{x} + \mathbf{b}^T\mathbf{x} + c,$$

where H, $\mathbf{b}$ and c are constant and H is symmetric, the Hessian maps differences in position into differences in gradient,.i.e.,

$$\mathbf{g}^{k+1} - \mathbf{g}^k = H(\mathbf{x}^{k+1} - \mathbf{x}^k). \tag{7.3}$$

The above property says that changes in gradient $\mathbf{g}$ ($=\nabla f(x)$) provide information about the second derivative of $q(\mathbf{x})$ along $(\mathbf{x}^{k+1} - \mathbf{x}^k)$. In the quasi-Newton methods

at $\mathbf{x}^k$ we have the information about the direction $\mathbf{d}^k$, G^k and the gradient $\mathbf{g}^k$. We can use these information to perform line search to obtain $\mathbf{x}^{k+1}$ and $\mathbf{g}^{k+1}$. We now need to calculate G^{k+1} (the approximate inverse of H^{k+1}) using the above information. At this point we impose the condition given by Equation (7.3) for the non-quadratic function f. In other words, we impose that changes in the gradient provide information about the second derivative of f along the search direction $\mathbf{d}^k$. Hence, we have:

$$H^{(k+1)^{-1}}(\mathbf{g}^{k+1} - \mathbf{g}^k) = (\mathbf{x}^{k+1} - \mathbf{x}^k) \tag{7.4}$$

Therefore, we would like have $G^{k+1} = G^k + \Delta G^k$ such that:

$$G^{k+1}\gamma^k = \delta^k, \tag{7.5}$$

where $G^{k+1} = H^{k+1^{-1}}$, $\delta^k = (\mathbf{x}^{k+1} - \mathbf{x}^k)$ and $\gamma^k = (\mathbf{g}^{k+1} - \mathbf{g}^k)$. This is known as the *quasi-Newton condition* and for the quasi-Newton algorithm the update H^{k+1} from H^k must satisfy Equation (7.5).

Methods differ in the way they update the matrix G^k. Essentially they are classified according to a rank one and rank two updating formulae.

7.3.1 The DFP Quasi-Newton Method

Rank two updating formulae are given by:

$$G^{k+1} = G^k\gamma^k + auu^T + bvv^T\gamma^k. \tag{7.6}$$

One method is to choose $u = \delta^k$ and $v = G^k\gamma^k$. Then $au^T\gamma^k = 1$ and $bv^T\gamma^k = -1$ determine a and b. Thus:

$$G^{k+1} = G^k + \frac{\delta^k\delta^{k^T}}{\delta^{k^T}\gamma^k} - \frac{G^k\gamma^k\gamma^{k^T}G^k}{\gamma^{k^T}G^k\gamma^k} \tag{7.7}$$

This formula was first suggested as part of a method due to Davidon (1959), and later also presented by Fletcher and Powel (1963). The Quasi-Newton method which goes with this updating formula is known as DFP (Davidson, Fletcher and Powel) method. The DFP algorithm is also known as the *variable matrix* algorithm. The DFP algorithm preserves the positive definiteness of G^k but can sometimes gives trouble when G^k becomes nearly singular. A modification (known as BFGS) introduced in 1970 can cure this problem. The algorithm for DFP method is given below:

1. Set $k = 0, G^0 = I$ and compute $\mathbf{g}^k = g(\mathbf{x}^k)$.
2. Compute $\mathbf{d}^k$ from $\mathbf{d}^k = -G^k\mathbf{g}^k$.
3. Compute $\alpha = \alpha^k$ such that $f(\mathbf{x}^k + \alpha^k\mathbf{d}^k)$, set $\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha^k\mathbf{d}^k$.
4. Compute $\mathbf{g}^{k+1}$ such that $\mathbf{g}^{k+1} = g(\mathbf{x}^{k+1})$.
5. If $\|g^{k+1}\| \leq \epsilon$ (ϵ is a user supplied small number) then go to (9).
6. Compute δ^k and γ^k such that $\delta^k = \mathbf{x}^{k+1} - \mathbf{x}^k$ and $\gamma = \mathbf{g}^{k+1} - \mathbf{g}^k$.
7. Compute G^{k+1}.

8. Set $k = k+1$ and go to (2).
9. Set $\mathbf{x}^* = \mathbf{x^{k+1}}$, STOP.

7.3.2 Exercises

1. Use Newton method to minimise the function:

$$f(x) = {x_1}^4 - 3x_1x_2 + (x_2+2)^2,$$

starting at the point $x^0 = [0,0]^T$ and show that the function value at x^0 cannot be improved searching in Newton direction.

2. Find the stationary points of:

$$f(x) = x_1^2 + x_2^2 - x_1^2x_2$$

and determine their nature. Plot the contours of f. Find the value of f after taking a basic Newton optimisation method from $x^0 = (1,1)^T$.

3. Using Newton method, find the minimiser of:

$$f(x) = \frac{1}{2}x^2 - \sin(x).$$

The initial value is $x^0 = 0.5$. The required accuracy is $\epsilon = 10^{-5}$ in the sense that you stop when $\left|x^{k+1} - x^k\right| < \epsilon$.

4. Using the DFP method, find the minimum of the following function:

$$f(\mathbf{x}) = 4x_1^2 - 4x_1x_2 + 3x_2^2 + x_1,$$

using the starting point (4, 3).

5. Find the minimum of the function given in question (2) utilising the DFP method. Use the same starting point.

Chapter 8

Direct Search Methods for Unconstrained Optimisation

Direct search methods, unlike the Descent methods discussed in earlier Chapters do not require the derivatives of the function. The Direct search methods require only the objective function values when finding minima and are often known as *zeroth-order methods* since they use the zeroth-order derivatives of the function. We will consider two **Direct Methods** in this course. Namely, the **Random Walk Method** and the **Downhill Simplex Method**.

8.1 Random Walk Method

The *random walk method* is based on generating a sequence of improved approximations to a minimum, where each approximation is derived from the previous approximation. Therefore, $\mathbf{x}_i$ is the approximation to the minimum obtained in the $(i-1)$th iteration, yielding the relation:

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \lambda \mathbf{u}_i,$$

where λ is some scalar step length and $\mathbf{u}_i$ some random unit vector generated at the ith stage.

We can describe the algorithm as follows:

1. Start with an initial point $\mathbf{x}_1$, a sufficiently large initial step length λ, a minimum allowable step length ϵ, and a maximum permissible number of iterations N.
2. Find the function value $f_1 = f(\mathbf{x}_1)$.
3. Set the iteration number, i, to 1

4. Generate a set of n random numbers, $r_1, \ldots, r_n$, each lying in the interval $[-1, 1]$ and formulate the unit vector $\mathbf{u}$ as:

$$\mathbf{u} = \frac{1}{(r_1^2 + r_2^2 + \ldots + r_n^2)^{1/2}} \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_n \end{bmatrix}.$$

   To avoid bias in the calculation, we only accept the vector if the length of:

$$\frac{1}{(r_1^2 + r_2^2 + \ldots + r_n^2)^{1/2}} \text{ is } \leq 1.$$

5. Compute the new vector and the corresponding function value $\mathbf{x} = \mathbf{x}_1 + \lambda \mathbf{u}$ and $f = f(\mathbf{x})$.
6. If $f < f_1$, then set the new values of $\mathbf{x}_1 = \mathbf{x}$ and $f_1 = f$ and go to step 3, else continue to 7.
7. If $i \leq N$, set the new iteration to $i + 1$ and go to step 4. Otherwise, if $i > N$, go to step 8.
8. Compute new, reduced, step length as $\lambda = \lambda/2$. If new step length is smaller than or equal to ϵ, then go to step 9, else go to step 4.
9. Stop the procedure by taking $\mathbf{x}_{\text{opt}} = \mathbf{x}_1$ and $f_{\text{opt}} = f_1$.

8.1.0.1 Example

Minimise $f(x_1, x_2) = x_1 - x_2 + 2x_1^2 + 2x_1x_2 + x_2^2$ using the random walk method. Begin with the initial point $x_0 = [0, 0]$ and a starting step length of $\lambda = 1$. Use $\epsilon = 0.05$ and iteration limit $N = 100$

If you code this method for the stated parameters, your output once the terminating criterion is met should approximately be: $x = [-0.99768499, 1.49885167]$ with the corresponding function value of $f(x) = -1.249993279604305$.

Let us plot the function to see if our answer makes sense:

8.2 Downhill Simplex Method of Nelder and Mead

A direct search method for the unconstrained optimisation problem is the Downhill simplex method developed by Nelder and Mead (1965). It does not make an assumption on the cost function to minimise. Importantly, the function in question does not need to satisfy any condition of differentiability unlike other methods, i.e. it is a zero order method. It makes use of simplices, or polytopes in given dimension $n+1$. For example, in 2 dimensions, the simplex is a polytope of 3 vertices (triangle). In 3 dimensional space it forms a tetrahedron.

The method starts from an initial simplex. Subsequent steps of the method consist of updating the simplex where it defines:

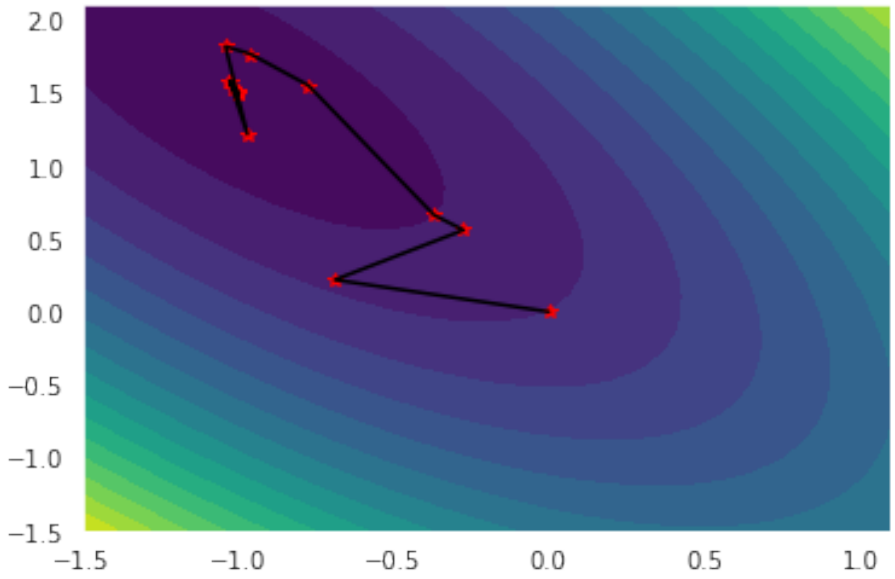


Figure 8.1: Random Walk

- $\mathbf{x}^h$ is the vertex with highest function value,
- $\mathbf{x}^s$ is the vertex with second highest function value,
- $\mathbf{x}^l$ is the vertex with lowest function value,
- $\mathbf{G}$ is the centroid of all the vertices except $\mathbf{x}^h$, ie. the centroid of n points out of $n+1$:

$$\mathbf{G} = \frac{1}{n} \sum_{j=1, j\neq h}^{n+1} \mathbf{x}^j \tag{8.1}$$

The movement of the simplex is achieved by using three operations, known as reflection, contraction and expansion.

These can be seen in the Figures below:

A common practice to generate the initial remaining simplex vertices is to make use of $\mathbf{x}_0 + \mathbf{e}_i b$, where $\mathbf{e}_i$ is the unit vector in the direction of the x_i coordinate and b an edge length. Assume a value of 0.1 for b.

Let $y = f(\mathbf{x})$ and $y^h = f(\mathbf{x}^h)$ then the algorithm suggested by Nelder and Mead is as follows:

The typical values for the above factors are $\alpha = 1$, $\gamma = 2$ and $\beta = 0.5$. The stopping

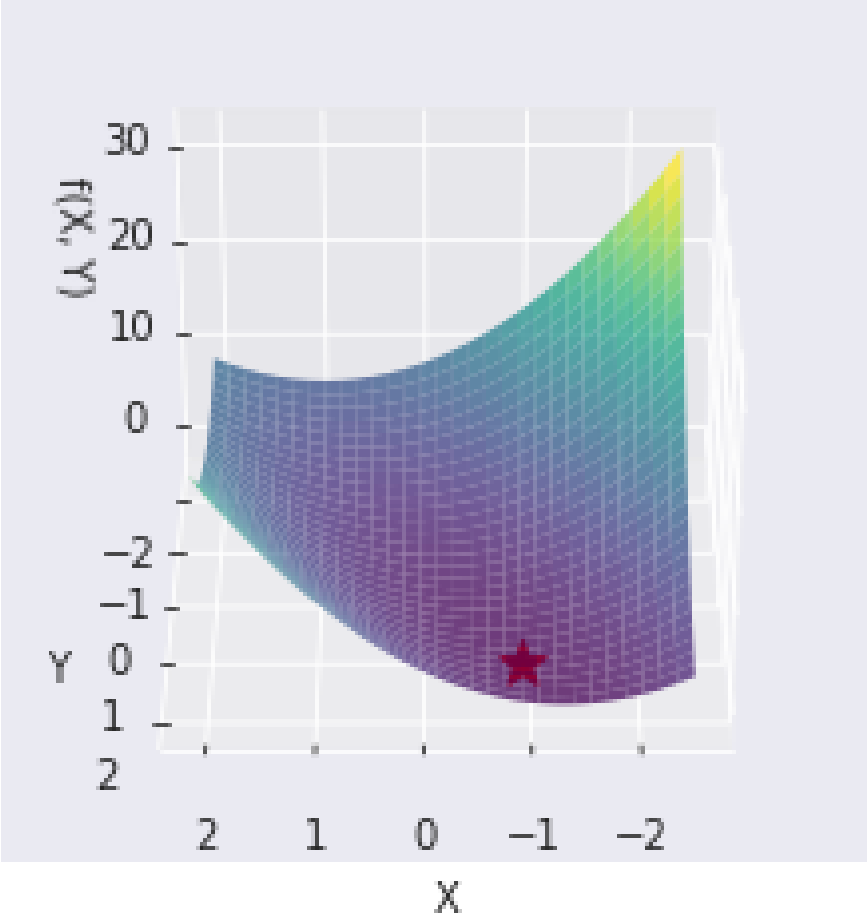


Figure 8.2: Random Walk

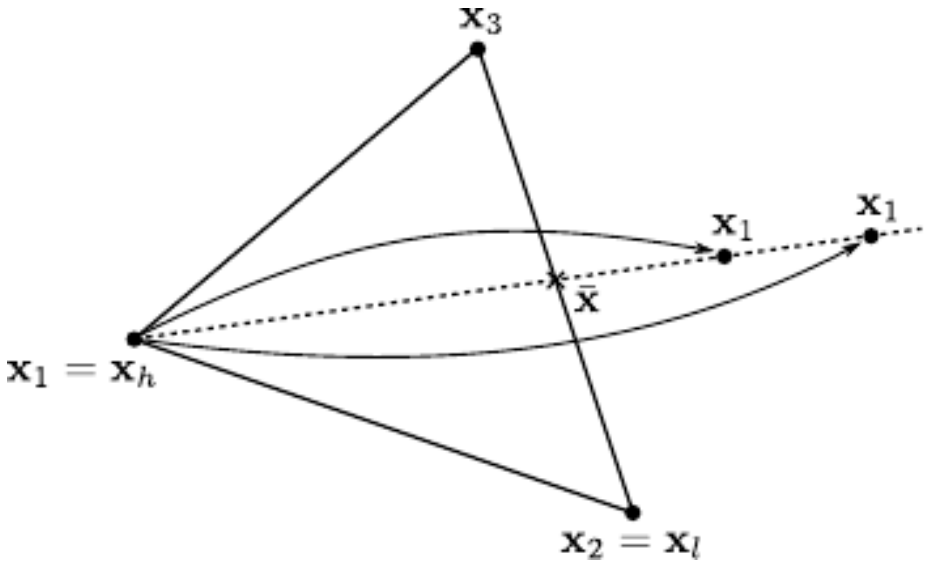


Figure 8.3: Here we have reflection and expansion

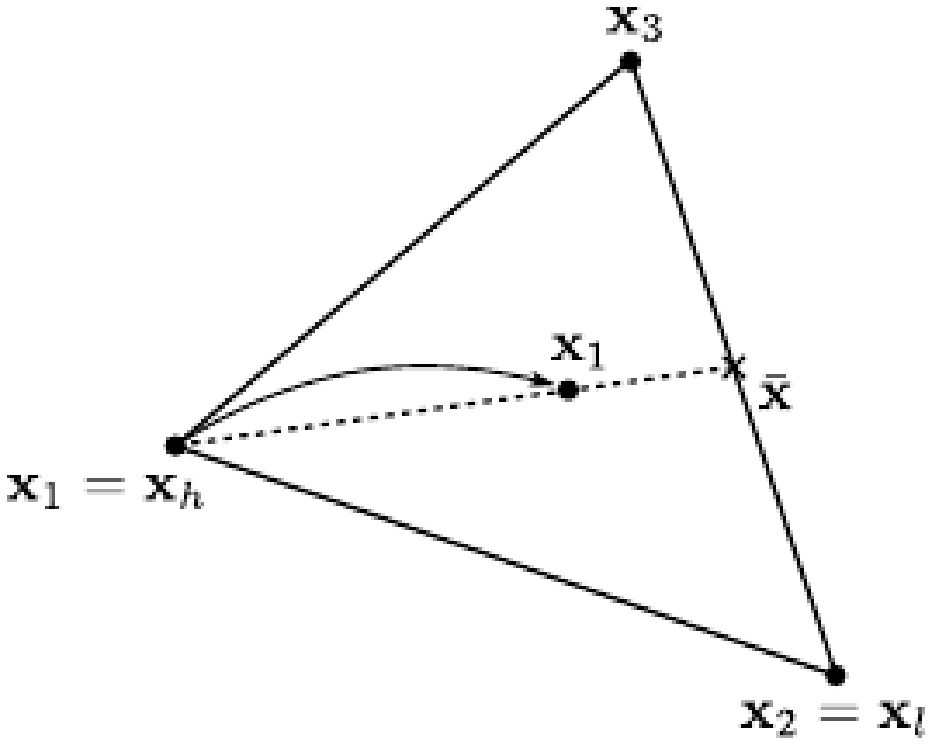


Figure 8.4: Here we have contraction

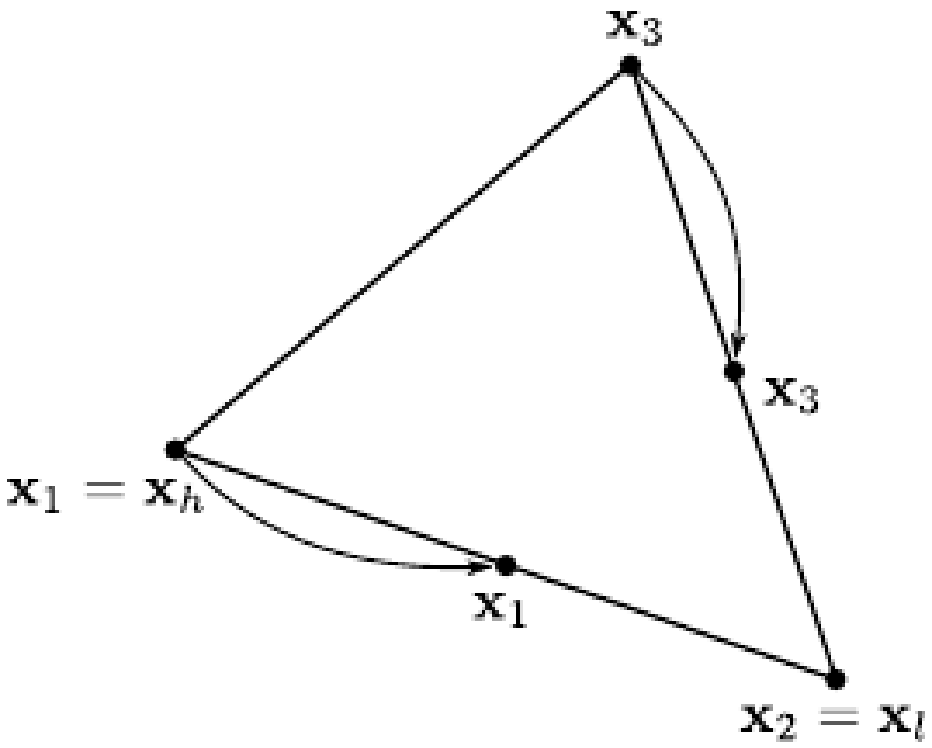


Figure 8.5: Here we have multiple contractions

Data: Input vertices of initial simplex and evaluated $f(\vec{x})$
Result: The $\vec{x}^*$, i.e. the local minimum of function f
while STOP-CRIT == FALSE **and** $k < k_{max}$ **do**
  Reflection: Reflect $\vec{x}^h$ using a reflection factor of $\alpha > 0$, i.e. find $\vec{x}'$;
  $\vec{x}' = (1+\alpha)\bar{G} - \alpha\vec{x}^h$;
  if $y^l \leq y' \leq y^s$ **then**
    Replace $\vec{x}^h$ by $\vec{x}'$;
    $\vec{x}' = (1+\alpha)\bar{G} - \alpha\vec{x}^h$;
  end
  if $y' < y^l$ **then**
    Expand the simplex, i.e. find x'';
    $\vec{x}'' = \gamma\vec{x}' + (1-\gamma)\bar{G}$;
    if $y'' < y^l$ **then**
      Replace $\vec{x}^h$ with $\vec{x}''$;
    else if $y'' \geq y^l$ **then**
      Replace $\vec{x}^h$ by $\vec{x}'$;
  else if $y' > y^s$ **then**
    Contract the simplex by contraction factor;
    if $y' < y^h$ **then**
      Find $\vec{x}''$ such that;
      $\vec{x}'' = \beta\vec{x}' + (1-\beta)\bar{G}$;
    else if $y' \geq y^h$ **then**
      Find $\vec{x}''$ such that;
      $\vec{x}'' = \beta\vec{x}^h + (1-\beta)\bar{G}$;
    if $y'' < y^h$ **and** $y'' < y'$ **then**
      Replace $\vec{x}^h$ by $\vec{x}''$;
    end
    if $y'' \geq y^h$ **and** $y'' > y'$ **then**
      Reduce the size of the simplex by halving the distances of $\vec{x}^l$;
    end
  $k = k+1$;
end
Return $\vec{x}^* = \vec{x}^l$

Algorithm 1: Downhill simplex of Nelder and Mead (1965)

Figure 8.6: Nelder Mead Algorithm

criteria to use is defined by:

$$\sqrt{\frac{1}{n+1}\sum_{i=0}^{n}\left(f(x_i)-\overline{f(x_i)}\right)^2} \leq \epsilon \tag{8.2}$$

8.2.1 Exercises

1. Apply the above two strategies to the all the multivariate function introduced in earlier chapters and achieve their respective minima.

Chapter 9

Lagrangian Multipliers for Constraint Optimisation

In this Chapter we will briefly consider the optimisation of continuous functions subjected to equality constraints (this will be covered extensively in the 3rd year course), that is the problem:

$$\text{minimize } \; z = f(\mathbf{x}) \tag{9.1}$$

subject to:

$$g_i(\mathbf{x}) = b_i$$

where f and g_i are differentiable. The Lagrange function, L, is defined by introducing one Lagrange multiplier λ_i for each constraint $g_i(\mathbf{x})$ as:

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda \sum_{i=1} \left[b_i - g_i(\mathbf{x}) \right] \tag{9.2}$$

The necessary condition of optimality are given by:

$$\frac{\partial L}{\partial x_j} = \frac{\partial f}{\partial x_j} + \sum_{i=1}^{m} \lambda_i \frac{\partial g}{\partial x_i} = 0, \;\; \frac{\partial L}{\partial \lambda_i} = g_i(\mathbf{x}) = 0 \tag{9.3}$$

9.0.1 Example

Use Lagrangian multipliers to minimise:

$$f(x_1, x_2) = x_1^2 + 4x_2^2$$

subject to:

$$x_1 + 2x_2 = 1$$

Solution:

$$\frac{\partial f}{\partial x_1} - \lambda \frac{\partial g}{\partial x_1} = 0,$$
$$\frac{\partial f}{\partial x_2} - \lambda \frac{\partial g}{\partial x_2} = 0,$$

and

$$g(x_1, x_2) = b.$$

Therefore, we solve:

$$2x_1 - \lambda = 0,$$
$$8x_2 - 2\lambda = 0,$$

and

$$x_1 + 2x_2 = 1.$$

Solving these three equations we obtain $x_1 = \frac{1}{2}$, $x_2 = \frac{1}{4}$ and $\lambda = 1$. Therefore, the optimum is:

$$f(x_1, x_2) = \frac{1}{2}.$$

9.0.2 Exercises

1. Minimise
$$f(\mathbf{x}) = {x_1}^2 + {x_2}^2$$
subject to
$$x_1 + 2x_2 + 1 = 0$$
2. Find the dimensions of a cylindrical tin of sheet metal to maximise its volume such that the total surface area is equal to $A_0 = 24\pi$.
