MSE2122 - Nonlinear Optimization Lecturer Notes I

Fernando Dias (based on previous version by Fabricio Oliveira)

September 3, 2023

Abstract

In this lecture, we introduce the concept of nonlinear optimisation problems and the difference between mathematical programming and optimisation. We also discuss a few essential examples to motivate future developments throughout the course.

Contents

1 What is optimisation?

Let us start with a simple definition, according to Oxford Dictionary:

Optimization (or optimisation, with British spelling):

- (*noun*) the action of making the **best** or **most effective** use of a **situation** or **resource**.

An **optimisation** is one of these words that has many meanings, depending on the context you take as a reference. In the context of this course, optimisation refers to **mathematical optimisation**, a subfield of applied mathematics.

In **mathematical optimisation**, we build upon concepts and techniques from basic mathematical subfields. From **calculus**, **analysis**, **linear algebra**, and other, we to model and develop **methods** that allow us to find values for variables within a given domain that **maximise** (or **minimise**) the value of a **function**. In specific, we are trying to solve the following general problem:

$$\begin{aligned} &\text{min. } f(x) \\ &\text{subject to: } x \in X. \end{aligned} \tag{1}$$

where $x \in \mathbb{R}^n$ is a vector of n variables, $f : \mathbb{R}^n \mapsto \mathbb{R}$ is a **function** to be optimised (minimised) and $X \subseteq \mathbb{R}^n$ is a **domain** containing **potential** values for x.

In a general sense, these problems can be solved by employing the following strategy:

- **Analysing/visualising** properties of functions under specific domains and deriving the conditions that must be satisfied such that a point x is a candidate optimal point.
- **Applying** numerical methods that iteratively search for points satisfying these conditions.

This idea is central in several domains of knowledge, and very often are defined under area-specific nomenclature. Fields such as economics, engineering, statistics, machine learning and, perhaps more broadly, operations research, are intensive users and developers of optimisation theory and applications. In addition, more and more fields, are recurring to operations research to find innovative solutions.

1.1 Mathematical programming and optimisation

Operations research and mathematical optimization are somewhat **intertwined**, as they were both born in similar circumstances. In the decades after the two world wars, o**perations research** tools were more widely applied to problems in **business**, **industry**, and **engineering**. Since that time, operation research has expanded into a field widely used in industries ranging from petrochemicals to airlines, finance, logistics, and government, moving to a focus on the development of mathematical models that can be used to analyze and optimize sometimes complex systems. It has become an area of active academic and industrial research.

With the development of computers over the next three decades after WWI, **Operations Research** can now solve problems with hundreds of thousands of variables and constraints. Moreover, the large volumes of data required for such problems can be stored and manipulated very efficiently. Much of operations research relies upon stochastic variables and, therefore, access to truly random numbers. Fortunately, the cybernetics field also required the same level of randomness. More recently, the operations research approach, which dates back to the 1950s, has been criticized for being a collection of **mathematical models** but lacking an empirical basis for data collection for applications.

Let us separate **mathematical programming** from (mathematical) **optimization**. Mathematical programming is a modelling paradigm in which we rely on (compelling) analogies to model *real-world* problems. In that, we look at a given decision problem considering that:

- **variables** represent *decisions* or *interest*, as in a business decision or a course of action. Examples include setting the parameter of (e.g., prediction) model, production systems layouts, geometries of structures, topologies of networks, and so forth;

- **domain** represents business rules or *constraints* and *limitations*, representing logic relations, design or engineering limitations, requirements, and such;

- function is an *objective function* that measures solution quality, profit or goal.

With these in mind, we can represent the decision problem as a **mathematical programming model** of the form of (1) that can be solved using **optimization** methods. From now on, we will refer to this specific class of models as mathematical optimization models or optimization models for short. We will also use the term to **solve the problem** to refer to finding optimal solutions to optimization models.

This course mostly focuses on the **optimization techniques** employed to find **optimal solutions** for these models. As we will see, depending on the nature of the functions f and g used to formulate the model, some methods might be more or less appropriate. Further complicating the issue, for models of a given nature, there might be alternative algorithms that can be employed and with no generalized consense whether one method is generally better performing than another.

Therefore, in this course, we are focusing on **optimization** methods to find optimal solutions considering that either f and/or g are **nonlinear** functions, hence the name of the course.

1.2 Types of mathematical optimisation models

In general, **the rule of thumb** for solving a problem is :

The simpler the assumptions on the parts forming the optimisation model, the more efficient the methods for solving it.

Let us define some additional notation that we will use from now on. Consider a model in the **general (or standard) form**:

$$\begin{aligned} \text{min. } & f(x) \\ \text{subject to: } & g_i(x) \leq 0, i = 1, \dots, m \\ & h_i(x) = 0, i = 1, \dots, l \\ & x \in X, \end{aligned}$$

where $f : \mathbb{R}^n \mapsto \mathbb{R}$ is the objective function, $g : \mathbb{R}^m \mapsto \mathbb{R}^m$ is a collection of m **inequality** constraints and $h : \mathbb{R}^n \mapsto \mathbb{R}^l$ is a collection of l **equality** constraints.

Remark: in fact, an **equality** constraint can represent every inequality constraint by making $h_i(x) = g_i(x) + x_{n+1}$ and augmenting the decision variable vector $x \in \mathbb{R}^n$ to include the **slack variable** x_{n+1}. However, since these constraints are of a very different nature, we will explicitly represent both whenever necessary.

Remark: be aware that some mathematicians and books might refer to both type of inequalities as a single one

The most general types of models are the following. We also use this as an opportunity to define some (admittedly confusing) nomenclature from the operations research field that we will use in these notes.

- **Unconstrained models:** in these, the set $X = \mathbb{R}^n$ and $m = l = 0$. These are prominent in, e.g., machine learning and statistics applications, where f represents a measure of model fitness or prediction error.

- **Linear programming (LP):** presumes linear objective function. $f(x) = c^\top x$ and constraints g and h affine, i.e., of the form $a_i^\top x - b_i$, with $a_i \in \mathbb{R}^n$ and $b \in \mathbb{R}$. Normally, $X = \{x \in \mathbb{R}^n \mid x_j \geq 0, j = 1, \dots, n\}$ enforce that decision variables are constrained to be the nonnegative orthant.

- **Nonlinear programming (NLP):** some or all of the functions f, g, and h are nonlinear.

- **Mixed-integer (linear) programming (MIP):** consists of an LP in which some (or all) of the variables are constrained to be integers. In other words, $X \subseteq \mathbb{R}^k \times \mathbb{Z}^{n-k}$. Very frequently, the integer variables are binary terms, i.e., $x_i \in \{0,1\}$, for $i = 1, \dots, n-k$ and are meant to represent true-or-false or yes-or-no conditions.

- **Mixed-integer nonlinear programming (MINLP):** are the intersection of MIPs and NLPs.

See Diagram below:

Remark: notice that we use the vector notation $c^\top x = \sum_{j \in J} c_j x_j$, with $J = \{1, \dots, N\}$. This is just a convenience for keeping the notation compact.

2 Examples of applications

We now discuss a few examples to illustrate the nature of the problems to which we will develop solution methods and their applicability to real-world contexts.

2.1 Resource allocation and portfolio optimisation

In a general sense, any mathematical optimisation model is an instantiation of the *resource allocation problem.* A resource allocation problem consists of how to design an optimal allocation of resources to tasks, such that a given outcome is optimised.

Classical examples typically include production planning settings, in which raw materials or labour resources are inputted into a system and a collection of products, a production plan, results from this allocation. The objective is to find the best production plan, that is, a plan with the maximum profit or minimum cost. Resource allocation problems can also appear in a less obvious setting, where the resources can be the capacity of transmission lines in an energy generation planning setting, for example.

Let $i \in I = \{1, \dots, M\}$ be a collection of resources and $j \in J = \{1, \dots, N\}$ be a collection of products. Suppose that, to produce one unit of product j, a quantity a_{ij} of resource i is required. Assume that the total availability of resource i is b_i and that the return per unit of product j is c_j.

Let x_j be the decision variable representing total of product j produced. The resource allocation problem can be modelled as:

$$\begin{aligned} \text{max.} \ & \sum_{j \in J} c_j x_j && (2) \\ \text{subject to:} \ & \sum_{j \in J} a_{ij} x_j \le b_i, \ \forall i \in I && (3) \\ & x_j \ge 0, \ \forall j \in J. && (4) \end{aligned}$$

Equation (2) represents the objective function, in which we maximise the total return obtained from a given production plan. Equation (3) quantify the resource requirements for a given production plan and enforce that such a requirement does not exceed the resource availability. Finally, constraint (4) defines the domain of the decision variables.

Notice that, as posed, the resource allocation problem is linear. This is perhaps the most basic, and also most diffused setting for optimisation models for which very reliable and mature technology is available. In this course, we will concentrate on methods that can solve variants of this model in which the objective function and/or the constraints are required to include nonlinear terms.

One classic variant of resource allocation that include nonlinear terms is the *portfolio optimisation problem.* In this, we assume that a collection of assets $j \in J = \{1, \dots, N\}$ are available for investment. In this case, capital is the single (actual) resource to be considered. Each asset has random return R_j, with an expected value $\mathbb{E}[R_j] = \mu_j$. Also, the covariance between two assets $i, j \in J$ is given by $\sigma_{ij} = \mathbb{E}[(R_i - \mu_i)(R_j - \mu_j)]$, which can be denoted as the covariance matrix:

$$\Sigma = \begin{bmatrix} \sigma_{11} & \dots & \sigma_{1N} \\ \vdots & \ddots & \vdots \\ \sigma_{N1} & \dots & \sigma_{NN} \end{bmatrix}$$

Markowitz (1952) proposed using $x^\top \Sigma x$ as a risk measure that captures the variability in the return of the assets. Given the above, the optimisation model that provides the investment portfolio with the least risk, given a minimum requirement ϵ in terms of expected returns is given by:

$$\begin{aligned} \text{min. } & x^\top \Sigma x && (5)\\ \text{subject to: } & \mu^\top x \geq \epsilon && (6)\\ & 0 \leq x_j \leq 1, \ \forall j \in J. && (7) \end{aligned}$$

Objective function (5) represents the portfolio risk to be minimised, while constraint (6) enforces that the expected return must be at least ϵ. Notice that ϵ can be seen as a resource that has to be (at least) completely depleted, if one wants to do a parallel with the resource allocation structure discussed early. Constraint (7) defined the domain of the decision variables. Notice how the problem is posed in a scaled form, where $x_j \in [0, 1]$ represents a percentage of a hypothetical available capital for investment.

In this example, the problem is nonlinear due to the quadratic nature of the objective function $x^\top \Sigma x = \sum_{i,j \in J} \sigma_{ij} x_i x_j$. As we will see later on, there are efficient methods that can be employed to solve quadratic problems like this.

2.2 The pooling problem: refinery operations planning

The *pooling problem* is another example of a resource allocation problem that naturally presents nonlinear constraints. In this case, the production depends on *mixing operations*, known as pooling, to obtain certain product specification for a given property.

As an illustration, suppose that products $j \in J = \{1, \dots, N\}$ are produced by mixing byproducts $i \in I_j \subseteq I = \{1, \dots, M\}$. Assume that the qualities of byproducts q_i are known and that there is no reaction between byproducts. Each product is required to have a property value q_j within an acceptable range $[\underline{q}_j, \overline{q}_j]$ to be classified as product j. In this case, mass and property balances are calculated as:

$$x_j = \sum_{i \in I_j} x_i, \ \forall j \in J \tag{8}$$

$$q_j = \frac{\sum_{i \in I_j} q_i x_i}{x_j}, \ \forall j \in J. \tag{9}$$

These can then incorporated into the resource allocation problem accordingly. One key aspect associated with pooling problem formulations is that the property balances represented by (9) define *nonconvex* feasibility regions. As we will see later, convexity is a powerful property that allows for developing efficient solution methods and its absence typically compromises computational performance and tractability in general.

2.3 Robust optimisation

Robust optimisation is a subarea of mathematical programming concerned with models that support decision-making under *uncertainty*. In specific, the idea is to devise a formulation mechanism that can guarantee feasibility of the optimal solution in face of variability, ultimately taking a risk-averse standpoint.

Consider the resource allocation problem from Section 2.1. Now, suppose that the parameters $\tilde{a}_i \in \mathbb{R}^N$ associated with a given constraint $i \in I = \{1, \dots, M\}$ are uncertain with a unknown probability distribution. The resource allocation problem can then be formulated as:

$$\begin{aligned} \text{max. } & c^\top x \\ \text{subject to: } & \tilde{a}_i^\top x \leq b_i, \ \forall i \in I \\ & x_j \geq 0, \ \forall j \in J. \end{aligned}$$

Let us assume that the only information available are observations $\hat{a}_i$, from which we can estimate a nominal value $\overline{a}_i$. This is illustrated in Figure 1, in which 100 random observations are generated for $\tilde{a}_i = [\tilde{a}_{i1}, \tilde{a}_{i2}]$ with $\tilde{a}_{i1} \sim \text{Normal}(10, 2)$ and $\tilde{a}_{i2} \sim \text{Normal}(5, 3)$ for a single constraint $i \in I$. The nominal values are assumed to have coordinates given by the average values used in the Normal distributions.

Our objective is to develop a model that incorporates a given level of protection in terms of feasibility guarantees. That is, we would like to develop a model that provides solutions that are guaranteed to

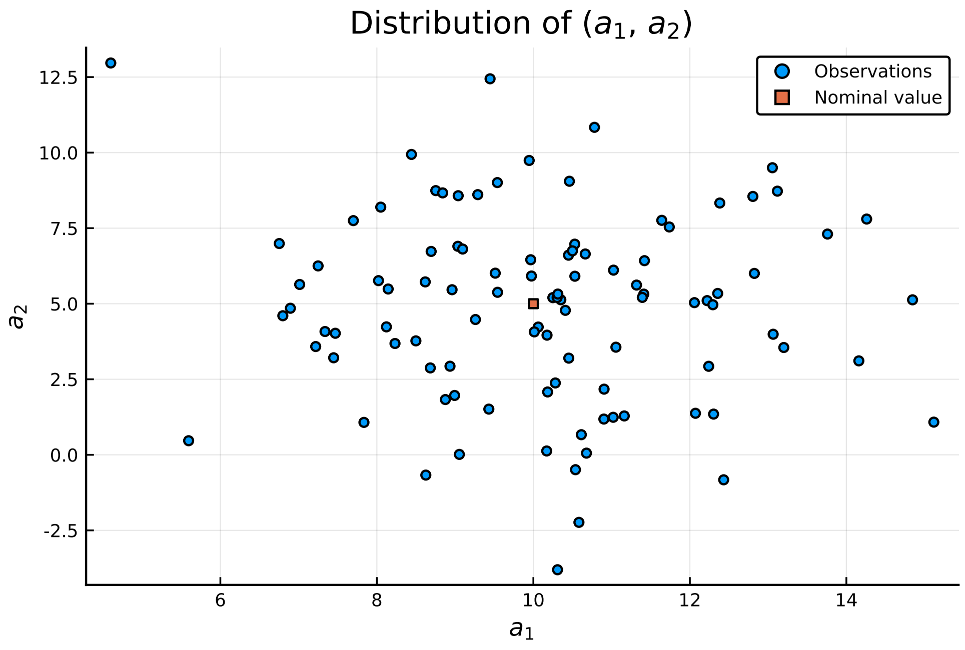

Figure 1: One hundred random realisations for $\tilde{a}_i$.

remain feasible if the realisation of $\tilde{a}_i$ falls within an *uncertainty set* ϵ_i of size controlled by the parameter Γ_i. The idea is that the bigger the uncertainty set ϵ_i, the more robust is the solution, which typically comes at the expense of accepting solutions with expected worse performance.

The tractability of robust optimisation models depends on the geometry of the uncertainty set employed. Let us assume in what follows that:

$$\epsilon_i = \{\overline{a}_i + P_i u \mid ||u||_2 \leq \Gamma_i\} \tag{10}$$

is an ellipsoid with the characteristic matrix P_i (i.e., its eigenvalues show how the ellipsoid extends in every direction from $\overline{a}_i$) and Γ_i employs a scaling of the ellipsoid size.

Remark: an alternative (perhaps more frequent) characterisation of an ellipsoid $\epsilon \subset \mathbb{R}^n$ centred at $\overline{x}$ is given by $\epsilon = \{x \in \mathbb{R}^n \mid (x - \overline{x})^\top A(x - \overline{x}) = 1\}$. By making $A = P^{-2}$, we recover the representation in (10).

We can now formulate the *robust counterpart*, which consists of a risk-averse version of the original resource allocation problem. In that, we try to anticipate the worst possible outcome and make decisions that are both optimal and guarantee feasibility in this worst-case sense. This standpoint translates into the following optimisation model.

$$\begin{aligned} \text{max.} \ & c^\top x \\ \text{subject to:} \ & \max_{a_i \in \epsilon_i} \{a_i^\top x\} \leq b_i, \ \forall i \in I \\ & x_j \geq 0, \forall j \in J. \end{aligned} \tag{11}$$

Notice how the constraint (11) has an embedded optimisation problem, turning into a *bi-level optimisation* problem. This highlights the issue associated with tractability, since solving the whole problem strongly depends on deriving tractable equivalent reformulations.

Assuming that the uncertainty set ϵ_i is an ellipsoid, the following result holds.

$$\max_{a_i \in \epsilon_i} \{a_i^\top x\} = \overline{a}_i^\top x + \max_u \{u^\top P_i x : ||u||_2 \leq \Gamma_i\} \tag{12}$$

$$= \overline{a}_i^\top x + \Gamma_i ||P_i x||_2. \tag{13}$$

In (12), we recast the inner problem in terms of the ellipsoidal uncertainty set, ultimately meaning that we recast the inner maximisation problem in terms of variable u. Since the only constraint is $||u||_2 \leq \Gamma_i$, in (13) we can derive a closed form for the inner optimisation problem.

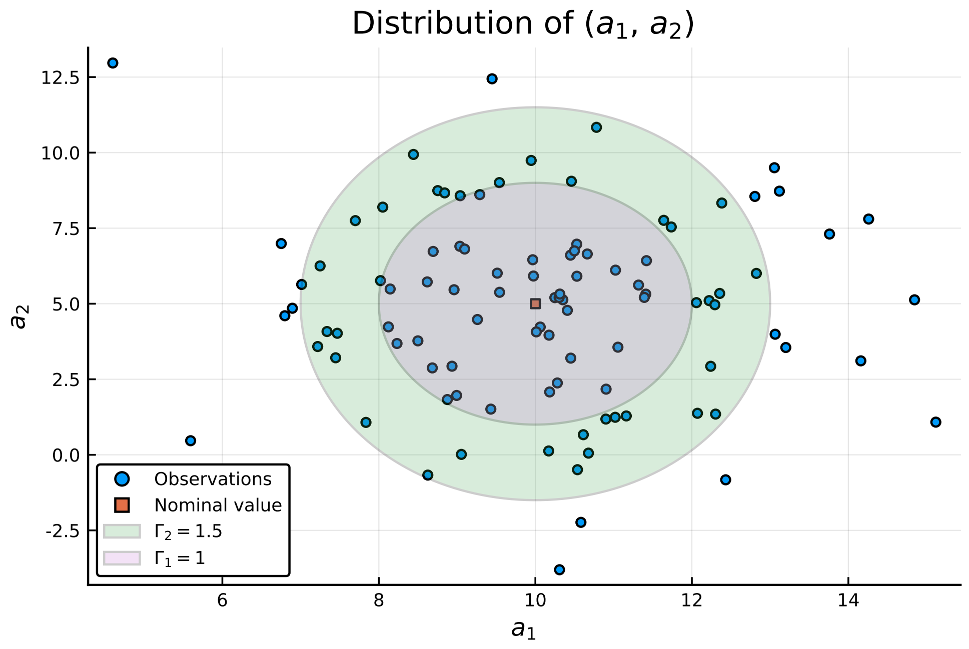

Figure 2: One hundred random realisations for $\tilde{a}_i$.

With the closed form derived in (13), we can reformulate the original bi-level problem as a tractable single-level problem of the following form:

$$\begin{aligned} \text{max.} \quad & c^\top x \\ \text{subject to:} \quad & \overline{a}_i^\top x + \Gamma_i ||P_i x||_2 \leq b_i, \ \forall i \in I \\ & x_j \geq 0, \ \forall j \in J. \end{aligned} \tag{14}$$

Notice how the term $\Gamma_i||P_i^\top x||_2$ creates a buffer for constraint (14), ultimately preventing the complete depletion of the resource. Clearly, this will lead to a suboptimal solution when compared to the original deterministic at the expense of providing protection against deviations in coefficients a_i. This difference is often referred to as the *price of robustness.*

In Figure 2, we show the ellipsoidal sets for two levels of Γ_i for a single constraint i. We define:

$$\epsilon_i = \{ \begin{bmatrix} 10 \\ 5 \end{bmatrix} + \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \} \tag{15}$$

using the average and standard deviation of the original distributions that generated the observations. We plot the ellipsoids for $\Gamma_1 = 1$ and $\Gamma_2 = 1.5$, illustrating how the protection level increases as Γ increases. This can be inferred since the uncertainty set covers more of the observations and the formulation is such that feasibility is guaranteed for any observation within the uncertainty set.

2.4 Combinatorial Optimization

Combinatorial optimization is a sub-field of mathematical optimization that consists of finding an optimal solution from a finite **set** of possible solutions from a discrete **set**. Examples of **combinatorial problems** are the **travelling salesman problem** ("TSP"), the minimum spanning tree problem ("MST"), and the knapsack problem. The most trivial solution for such problems is exhaustive and enumerative search search, although it is not tractable for larger solutions.

There is a large amount of literature on polynomial-time algorithms for certain special classes of discrete optimization. The theory of linear programming unifies a considerable amount of it. Some examples of combinatorial optimization problems this framework covers are shortest paths and shortest-path trees, flows and circulations, spanning trees, matching, and matroid problems.

Many problems covered by linear optimization could also be described as combinatorial optimization problems (as part of discrete optimization). For example, shortest path, flow and circulations, spanning tree, matching, etc. In this theoretical framework, research on such problems focuses on the following topics: solution for special cases of discrete problems using specific parameters (fixed-parameter tractable

(FPT) problems); approximation algorithms; heuristics and meta-heuristics; parameterized optimization (variation of FPT); NP-completeness.

Combinatorial optimization problems can be viewed as searching for the best solution amongst a set of viable solutions. Perhaps, as a search procedure. Many algorithms, such as branch-and-bound, branch-and-cut, dynamic programming, tabu search, can solve many variations of discrete problems. However, generic search algorithms are not guaranteed to find an optimal solution first, nor are they guaranteed to run quickly (in polynomial time). Since some discrete optimization problems are NP-complete, such as the travelling salesman (decision) problem (unless P=NP).

We can formally define a combinatorial optimization problem using the following notation:

- a set of **candidate** solutions I;
- a subset of **feasible** solutions $f(x)$, such that $x \in I$;
- the performance **measure** of the feasible solutions $m(x, y)$;
- the **goal** function (generally minimization or maximization)

In this way, *an optimal solution can be found* when:

$$m(x, y) = g\{m(x, y')|y^{'} \in f(x)\} \tag{16}$$

Each combinatorial optimization problem has a corresponding decision problem that asks whether there is a feasible solution for some particular measure m_0.

The field of approximation algorithms deals with algorithms to find near-optimal solutions to challenging problems where exact and fast solutions are costly (and sometimes, impossible).

Classical examples of combinatorial problems are:

- TSP (**Travelling Salesman Problem**):
  This problem can be defined as:

  > *Given a set of cities and the distances between each pair of cities, what is the shortest possible route that simultaneously visits each city only once and returns to the origin city?*

  Variations of this problem, such as the travelling purchaser and vehicle routing problem (very present in logistics and transportation problems), are vastly researched.

- **Minimum spanning tree**:
  A minimum spanning tree problem is a subset of edges from an undirected graph connecting all vertices without cycles, minimizing possible total edge weight.

  There are many algorithms and solutions for this problem. However, they lack versatility when this problem's alterations and variations are brought into question.

2.5 Classification: support-vector machines

This is an example in which the resource allocation structure within the optimisation model is not as obvious. Suppose we are given a data set $D \in \mathbb{R}^n$ with $|D| = N + M$ that can be divided into two disjunct sets $I^- = \{x_1, \ldots, x_N\}$ and $I^+ = \{x_1, \ldots, x_M\}$.

Each element in D is an observation of a given set of n features with values represented by a $x \in \mathbb{R}^n$ that has been classified as belonging to set I^- and I^+. Because of the availability of labelled data, classification is said to be ane xample of supervised learning in the field of machine learning.

Figure 3 illustrates this situation for $n = 2$, in which the orange dots represent points classified as belonging to I^- (negative observations) and the blue dots represent points classified as belonging to I^+

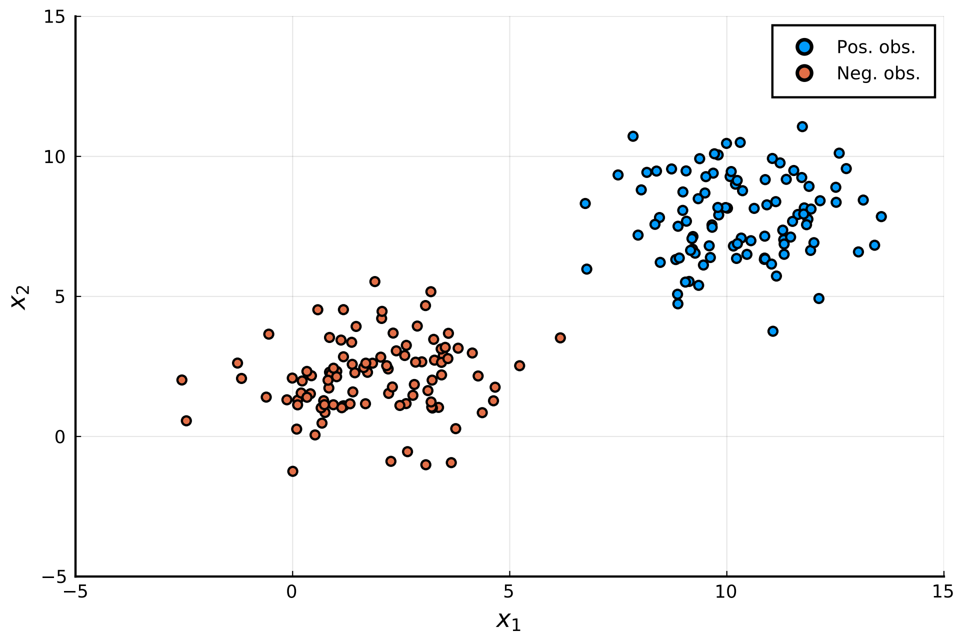

Figure 3: Two hundred observations for x_i classified to belong to I^- (orange) or I^+ (blue).

(positive observations).

Our task is to obtain a function $f : \mathbb{R}^n \mapsto \mathbb{R}$ from a given family of functions that is capable to, given an observed set of features $\hat{x}$, classify whether it belongs to I^- or I^+. In other words, we want to calibrate f such that:

$$f(x_i) < 0,\ \forall x_i \in I^-,\ \text{and}\ f(x_i) > 0,\ \forall x_i \in I^+. \tag{17}$$

This function would then act as a classifier that could be employed to any new observation $\hat{x}$ made. If f is presumed to be an affine function of the form $f(x) = a^\top x - b$, then we obtain a *linear classifier.*

Our objective is to obtain $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$ such that misclassification error is minimised. Let us define the error measure as:

$$e^-(x_i \in I^-; a, b) := \begin{cases} 0, \text{ if } a^\top x_i - b \leq 0, \\ a^\top x_i - b, \text{ if } a^\top x_i - b > 0. \end{cases}$$

$$e^+(x_i \in I^+; a, b) := \begin{cases} 0, \text{ if } a^\top x_i - b \geq 0, \\ b - a^\top x_i, \text{ if } a^\top x_i - b < 0. \end{cases}$$

Using this error measure, we can define constraints that capture deviation on each measure by means of nonnegative slack variables. Let $u_i \geq 0$ for $i = 1, \dots, N$ and $v_i \geq 0$ for $i = 1, \dots, M$ be slack variables that measure the *misclassification error* for $x_i \in I^-$ and $x_i \in I^+$, respectively.

The optimisation problem that finds optimal parameters a and b can be stated as:

$$\begin{aligned} \text{min.} \quad & \sum_{i=1}^{M} u_i + \sum_{i=1}^{N} v_i && (18) \\ \text{subject to:} \quad & a^\top x_i - b - u_i \leq 0, i = 1, \dots, M && (19) \\ & a^\top x_i - b + v_i \geq 0, i = 1, \dots, N && (20) \\ & ||a||_2 = 1 && (21) \\ & u_i \geq 0, i = 1, \dots, N && (22) \\ & v_i \geq 0, i = 1, \dots, M && (23) \\ & a \in \mathbb{R}^n, b \in \mathbb{R}. && (24) \end{aligned}$$

The objective function (18) accumulates the total misclassification error. Constraint (19) allows for capturing the misclassification error for each $x_i \in I^-$. Notice that $u_i = \max\{0, a^\top x_i - b\} = e^-(x_i \in I^-; a, b)$. Likewise, constraint (20) guarantees that $v_i = e^+(x_i \in I^+; a, b)$. To avoid trivial solutions in which $(a, b) = (0, 0)$, the normalisation constraint $||a||_2 = 1$ is imposed in constraint (21), which turns the model nonlinear.

Solving the model (18)–(24) provides optimal (a, b) which translates into the classifier represented as the green line in Figure 4.

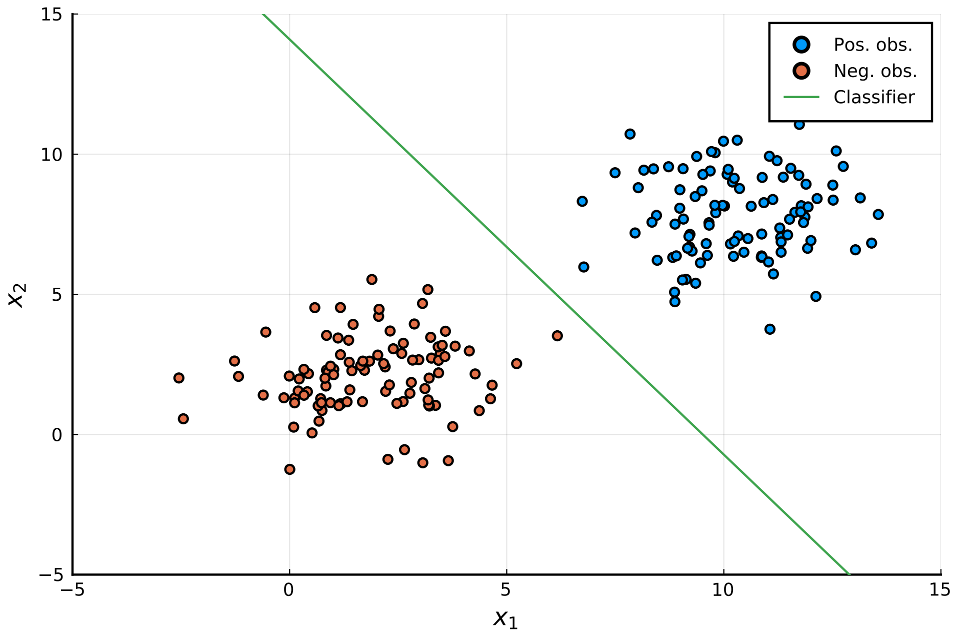

Figure 4: Two hundred observations for x_i classified to belong to I^- (orange) or I^+ (blue) with a classifier (green).

A variant referred to as *robust classifier* penalises not only the the misclassification error, but also the observations within a given slab $S = \{x \in \mathbb{R}^n \mid -1 \leq a^\top x - b \leq 1\}$. Notice that, being the two lines defined by $f^-(x) : a^\top x - b = -1$ and $f^+(x) : a^\top x - b = +1$, the distance between the two hyperplanes is given by $\frac{2}{||a||_2}$.

Accordingly, we redefine our error measures as follows.

$$e^-(x_i \in I^-; a, b) := \begin{cases} 0, \text{ if } a^\top x_i - b \leq -1, \\ |a^\top x_i - b|, \text{ if } a^\top x_i - b > -1. \end{cases}$$

$$e^+(x_i \in I^+; a, b) := \begin{cases} 0, \text{ if } a^\top x_i - b \geq 1, \\ |b - a^\top x_i|, \text{ if } a^\top x_i - b < 1. \end{cases}$$

By doing so, a penalty is applied not only to those points that were misclassified but also to those points correctly classified that happen to be inside the slab S. To define an optimal robust classifier, one must trade off the size of the slab, which is inversely proportional to $||a||$, and the total of observations that fall in the slab S. The formulation for the robust classifier then becomes:

$$\begin{aligned} \text{min.} \quad & \sum_{i=1}^{M} u_i + \sum_{i=1}^{N} v_i + \gamma ||a||_2^2 && (25) \\ \text{subject to:} \quad & a^\top x_i - b - u_i \leq -1, \ i = 1, \dots, M && (26) \\ & a^\top x_i - b + v_i \geq 1, \ i = 1, \dots, N && (27) \\ & u_i \geq 0, i = 1, \dots, N && (28) \\ & v_i \geq 0, i = 1, \dots, M && (29) \\ & a \in \mathbb{R}^n, b \in \mathbb{R}. && (30) \end{aligned}$$

In objective function (25), the errors accumulated in variables u_i, $i = 1, \dots, N$ and v_i, $i = 1, \dots, M$ and the squared norm $||a||_2^2$ are considered simultaneously. The term γ is a scalar used to impose an emphasis on minimising the norm $||a||_2$ and incentivising a larger slab S (recall that the slab is large for smaller $||a||_2$). The squared norm $||a||_2^2$ is considered instead as a means to recover differentiability, as the norm $||a||_2$ is not differentiable. Later on, we will see how beneficial it is for optimisation methods to be able to assume differentiability. Moreover, note how in constraints (26) and (27) u and v also accumulate penalties for correctly classified x_i that happen to be between the slab S, that is, that have term $a^\top x - b$ larger/ smaller than -1/ +1. Figure 5 shows a robust classifier an arbitrary value of γ.

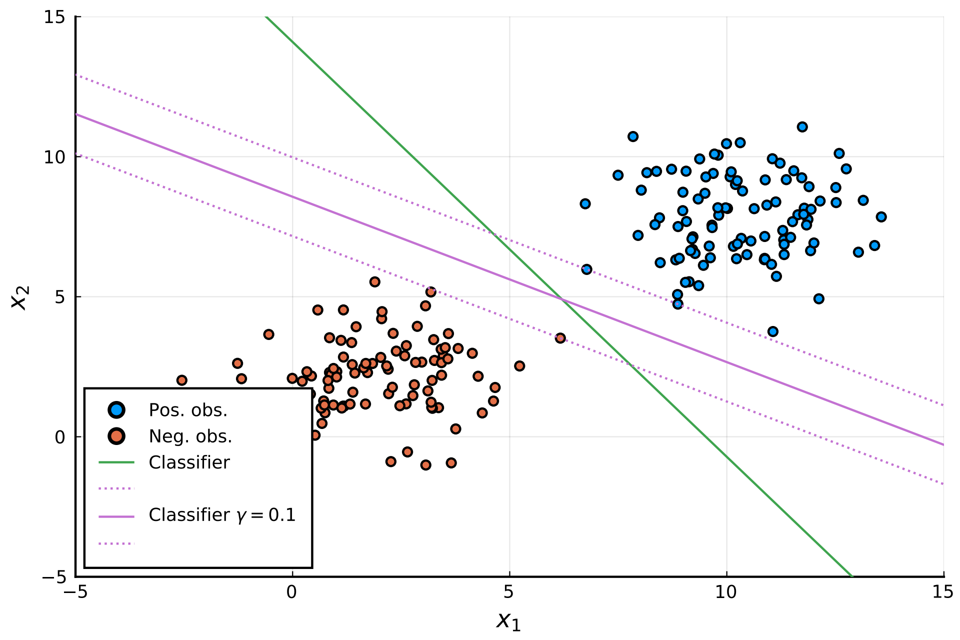

Figure 5: Two hundred observations for x_i classified to belong to I^- (orange) or I^+ (blue).

Remark: robust classifiers are known in the machine learning literature as *support vector machines*, where the support vectors are the observations that support the slab.