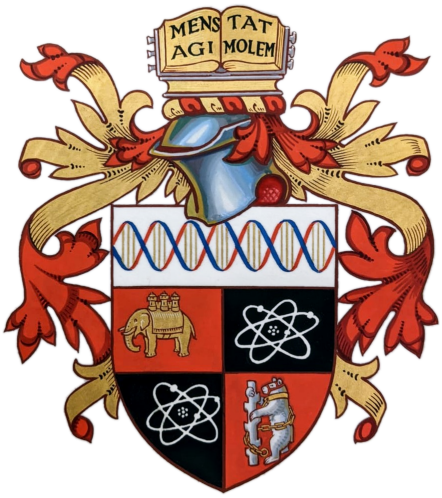
WARWICK
THE UNIVERSITY OF WARWICK



An Introduction to Convex Optimisation

by

Samuel Halford-Maw

A second year research paper
for the requirements of the module

MA262 Scientific Communication

University of Warwick

Department of Mathematics

October 2024

Contents

1 Mathematical Optimisation and Duality

"...in fact, the great watershed in optimization isn't between linearity and nonlinearity, but convexity and nonconvexity."

R. Tyrrell Rockafellar, in SIAM Review, 1993

1.1 A Brief History Of Convex Optimisation

Fundamentally, mathematical optimisation - or mathematical programming - refers to the problem of picking an optimal choice, relative your alternatives. Convex optimisation is a subfield. In the subfield, we require out problems to be of a certain form, and we leverage the regularity conditions of the problem to find a solution.

Mathematical optimisation goes all the way back to the founders of calculus, Newton and Leibniz, where using the tools of a derivative, we could find local and potentially global minimum and maximums. A bit later, came Lagrange who developed the method of Lagrange multipliers to solve constrained minimisation problems, those where the domain is restricted by 'constraints'.

Soon after came Minowski [1], Carathedory, Steiniz, who began the work on convex sets, and convex functions. While they were not interested so much in its application to optimisation, their work would prove fundamental for what was to come.

In the late 1940's, interest to convex optimisation exploded. George Dantzig, a mathematician working for RAND, published the simplex method in 1947. This method allowed the solving of a subset of convex problems at the time, those known as linear problems. Interestingly, his work was motivated by support from the US military during and after World War Two[2]. It allowed logisticians and planners to effectively plan tasks, procure weapons, fuel and food. With the advent of computers, even large linear problems became solveable in a feasible amount of time. (See the Stigler Diet below[3])

Around the same time, Von Neumann's published his work on duality, solving an optimisation problem by considering a dual related problem. Nash began working on max-min theory. Fenchel published a seminal paper [4] connecting convex functions and sets with convex problems. Kuhn and Tucker published the much celebrated KKT conditions in 1951 [5], establishing necessary and sufficient conditions relating the solution of the dual and primal problem. Interestingly, it was later found the necessary conditions had been stated by William Karush, then a graduate student, in 1939.

While the simplex method allowed solving of linear problems, for convex non-linear problems that could not be solved analytically methods were fragmented and lacked generalisability. Even the simplex method has

its problems, in its worse case, the time to solve grows exponentially with respect to the number of constraints and inputs to the problem. The ellipsoid method for rational data was developed by Soviet mathematicians, with a version attributed to Shor. This gave a polynomial time algorithm to linear program proving they could be solved in polynomial time, however this method was often slower in practice due to its high overhead cost.

The next leap forward was in the 1980's, through a series of inner point methods. Given the name by virtue of the fact they iterate through the interior of the feasible region to approach the optimal solution. The first inner point methods were again developed by Soviet Mathematicians throughout the 1970's and were in some sense reinvented by American mathematician in the mid 1980's. Narendra Karmarkar developed an inner point polynomial time algorithm for linear program that was also efficient in practice [6]. At the announcement of Karmarkar's work, a headline on the New York Times front sheet read, 'Breakthrough in Problem Solving'[7]. These methods for linear problems were generalised to convex problems by Nesterov and Nemirovski [8].

Since the 1980's, inner point methods have become more and more sophisticated. Optimisation, both convex and non-convex, has enjoyed renewed interest due to the AI boom in recent years. Specifically, due to the training of neural networks. Unfortunately, by design, the training of a neural network is a non-convex problem. Yet this has led to a birth of a subfield, applying convex methods to non-convex problems, with some paticularily interesting mathematics. (See the appendix for implementation of some of these gradient descent methods)

1.2 Examples of Convex Problems

To illustrate how useful the study of convex problems is, we give some historical and practical examples.

1.2.1 Stilger's Diet Problem

Before the Dantzig published his paper on the simplex method, Nobel Laureate Stilger posed the following question.

"For a moderately active man weighing 154 pounds, how much of each of 77 foods should be eaten on a daily basis so that the man's intake of nine nutrients will be at least equal to the recommended dietary allowances (RDAs) suggested by the National Research Council in 1943, with the cost of the diet being minimal?" [3]

This can be formulated as follows: If we require k different minerals and nutrients, in quantities $b_1, b_2, \dots, b_k$. Suppose we have n different types of food, and we consume in quantities $x_1, \dots, x_n$. A unit of food j, has $a_{i,j}$ content of nutrient i, with cost c_j associated with said unit. We can write the problem as below. [9]

$$\begin{aligned} \text{minimise} \quad & c^T x \\ \text{subject.} \quad & Ax \succeq b \\ & x \succeq 0 \end{aligned}$$

1.2.2 Markowitz Portfolio Theory

Suppose I am able to buy n financial assets. Let x_i denote the proportion of my portfolio in asset i. Suppose that I have a price model for the assets, a random vector $p \in \mathbb{R}^n$ with known mean $\overline{p}$, and covariance matrix Σ.

Then the return, r, is a random variable with mean $\bar{p}^T x$ and variance $x^T \Sigma x$. We would like to minimise the variance of our return, the risk, while maintaining a minimum average return. We formulate the problem as follows.

$$\begin{aligned} \text{minimise} \quad & x^T \Sigma x \\ \text{subject.} \quad & \bar{p}^T x \geq r_{\min} \\ & 1^T x = 1 \end{aligned}$$

Hopefully with a bot of motivation for why this topic is so useful, we can begin with the theory. We will take a tourist's tour through convex sets, convex functions and convex optimisation. If these topics interest you I would encourage you to see the bibliography for further reading. The natural next steps would be understanding the KKT conditions and implementations and numerical analysis of current inner point methods.

Throughout the text I have taken and modified definitions, theorems and proofs from [9], [4] and [8]. Due to the relative maturity of the field, many of these definitions or theorems would now be considered "universally known". Regardless, I do not claim any novelty in the ideas other than the presentation.

2 Convex Sets

Convex sets are the building blocks for convex optimisation, and while the definition is seemingly innocuous, in time we will see how we can leverage this "regularity property."

2.1 Introducing Convexity

Definition 2.1.1. *A set* $C \subseteq \mathbb{R}^n$ *is* convex *if for all* $x_1, x_2 \in C$*, any point on the line segment between these two points is contained inside the set.*

Formally, for all $x_1, x_2 \in C$ *and* $\theta \in [0,1]$ *we have:*

$$\theta x_1 + (1-\theta)x_2 \in C.$$

Please note for this essay we are restricting ourselves to the notion of convexity in Euclidean spaces. [1] We give the following definition for future clarity.

Definition 2.1.2. *Let* $C \subseteq \mathbb{R}^n$*. A point* $c \in \mathbb{R}^n$ *is said to be a* convex combination *of points in* C *if there exist points*

$$x_1, x_2, \dots, x_k \in C,$$

and scalars

$$\theta_1, \theta_2, \dots, \theta_k \quad \theta_i \in [0,1],$$

such that

$$\theta_1 + \theta_2 + \cdots + \theta_k = 1$$

and

$$c = \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_k x_k.$$

This leads to the intuitive result.

Lemma 2.1.1. *A set* $C \subseteq \mathbb{R}^n$ *is* convex *if and only if any convex combinations of points in* C *is contained in* C*.*

Proof.
Suppose any convex combination of points in C is contained in C. Then specifically, for any $x_1, x_2 \in C$ and $\theta \in \mathbb{R}$, we have that $p = \theta x_1 + (1-\theta)x_2$. So p is a convex combination of points in C and hence $p \in C$. So we have that C is convex.

[1] In principle, we could extend the theory to non-euclidean spaces that are geodiscally convex. This is an exciting area of active study due to the applications in machine learning, where data may be distributed on some 'nice' manifold.

Suppose we have that C is convex. We outline the proof for the induction on the length of k. Take $k = 1$. Then trivially any convex combination of points of length 1 is just a single point, and trivially $x_1 \in C$. Suppose the statement is true for length k, then for $k + 1$. If $\theta_{k+1} = 1$ then $\theta_i = 0, i \in [1, k]$, so the convex combination is the point itself which is in the set. If $\theta_{k+1} \neq 1$,

$$\begin{aligned} p &= \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_k x_k + \theta_{k+1} x_{k+1} \\ &= (1 - \theta_{k+1}) \left(\underbrace{\frac{\theta_1 x_1}{1 - \theta_{k+1}} + \frac{\theta_2 x_2}{1 - \theta_{k+1}} + \ldots + \frac{\theta_k x_k}{1 - \theta_{k+1}}}_{\text{A convex combination of length k}} \right) + \theta_{k+1} x_{k+1} \end{aligned}$$

Then as convex, and as $\theta_{k+1} + (1 - \theta_{k+1}) = 1$, $p \in C$ □

In fact this can be extended to define countable and uncountable convex combinations of points on C. This becomes more technical and uses the technology we are now going to introduce.

2.2 Hyperplanes and halfspaces

Definition 2.2.1. *A hyperplane* $\Pi_{(a,b)} \subseteq \mathbb{R}^n$ *is defined,*

$$\Pi_{(a,b)} = \{x \in \mathbb{R}^n \mid \langle a, x \rangle = b\}$$

where $a \in \mathbb{R}^n \backslash \{\boldsymbol{0}\}$ *and* $b \in \mathbb{R}$
Note: here and for the remainder of this text $\langle a, x \rangle := a^T x$.

This is a natural extension of the notion of a line or plane in n dimensional space.[2] And is easy to show that one can find some $x_0 \in \mathbb{R}^n$ such that $a^T x_0 = b$ and therefore write $\Pi_{(a,b)} = x_0 + a^\perp$, where $a^\perp$ denotes the orthogonal complement of a. From this reformulation we see that $\Pi_{(a,b)}$ is a "shifted" $n - 1$ dimensional vector space, what is known as an $n - 1$ dimensional *affine* space.

Like in $\mathbb{R}^2$ and $\mathbb{R}^3$ the hyperplane defines a partition on $\mathbb{R}^n$. The points that are on or "above" the hyperplane and those "below". We give the natural formalisation,

Definition 2.2.2. *The (closed) halfspace* $\mathscr{H}_{(a,b)} \subseteq \mathbb{R}^n$ *is defined,*

$$\mathscr{H}_{(a,b)} = \{x \in \mathbb{R}^n \mid \langle a, x \rangle \leq b\}$$

where $a \in \mathbb{R}^n \backslash \{\boldsymbol{0}\}$ *and* $b \in \mathbb{R}$

See that the boundary of $\mathscr{H}_{(a,b)}$, denoted $\partial \mathscr{H}_{(a,b)} = \Pi_{(a,b)}$. We say that $\mathscr{H}_{(a,b)}$ is the halfspace below $\Pi_{(a,b)}$. Similarly, we say $\mathscr{H}_{(-a,-b)}$ is the halfspace above $\Pi_{(a,b)}$ We see that both a hyperplane and halfspace are convex spaces. Simply, show that the convex combination of two points, fits the specification of the sets by linearity.

It is through this language we can begin to view the "dual" notion of a convex set, and while this may seem philosophical initially it will pay dividends. Instead of considering a convex set as a collection of points, we can see it as an intersection of halfspaces.

We quickly note the following lemma,

[2] Note any line or plane is a hyperplane in $\mathbb{R}^2$ and $\mathbb{R}^3$ respectively.

Lemma 2.2.1. *Let $\mathscr{C}$ be an arbitrary collection of convex sets, then the set defined by taking the intersection of every element in $\mathscr{C}$ is convex.*

Proof.
Suppose $\mathscr{C}$ is an arbitrary collection of convex sets, that is for some index set I,

$$\mathscr{C} = \{C_i \subseteq \mathbb{R}^n \mid C_i \text{ is convex}, i \in I\}$$

The intersection of every element of $\mathscr{C}$, we will call the set S, and it is defined:

$$S := \bigcap_{i \in I} C_i$$

If $S = \emptyset$, then S trivially satisfies the definition. Otherwise, taking any two arbitrary, not necessarily unique elements $x_1, x_2 \in S$ we have that by definition of intersection,

$$x_1, x_2 \in C_i \qquad \text{for all } i \in I$$

and as C_i convex for all $i \in I$, for every $\theta \in [0, 1]$

$$\theta x_1 + (1 - \theta)x_2 \in C_i$$

so by definition of intersection we have, for all $\theta \in [0, 1]$, $x_1, x_2 \in S$

$$\theta x_1 + (1 - \theta)x_2 \in S$$

and hence S is convex. □

Now as we know that halfspaces and hyperplanes are convex, we see that the arbitrary intersections of any number of halfspaces and hyperplanes is also convex. Now we will state and prove our first theorem proved by Minowski[1]. This will be crucial in our future characterisation of convex sets.

2.3 The Separating Hyperplane Theorem

Theorem 2.3.1. *Let* $C, D \subseteq \mathbb{R}^n$ *be disjoint convex sets. Then there exists* $a \in \mathbb{R}^n \setminus \boldsymbol{0}, b \in \mathbb{R}^n$ *such that,*

$$\langle a, x\rangle \leq b \text{ for all } x \in C, \quad \text{and} \quad \langle a, x\rangle \geq b \text{ for all } x \in D.$$

If we require C, D *to be both closed and one set to be compact. Or we require* C, D *to both be open, then we have,*

$$\langle a, x\rangle < b \text{ for all } x \in C, \quad \text{and} \quad \langle a, x\rangle > b \text{ for all } x \in D.$$

In the first statement, we are saying that for every two disjoint, convex sets C, D, we can find a hyperplane $\Pi_{(a,b)}$ where any point in C is in the closed halfspace below the hyperplane and any point in D is in the closed halfspace above the hyperplane. We say this hyperplane separates C, D.

In the second statement, the separation is strict and no points in C or D lie on the hyperplane.

The requirements for a strict separation at first seem arbitrary. But in fact, it is this strict statement where both sets are closed and one is compact that is easier to prove and will lead us to the more general first statement. These requirements, for instance, allow us to prove this crucial lemma for which the proof of the Theorem 2.3.1 is close to immediate.

Lemma 2.3.1. *Suppose* $A, B \subseteq \mathbb{R}^n$ *are two disjoint, closed sets and A is compact. Then there exists* $a_0 \in A$ *and* $b_0 \in B$ *such that.*

$$||a_0 - b_0|| \leq ||a - b|| \quad \textit{for any } a \in A, b \in B$$

That is there exists a pair of points that minimise the distance between the two sets.

Proof.
Take any two points, $a \in A, b \in B$. Define $r_1 = ||b - a||$ By Heine-Borel, as A is compact, subset of $\mathbb{R}^n$, it is bounded by some ball with radius r_2. Let $S := B \cap \overline{B_{r_1+r_2}(a)}$. Then as S is non-empty, as $b \in S$ and as S is bounded as, the intersection of closed sets is closed. It follows that S is compact. Since the distance function is continuous and $A \times S$ is compact, the distance function attains its minimum on $A \times S$ at some points $(a_0, b_0) \in A \times S$, where $a_0 \in A,\ b_0 \in B$.

It remains to show that this is minimal on $A \times B$. Well suppose there exists $a' \in A,\ b' \in B$ such that $||b' - a'|| < ||b_0 - a_0||$. Well then as $||b_0 - a_0|| \leq r_1$, $||b' - a'|| < r_1$, and by triangle inequality.

$$||a - b'|| \leq ||a - a'|| + ||b' - a'|| < r_1 \ + \ r_2$$

Hence $b' \in S$ and $(a', b') \in A \times S$, which contradicts the fact that $||b_0 - a_0||$ is minimal on $A \times S$. Hence $(a_0,\ b_0)$ is minimal on $A \times B$.

□

Now from this lemma we can construct two quite natural tangent hyperplanes at the two closest points and then by convexity show that a tangent hyperplane, "sandwiched" between them must separate the sets.

Proof of Theorem 2.3.1.
By Lemma 2.3.1, there exists $c_0 \in C,\ d_0 \in D$ that minimise the distance function on $C \times D$. Since C, D are disjoint, we have $c_0 \neq d_0$. Consider the two hyperplanes,

$$v := d_0 - c_0$$

$$L_C = \{x \in \mathbb{R}^n \mid \langle x, v\rangle = \langle c_0, v\rangle\}, \qquad L_D = \{x \in \mathbb{R}^n \mid \langle x, v\rangle = \langle d_0, v\rangle\}$$

We now just need to show that for any $c \in C$, $\langle c, v\rangle \leq \langle c_0, v\rangle$ and for any $d \in D$ $\langle d, v\rangle \geq \langle d_0, v\rangle$

Suppose there is some $c' \in C$ such that $\langle c', v\rangle > \langle c_0, v\rangle$. Letting $r_1 = ||d_0 - x_0||$ as before, either $c' \in B_{r_1}(d_0)$ or it is not. Suppose it is, then $||d_0 - c'|| < ||d_0 - c_0||$, but this contradicts minimum distance. So $c' \notin B_{r_1}(d_0)$. But C is convex, so the line segment between c' and c_0 is wholly contained in C by convexity. By minimum distance the line segment between c' and c_0 and the ball $B_{r_1}(d_0)$ are disjoint. But then $\langle c, v\rangle \leq \langle c_0, v\rangle$ so contradiction.

An almost identical argument holds for $d \in D$. So take the midpoint between c_0, d_0 and as for any point in $c \in C$,

$$\langle c, v\rangle \leq \langle c_0, v\rangle < \langle c_0 + \frac{v}{2}, v\rangle$$

and for any $d \in D$

$$\langle d_0 - \frac{v}{2}, v\rangle < \langle d_0, v\rangle \leq \langle d, v\rangle$$

We have found a suitable *strictly* separating hyperplane.

For the proof of the non-strict case, we need slightly more complicated machinery. Please see [9]

□

Below we give a counterexample to demonstrate that this compactness is required for the strict version of the theorem to hold. Fundamentally this is because without compactness it is possible for the distance between two closed convex sets to approach zero. Understanding the above proof and this fact makes the above counter example easier to come up with.

Example 2.3.1. *Let* $C, D \subseteq \mathbb{R}^2$*, where* $C = \{(x, y) \in \mathbb{R}^2 \mid x \leq 0\}$,[3] *and* $D = \{(x, y) \in \mathbb{R}^2 \mid x > 0$ *and* $y \geq 1/x\}$

Well it does not take much convincing that C, D are disjoint, neither C, D are compact (Heine-Borel) but C, D are closed. Yet we can show that the only separating hyperplane is given $\Pi_{((1,0),\ 0)}$. Hence this is aa counterexample, showing we cannot relax compactness in the statement and maintain the strict inequality.

A counter example for the case where only one of the sets is open is easier to come up with.

Example 2.3.2. *Let* $C, D \subseteq \mathbb{R}^2$*, where* $C = \{(x, y) \in \mathbb{R}^2 \mid x \leq 0\}$*, and* $D = \{(x, y) \in \mathbb{R}^2 \mid x > 0\}$

We see that C, D are disjoint, D is open, yet the only hyperplane that separates the sets is given $\Pi_{((1,0),\ 0)}$ but this is a subset of C and hence not strict.

[3] We recognise this is the halfspace $\mathscr{H}_{((0,1),\ 0)}$

3 Convex Functions

3.1 Introduction to convex functions

Definition 3.1.1. *Let* $U \subseteq \mathbb{R}^n$ *be a convex set. We say that the function,* $f : U \subseteq \mathbb{R}^n \to \mathbb{R}$ *is convex if for every* $x, y \in U$ *and for every* $\theta \in [0,1]$*, we have*

$$\theta f(x) + (1-\theta)f(y) \geq f(\theta x + (1-\theta)y)$$

Definition 3.1.2. *We define* graph $f = \{(x, f(x)) \mid x \in U\} \subseteq \mathbb{R}^n \times \mathbb{R}$*. That is the* graph *of a function is the relation between the domain and codomain that defines the function.*

Definition 3.1.3. *We define* epigraph $f = \{(x, r) \mid r \geq f(x)\} \subseteq \mathbb{R}^n \times \mathbb{R}$

Lemma 3.1.1. $f : U \subseteq \mathbb{R}^n \to \mathbb{R}$ *is convex if and only if* epigraph f *is a convex set.*

Proof. Suppose *epigraph* f is a convex set. Suppose f is not a convex function. Then there exists $x, y \in U$ such that for some $\theta \in [0,1]$

$$\theta f(x) + (1-\theta)f(y) < f(\theta x + (1-\theta)y)$$

Well as U is convex. $(\theta x + (1-\theta)y, f(\theta x + (1-\theta)y) \in$ *epigraph* f but

$$\theta f(x) + (1-\theta)f(y) < f(\theta x + (1-\theta)y)$$

so $(\theta x + (1-\theta)y, \theta f(x) + (1-\theta)f(y) \notin$ *epigraph* f so *epigraph* f is not a convex set.
Hence if *epigraph* f is a convex set, f is a convex function.
Now suppose f is a convex function. Then for any $(x, r), (y, s) \in$ *epigraph* f. Then:

$$f(x) \leq r \quad f(y) \leq s$$

We have for any $\theta \in [0,1]$

$$\begin{aligned} f(\theta x + (1-\theta)y) &\leq \theta f(x) + (1-\theta)f(y) \\ &\leq \theta r + (1-\theta)s \end{aligned}$$

Hence we have

$$(\theta x + (1-\theta)y, \theta r + (1-\theta)s) \in \textit{epigraph } f$$

Hence *epigraph* f is a convex set. □

We have previously shown that any affine function is a convex function. This is because the epigraph of an affine function is its closed halfspace, which we have shown to be convex.

3.2 Operations that preserve convexity

Similar to the analysis of limits, sequences et cetera, we can define a 'calculus' of convex functions. This makes it a lot easier to show a certain

Proposition 3.2.1. *The non-negative weighted sum of two convex functions is a convex function. Formally, that is for two convex functions,*

$$f : U \subseteq \mathbb{R}^n \to \mathbb{R}$$

$$g : V \subseteq \mathbb{R}^n \to \mathbb{R}$$

Then the function

$$h : U \cap V \subseteq \mathbb{R}^n \to \mathbb{R} \quad \textit{defined,}$$

$$h(x) = \lambda f(x) + \mu g(x) + \eta$$

where $\lambda, \mu \geq 0, \eta \in \mathbb{R}$

is a convex function.

Proof. The proof is a fairly routine manipulation. For any $\theta \in [0,1]$ and $x, y \in U \cap V$

$$\begin{aligned} \theta h(x) + (1-\theta)h(y) &= \lambda\theta f(x) + \lambda(1-\theta)f(y) + \mu\theta g(x) + \mu(1-\theta)g(y) + \eta \\ &\geq f(\theta x + (1-\theta)y) + g(\theta x + (1-\theta)y) + \eta \\ &= h(\theta x + (1-\theta)y) \end{aligned}$$

And we are done. □

By inductive argument. The non-negative weighted combination of any finite number of convex functions is itself a convex function. We also get the degenerate cases that the sum of finitely many convex functions is convex, and that scaling a convex function by a non-negative constraint is convex.

Proposition 3.2.2. *The composition of a convex function with an affine function is convex. That is, let*

$$f : \mathbb{R}^n \to \mathbb{R}$$

be convex. Let

$$g : U \subseteq \mathbb{R}^m \to \mathbb{R}^n$$

be affine. That is there is some $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^n$ *where*

$$g(x) = Ax + b$$

. then $f \circ g$, $(f \circ g)(x) = f(Ax + b)$ *is convex.*

Proof. Similarly, this is just careful manipulation. For any $\theta \in [0,1]$ and $x, y \in U \cap V$

$$\begin{aligned} f(g(\theta x + (1-\theta)y)) &= f(A(\theta x + (1-\theta)y) + b) \\ &= f(\theta(Ax + b) + (1-\theta)(Ax + b)) \\ &\leq \theta f(Ax + b) + (1-\theta)f(Ax + b) \\ &= \theta f(g(x)) + (1-\theta)f(g(x)) \end{aligned}$$

Hence convex and we are done. □

3.3 Tangent hyperplanes

Definition 3.3.1. *Let U be a convex set. Let* $f : U \subseteq \mathbb{R}^n \to \mathbb{R}$ *be differentiable. We define the tangent hyperplane at* $x_0 \in U$ *to be*

$$T_{x_0} = \{(x, r) \mid r = f(x_0) + \langle \nabla f(x_0), x - x_0 \rangle\}$$

That is the hyperplane defined by the tangent vector to the graph of the function at x_0.

You will recognise this is the first Taylor expansion for the function around x_0.

Lemma 3.3.1. *Let* $U \subseteq \mathbb{R}^n$ *be a convex set. Let* $f : U \subseteq \mathbb{R}^n \to \mathbb{R}$ *be differentiable.* f *is a convex function if and only if for every* $x, y \in U$

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle$$

Proof. We prove the case where $n = 1$. Suppose f is convex. For any $x, y \in U$ and $\theta \in [0, 1]$, we have

$$\begin{aligned} f(\theta x + (1 - \theta)y) &= f(y + \theta(x - y)) \\ &\leq \theta f(x) + (1 - \theta) f(y) \end{aligned}$$

Rearranging,

$$f(x) \geq f(y) + \frac{f(y + \theta(x - y)) - f(y)}{\theta}$$

$$f(x) \geq f(y) + \frac{f(y + \theta(x - y)) - f(y)}{\theta(x - y)}(x - y)$$

and then taking $\theta(x - y) \to 0$

$$f(x) \geq f(y) + f'(y)(x - y)$$

Now for the reverse direction. Suppose that for every $x, y \in U$

$$f(x) \geq f(y) + f'(y)(x - y)$$

Then consider $z = \theta x + (1 - \theta)y$, $z \in U$ as U is convex. Well then we can consider the two following facts.

$$f(x) \geq f(z) + f'(z)(x - z)$$

$$f(y) \geq f(z) + f'(z)(y - z)$$

So by combining the inequalities, and then slogging through the algebra you have,

$$\theta f(x) + (1 - \theta) f(y) \geq f(z) = f(\theta x + (1 - \theta)y)$$

So hence f is convex.

For $n > 1$, the argument revolves around restricting the function to a line between points. We cover this in the appendix due to space constraints. □

From this inequality we get the following facts. First of all, if there exists $x \in U$, where $\nabla f(x) = 0$, then we have $f(y) \geq f(x)$ for any $y \in U$. In other words, we have taken a local property, a local minimum and using convexity taken it to a global property. It is a global maximum.

As well as this we see that for any $x \in U$. The tangent hyperplane T_x supports the *epigraph* f.

4 Convex Problems and Duality

4.1 Introduction to convex problems

After a significant amount of theory building, we now give the standard definition of a convex problem.

Definition 4.1.1.

$$\begin{aligned} \text{minimise} \quad & f(x) \\ \text{subject.} \quad & g_i(x) \leq 0 \quad i = 1, \dots, m \\ & h_j(x) = 0 \quad j = 1, \dots, n \end{aligned}$$

where $f, g_i, h_j : \mathbb{R}^n \to \mathbb{R}$. f, g_i *are convex, and* h_j *are linear.*

Fundamentally, we are minimising a convex function in a region determined by the intersection of convex sets defined by convex functions. We call this region the "feasible region". We require the equality constraints to be linear, as we can decompose $h_j(x) = 0$ into two inequality constraints $h(x) \leq 0$ and $-h(x) \leq 0$. Linear functions are the only functions for which both $h, -h$ are convex functions.

For the remainder of the essay, we will consider convex problems, where both the objective and the constraints are differentiable functions. This is to allow the scope to be more manageable.

Suppose that our function f is differentiable. We would like to apply our tools of calculus to find a local minimum of f and then use the argument from the previous section to show any local minimum is in fact a global minimum. Unfortunately if there are any constraints, we may find that the global minimiser is not in the feasible region and hence we have not found the minimal value of our function.

We will work through a toy problem to gain intuition and then formalise.

Consider the problem

$$\begin{aligned} \text{minimise} \quad & x_1 + x_2 \\ \text{subject.} \quad & x_1^2 + x_2^2 - 1 \leq 0 \end{aligned} \tag{4.1}$$

We are maximising the linear function, inside a feasible region $D \subset \mathbb{R}^2$ which area contained by the unit circle. In the current formulation, the circle constraint is considered "hard". Taking a value outside this region is impossible. Informally, we can consider if $x \notin D, f(x) = \infty$, it is a "worse" case than any possible value taken in the feasible region.

One way we could approximate this function on the extended reals, is by adding a penalty function. We write the penalty function as:

$$P : \mathbb{R} \to \overline{\mathbb{R}}$$

where

$$P(y) = \begin{cases} 0 & \text{if } y \leq 0 \\ \infty & \text{if } y > 0 \end{cases}$$

In which case, (4.1) is equivalent to our new problem

$$\min_x \quad x_1 + x_2 + P(x_1^2 + x_2^2 - 1) \tag{4.2}$$

As x_{opt} is the solution to (4.1) if and only if x_{opt} is the solution to (4.2).

Unfortunately adding our penalty function both introduces the extended real numbers and non-continuity to the mix. We can instead consider a linear penalty function.

For our constraint $x_1^2 + x_2^2 - 1 \leq 0$, we define the function we consider $P_u(x_1^2 + x_2^2 - 1) = u(x_1^2 + x_2^2 - 1)$, where $u \in \mathbb{R}$, $u \geq 0$.

For some u, consider the problem

$$\min_x \quad x_1 + x_2 + u(x_1^2 + x_2^2 - 1) \tag{4.3}$$

Note that this problem is not equivalent to those before. When the boundary condition is violated, it is punished by an amount proportional to its violation. When the boundary condition is satisfied, it is rewarded by an amount that is linearly proportional to its compliance.

If we instead consider the penalty function

$$\max_{u \geq 0} u(x_1^2 + x_2^2 - 1)$$

If we violate the constraint, we can make u as large as we like to make the penalty as large as we like. If we satisfy the constraint, we can make u as small as we like to make the reward as small as possible. So considering

$$\min_x \max_{u \geq 0} \quad x_1 + x_2 + u(x_1^2 + x_2^2 - 1) \tag{4.4}$$

This is a min-max problem and is equivalent to our previous problems (4.1) and (4.2).

We now use the following result to obtain a lower bound for our problem.

Proposition 4.1.1. *Let* $f : X \times Y \to Z$ *then*

$$\max_{x \in X} \min_{y \in Y} f(x, y) \leq \min_{y \in y} \max_{x \in X} f(x, y)$$

Proof. For all $x' \in X, y' \in Y$,

$$\min_{y \in Y} f(x', y) \leq f(x', y')$$

As true for all x', must be true at maximum

$$\max_{x \in X} \min_{y \in Y} f(x, y) \leq \max_{x \in x} f(x, y')$$

As true for all y', must be true at minimum

$$\max_{x \in X} \min_{y \in Y} f(x, y) \leq \min_{y \in Y} \max_{x \in x} f(x, y)$$

□

So by Proposition 4.1.1, we have a lower bound for our minimum. Namely,

$$\max_{u\geq 0}\min_{x} x_1 + x_2 + u(x_1^2 + x_2^2 - 1) \leq \min_{x}\max_{u\geq 0} x_1 + x_2 + u(x_1^2 + x_2^2 - 1)$$

We call this left side the dual problem, while we call the right hand side the primal problem. We can solve the left hand side analytically. Consider,

$$\min_{x} f(x_1, x_2) = \min_{x} x_1 + x_2 + u(x_1^2 + x_2^2 - 1)$$

To find, take the gradient of the function,

$$\nabla f = \begin{pmatrix} 1 + 2ux_1 \\ 1 + 2ux_2 \end{pmatrix}$$

Global minimum if and only if $\nabla f = 0$ (by convexity)

$$\text{So we have } x_1 = x_2 = -\frac{1}{2u}.$$

$$f(-\frac{1}{2u}, -\frac{1}{2u}) = -u - \frac{1}{2u}$$

So now maximising this function over $u \geq 0$. We have $u = -\frac{\sqrt{2}}{2}$. So we have $x_1 = x_2 = -\frac{\sqrt{2}}{2}$.

Note we have not solved the original, primal problem, but the dual problem. But due to Slater's condition, below, they are equivalent.

For are general convex optimisation problem, let us write it as before with linear penalty functions. Again, this is known as the primal problem.

$$\min_{x}\max_{\lambda_i\geq 0,\, \nu} f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) + \sum_{j=1}^{n} \nu_i h_j(x)$$

Definition 4.1.2. *For a convex problem, the associated Lagrangian,* $L : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$ *is defined*

$$L(x, \lambda, \nu) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) + \sum_{j=1}^{n} \nu_i h_j(x)$$

We define the Lagrange dual problem to be

$$g(\lambda, \nu) = \min_{x\in D} L(x, \lambda, \nu)$$

Note by Proposition 4.1.1.,

$$\max_{\lambda_i\geq 0,\nu} g(\lambda, \nu) \leq \min_{x}\max_{\lambda_i\geq 0,\, \nu} f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) + \sum_{j=1}^{n} \nu_i h_j(x)$$

This inequality is known as weak duality. That is that the solution to the dual problem bounds the primal problem. When this inequality is an equality, the problems exhibits what is known as strong duality.

A sufficient condition is Slater's Condition. We omit the proof due to its technicality.

Theorem 4.1.1. *Consider a convex problem as defined in (4.1.1). Suppose there exists* $x \in \mathbb{R}^n$ *such that* $g_i(x) < 0$, $h_j(x) = 0$. *We say x is strictly feasible. Then we have strong duality in the problem.*

Bibliography

1. Minkowski, H. *Geometrie der Zahlen* (Teubner, Leipzig, 1896).
2. Holley, J. Vanguard Mathematician George Dantzig Dies. *The Washington Post.* Obituary of George B. Dantzig, age 90, B06 (May 18, 2005).
3. Stigler, G. J. The Cost of Subsistence. *Journal of Farm Economics* **27,** 303–314 (1945).
4. Fenchel, W. On Conjugate Convex Functions. *Canadian Journal of Mathematics* **1,** 73–77 (1949).
5. Kuhn, H. W. & Tucker, A. W. in *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability* (ed Neyman, J.) 481–492 (University of California Press, Berkeley, CA, 1951). `https://projecteuclid.org/euclid.bsmsp/1200500249`.
6. Karmarkar, N. A New Polynomial-Time Algorithm for Linear Programming. *Combinatorica* **4,** 373–395 (1984).
7. The New York Times. BREAKTHROUGH IN PROBLEM SOLVING. *New York Times.* Front-page feature on Karmarkar's algorithm, A1.
8. Nesterov, Y. *Introductory Lectures on Convex Optimization: A Basic Course* (Kluwer Academic Publishers, Boston, MA, 2004).
9. Boyd, S. & Vandenberghe, L. *Convex Optimization* (Cambridge University Press, Cambridge, UK, 2004).
10. Cauchy, A.-L. Méthode générale pour la résolution des systèmes d'équations simultanées. *Comptes Rendus de l'Académie des Sciences* **25,** 536–538 (1847).
11. Polyak, B. Some Methods of Speeding up the Convergence of Iteration Methods. *USSR Computational Mathematics and Mathematical Physics* **4,** 1–17 (1964).
12. Nesterov, Y. A Method for Solving the Convex Programming Problem with Convergence Rate O(1/k^2). *Soviet Mathematics Doklady* **27,** 372–376 (1983).
13. Duchi, J., Hazan, E. & Singer, Y. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research* **12,** 2121–2159 (2011).
14. Tieleman, T. & Hinton, G. *RMSProp: Divide the Gradient by a Running Average of Its Recent Magnitude* tech. rep. (COURSERA: Neural Networks for Machine Learning, Lecture 6.5, 2012).
15. Kingma, D. P. & Ba, J. *Adam: A Method for Stochastic Optimization* in *3rd International Conference on Learning Representations (ICLR)* (2015). `https://arxiv.org/abs/1412.6980`.

First Order Methods

To oversimplify, inner point methods generally rely on two things: a numerical unconstrained optimiser and a method to change the convex constrained problem into an unconstrained one. Furthermore, unconstrained optimisers can be categorised as first, second, or higher order solvers. The order of the solver refers to the order of the derivative the optimiser takes into account when iterating towards an optimal point.

Unfortunately, like most things in the real world, the choice of both the order of the optimiser and the optimiser itself both have advantages and disadvantages. This is dependent on the form of the problem, both if the problem exhibits complexity, or the n-differentiability of the function. Generally, second-order methods converge faster to a solution, they compute a Hessian matrix and using that information move in an optimal direction. Newton's Method, the multivariable case of the Newton-Rhapson method, is a second-order solver and often the main machinery behind an inner point method. First order methods have the advantage when the problem is high-dimensional, Hessian calculation becomes computationally intensive, and when the function is poorly behaved.

An example of a non-convex, high dimensional problem is neural network training. In a neural network, weights and biases need to be set to optimise a performance metric. Whether that be, the percentage chance of correctly identifying a cat, or correctly predicting the fair market value of a house. Due to the dimensionality we use first order solvers, and by borrowing ideas from convex analysis and modifying to fit the 'non-convexity' of the problem.

In this appendix, I will reference the optimiser's original paper but provide python implementations on a toy example of polynomial regression. Here, given some data points and the order of our target polynomial, we aim to output the optimal polynomial that is the 'best fit'. The 'best fit' is average of the squares of the errors between the data point and the polynomial, the mean squared error or MSE.

Below is the code to set up the problem, to make it easier to follow along.

```
# Import libraries
import numpy as np
import matplotlib.pyplot as plt
import random as rd

# Set random seeds
np.random.seed(42)
rd.seed(42)

# Create random ring
rng = np.random.default_rng()
```

```
def polynomial(coefficients: list, input: float):
  '''
  Evaluates a polynomial defined, a_0 + a_1 * x + ... + a_n * x ^ n, where coefficients = [a_0,
      a_1, ... a_n], input = x
  '''
  sum = 0
  for i in range(len(coefficients)):
    sum += coefficients[i] * (input ** i)
  return sum

def MSE(points: list, coefficients: list):
  '''
  Calculates MSE for a list of points, to a polynomial with given coefficients
  '''
  n = len(points)
  sum = 0

  for i in range(len(points)):
    sum += (points[i][1] - polynomial(coefficients, points[i][0])) ** 2

  return 1/n * sum

def negative_gradient_polynomial(points, coefficients):
  '''
  Numerically calculates the gradient for the coefficients at this point
  '''
  grad = []

  for j in range(len(coefficients)):
    sum = 0
    for i in range(len(points)):
      sum += (points[i][0] ** j) * (points[i][1] - polynomial(coefficients, points[i][0]))
    grad.append(2 *sum/len(points))

  return grad
```

For a greater spotlight of the methods and their pseudocode, please see the original papers in the bibliography.

The first, first order method we will discuss is the basis for every one of the others. In gradient descent[10], we start at the origin in the coefficient space. We then calculate the gradient vector of the MSE at that point and take a small step in that direction. We iteratively repeat this process descending to a minimum.

It is prudent to mention that in gradient descent you can take all data into account or they can take batches of data or singular data points into account. Tis is called batch (stochastic) gradient descent and stochastic gradient descent respectively.

```
  def gradient_descent(points, initial_coefficients, step_size, trials):
  '''
```

```
Given points and inital coefficients, iteratively descends to attempt to reach optimum
'''
coefficients = initial_coefficients
mse_history = []
coefficients_history = []

for j in range(trials):
  gradient_vector = negative_gradient_polynomial(points, coefficients)
  coefficients = [coefficients[i] + gradient_vector[i] * step_size for i in
      range(len(coefficients))]

  mse_history.append(MSE(points, coefficients))
  coefficients_history.append(list(coefficients))

return (coefficients, mse_history, coefficients_history)
```

An issue with gradient descent is that if the gradient is shallow, it may take a long time to converge. To mitigate this we add the concept of momentum. In momentum gradient descent[11], the descent calculation is almost the same. However, as well as a step determined by the direction of the gradient, we take a smaller step in the direction of our previous step. If we descend in the same direction repeatedly, then momentum will accrue leading to faster convergence.

```
  def momentum_gradient_descent(points, initial_coefficients, step_size, trials,
      momentum_decay, initial_momentum):
'''
Similar to gradient descent but with added momentum,
Note initial_momentum is a list with the same length of initial_coefficients, should be
    initialised np.zeros(len(inital_coefficients))
'''
coefficients = initial_coefficients
dimension = len(coefficients)
momentum = initial_momentum
mse_history = []
coefficients_history = []
momentum_history = []

for j in range(trials):
  gradient_vector = negative_gradient_polynomial(points, coefficients)
  momentum = [momentum_decay * momentum[i] + step_size * gradient_vector[i] for i in
      range(dimension)]
  coefficients = [coefficients[i] + momentum[i] for i in range(dimension)]

  mse_history.append(MSE(points, coefficients))
  coefficients_history.append(list(coefficients))
  momentum_history.append(momentum)
```

```
  return(coefficients, mse_history, coefficients_history)
```

Adding to the concept of momentum, we have Nesterov momentum[12]. This works like Nesterov momentum but with a 'look ahead' term. By calculating the potential next points gradient it can adjust its path accordingly.

```
def nesterov_momentum_gradient_descent(points, initial_coefficients, step_size, trials,
    momentum_decay, initial_momentum):
  '''
  Look ahead momentum technique, not that useful for stochastic gradient descent
  '''
  coefficients = initial_coefficients
  dimension = len(coefficients)
  momentum = initial_momentum
  mse_history = []
  coefficients_history = []
  momentum_history = []

  for j in range(trials):
    forward_coefficients = [coefficients[i] + momentum_decay * momentum[i] for i in
        range(dimension)]
    gradient_vector = negative_gradient_polynomial(points, forward_coefficients)
    momentum = [momentum_decay * momentum[i] + step_size * gradient_vector[i] for i in
        range(dimension)]
    coefficients = [coefficients[i] + momentum[i] for i in range(dimension)]

    mse_history.append(MSE(points, coefficients))
    coefficients_history.append(list(coefficients))
    momentum_history.append(momentum)

  return(coefficients, mse_history, coefficients_history)
```

In addition to momentum methods, we also have adaptive methods. Adaptive methods change the size of the step relative to their previous steps. The first method we implement is AdaGrad[13]. AdaGrad remembers which directions it has frequently moved in and those in which it has not. It moves slower in frequent directions, and faster in infrequent directions.

```
  def AdaGrad(points, initial_coefficients, base_step_size, trials):
  '''
  Effectively stores the previous mean squared error for each dimension
  Issues with monotonicity decreasing decay
  '''
  epsilon = 1e-8

  coefficients = initial_coefficients
```

```
dimension = len(coefficients)
mse_history = []
coefficients_history = []
historical_grad_square = np.zeros(dimension)

for j in range(trials):
  gradient_vector = negative_gradient_polynomial(points, coefficients)
  historical_grad_square = [historical_grad_square[i] + gradient_vector[i] ** 2 for i in
      range(dimension)]
  coefficients = [coefficients[i] + (base_step_size * gradient_vector[i]/(epsilon +
      historical_grad_square[i])**(0.5)) for i in range(dimension)]

  mse_history.append(MSE(points, coefficients))
  coefficients_history.append(list(coefficients))

return(coefficients, mse_history, coefficients_history)
```

Often we want to reduce the scope of the memory, to prevent iterations at the start having a sizeable impact at the end. This is implemented by RMSProp[14] where we add an exponential decay factor to the memory.

```
def RMSProp(points, initial_coefficients, base_step_size, trials, decay_rate):
'''
Mitigates the decay of AdaGrad by acting in some effective window, it does this with an
    exponential decay factor
'''
epsilon = 1e-8

coefficients = initial_coefficients
dimension = len(coefficients)
mse_history = []
coefficients_history = []
trailing_average = np.zeros(dimension)

for i in range(trials):
  gradient_vector = negative_gradient_polynomial(points, coefficients)
  trailing_average = [decay_rate * trailing_average[j] + (1 - decay_rate) *
      (gradient_vector[j] ** 2) for j in range(dimension)]
  coefficients = [coefficients[j] + (base_step_size * gradient_vector[j]/(epsilon +
      trailing_average[j])**(0.5)) for j in range(dimension)]

  mse_history.append(MSE(points, coefficients))
  coefficients_history.append(list(coefficients))

return(coefficients, mse_history, coefficients_history)
```

We can combine momentum methods and adaptive methods into adaptive momentum methods, or how it is commonly known, Adam. The majority of industry neural networks use Adam[15] or a derivative to train their models.

```
def Adam(points, initial_coefficients, base_step_size, trials, first_decay, second_decay):
 epsilon = 1e-8

 coefficients = initial_coefficients
 dimension = len(coefficients)
 mse_history = []
 coefficients_history = []
 first_moment = np.zeros(dimension)
 second_moment = np.zeros(dimension)

 for i in range(trials):
  gradient_vector = negative_gradient_polynomial(points, coefficients)
  first_moment = [first_decay * first_moment[j] - (1 - first_decay) * gradient_vector[j] for j
      in range(dimension)]
  second_moment = [second_decay * second_moment[j] + (1 - first_decay) * (gradient_vector[j]
      ** 2) for j in range(dimension)]

  unbiased_first_moment = [first_moment[j] / (1 - (first_decay ** (i + 1))) for j in
      range(dimension)]
  unbiased_second_moment = [second_moment[j] / (1 - (second_decay ** (i + 1))) for j in
      range(dimension)]

  coefficients = [coefficients[j] - (base_step_size * unbiased_first_moment[j])/(epsilon +
      (unbiased_second_moment[j]) ** 0.5 ) for j in range(dimension)]

  mse_history.append(MSE(points, coefficients))
  coefficients_history.append(list(coefficients))

 return(coefficients, mse_history, coefficients_history)
```