

**mr**  
manual de referencia

ECONOMÍA Y ORGANIZACIÓN DE EMPRESAS

# NOTEBOOKS FOR OPERATIONS RESEARCH

## A PRACTICAL GUIDE TO OPERATIONS RESEARCH WITH PYTHON

Francisco Fraile Gil



edUPV

Universitat Politècnica de València

ECONOMÍA Y

DIRECCIÓN DE EMPRESAS

Francisco Fraile

# **Notebooks for Operations Research**

A practical guide to operations  
research with Python



**edUPV**

---

Universitat Politècnica de València



Collection *Manual de Referencia*, Series Economics and Business Organization  
[http://tiny.cc/edUPV\\_MR](http://tiny.cc/edUPV_MR)

To cite this publication please use the following reference:

Fraile Gil, Francisco (2024). *Notebooks for Operations Research. A practical guide to operations research with Python*. Valencia: edUPV. <https://doi.org/10.4995/MR.2024.677801>

The contents of this publication have been evaluated by a double –blind system as can be consulted on the following page [http://bit.ly/Evaluacion\\_Obras](http://bit.ly/Evaluacion_Obras)

Autorship  
Francisco Fraile Gil

Publisher  
2024, edUPV (Editorial Universitat Politècnica de València)  
[www.lalibreria.upv.es](http://www.lalibreria.upv.es) / Ref.: 6778\_01\_01\_01

© of the texts and images: the authors

ISBN: 978-84-1396-253-5  
DOI: <https://doi.org/10.4995/MR.2024.677801>

If the reader detects a mistake in the book or wishes to contact the authors, he can send an email to [edicion@editorial.upv.es](mailto:edicion@editorial.upv.es)



Notebooks for Operations research, A practical guide to operations research with Python / edUPV

The reuse of the contents is allowed through the copying, distribution, exhibition and representation of the work, as well as the generation of derivative works as long as the authorship is acknowledged and it is cited with complete bibliographic information. Commercial use is not permitted and derivative works must be distributed under the same license as the original work.

## **AUTHOR**

Francisco Fraile holds a PhD in Telecommunications from the Polytechnic University of Valencia, with a background in research, innovation and teaching. He currently works at the Centre for Research in Production Management and Engineering (CIGIP), where he leads projects in digital platforms for manufacturing and logistics, integration of artificial intelligence and robotics, and zero defects manufacturing. He has coordinated and participated in R&D&I projects, in European, national and regional calls, standing out in the transfer of sustainable technologies to industry. Associate professor at EDEM, he is dedicated to training future engineers in operations research and programming. His work combines applied research and education in automated production systems engineering, pursuing a strong social and technological impact.

## **SUMMARY**

Notebooks for Operations Research is a practical guide that uses Python and Jupyter Notebooks to teach the principles and applications of operations research. Designed for both beginning students and professionals, the text connects theory and practice through interactive tutorials, solved exercises, and the use of Python libraries to solve problems with a hands-on approach.

Chapters cover topics such as linear programming, combinatorial optimisation, non-linear programming, decision and game theory, simulation and Markov chains. Each section integrates mathematical concepts with real-world applications, demonstrating how to solve optimisation and decision-making problems in complex contexts, especially in supply chain management.

In addition, the book highlights the relationship between operations research and artificial intelligence, focusing on mathematical reasoning and optimisation techniques, and provides an introduction to machine learning. It is an interactive book in the sense that all content is available in an interactive online environment that allows experimentation with the code. It provides fundamentals, tools and open source resources for those seeking to master operations research in a practical and modern approach.

# CONTENTS

<b>1</b>	<b>Introduction: Python Notebooks for Operations Research</b>	<b>3</b>
1.1	Operations Research Introduction	3
1.2	Management Framework: Supply chain models	4
1.3	Operations Research and Artificial Intelligence	6
1.4	Brief introduction to Jupyter Notebooks	10
1.5	Introduction to Python and Markdown	11
<b>2</b>	<b>Continuous Linear Programming</b>	<b>43</b>
2.1	Tutorials	43
2.2	Exercises	81
2.3	Solved Exercises	100
2.4	Python libraries	133
<b>3</b>	<b>Combinatorial Optimization, Mixed Integer Programming and Network Theory</b>	<b>147</b>
3.1	Tutorials	147
3.2	Exercises	203
3.3	Solved Exercises	212
3.4	Python libraries	256
<b>4</b>	<b>Non-Linear Programming</b>	<b>267</b>
4.1	Tutorials	267
4.2	Exercises	296
4.3	Solved Exercises	300
<b>5</b>	<b>Decision Theory and Game Theory</b>	<b>317</b>
5.1	Tutorials	317
5.2	Decision Theory Exercises	341
5.3	Decision Theory Solved Exercises	344
5.4	Game Theory Exercises	363
5.5	Game Theory Solved Exercises	367
<b>6</b>	<b>Simulation and Markov Chains</b>	<b>379</b>
6.1	Tutorials	379
6.2	MonteCarlo Method Exercises	383
6.3	MonteCarlo Method Solved Exercises	385
6.4	Markov Chain Exercises	394
6.5	Markov Chain Solved Exercises	397
<b>7</b>	<b>Bibliography</b>	<b>407</b>
<b>8</b>	<b>Call for Contributions</b>	<b>409</b>



## **Preamble**

Welcome to the *Notebooks for Operations Research* interactive book! This book contains a collection of [Jupyter Notebooks](#) with tutorials, exercises and Python libraries designed to explain the principles and applications of operations research.

Our primary goal and motivation is to bridge the gap between theoretical background and practical applications, leveraging the interactive features of Jupyter Notebook so that learners can engage with the content. Operations research is a fascinating field with countless real life applications, and we believe that, just as with many other fields, the best way to learn is to incorporate practical applications and skills. In this sense, Python provides a comprehensive toolset to solve operations research problems, and a user-friendly programming language that is easy to learn and close to mathematical language, thus it is perfect match for our objectives!

Before you embark in this educational project, you need to know that the interactive book has been carefully crafted to introduce operations research to novice students. The chapters are organised to guide readers from the fundamentals of operations research to the application of complex problem solving methods and techniques in real life scenarios using Python. The book primarily covers applications related to engineering and management, as its origins root back to the a course on Operations Research at [EDEM](#), although it is open for the entire community. It is also important to note that this interactive book is a living document, open for contributions and continuously evolving to adding new content and advancements in the field.

We happily invite you to join us and learn the fundamentals, challenge yourself with exercises, and learn the Python libraries to apply operations research in the real world. We have tried to make theory more easy to reach through the interaction with code. Additionally, the interactive notebooks incorporate questions for analysis and template prompts so that you can delve into its contents with the help of an AI assistant. We want to provide an integrated environment with all the tools you need to learn operations research with a practical mindset, at your own path.





# INTRODUCTION: PYTHON NOTEBOOKS FOR OPERATIONS RESEARCH

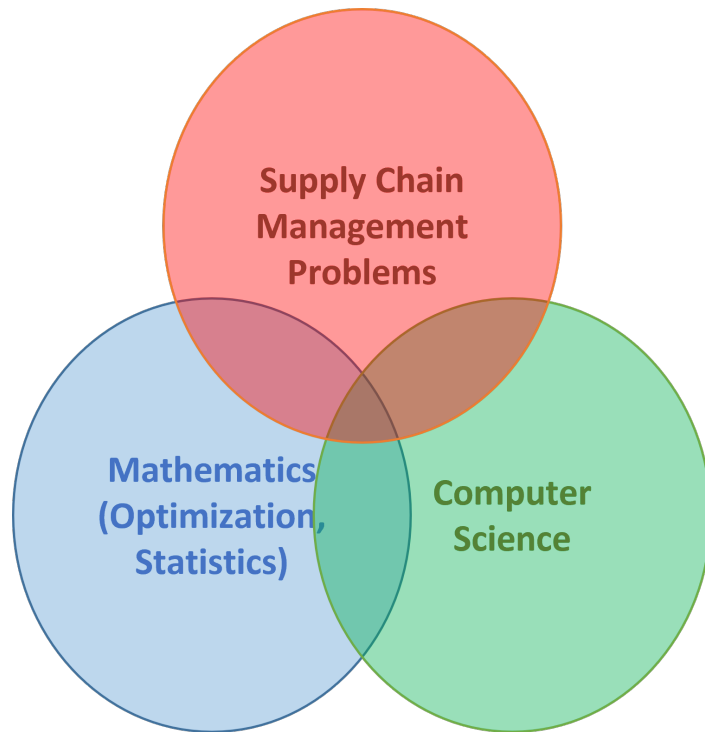
This chapter contains an introduction to Operations Research, and also covers the fundamentals of Python programming and Jupyter Notebooks used in the course, including basic tutorials and usage tips for two important libraries: Numpy and Matplotlib.

## 1.1 Operations Research Introduction

### 1.1.1 Motivation

The main objective of operations research is to support decision making through methods and techniques based on the scientific method and mathematics. Formally, making a decision is the process of selecting the best alternative between a set of alternative options. Depending on the problem, there can be a finite set or an infinite set of alternative options. The connection between Operations Research and management is clear: Managers make decisions and in extension, decision making is strongly connected to management. From this perspective, Operations Research gathers a set of tools that identify the best (optimal) alternative in a systematic way, based on a set of assumptions that allows us to model the decision making process using the mathematical language.

In a nutshell, the objective of OR is to find optimal solutions to **management problems**, optimizing the performance of **complex systems**. In practice, the optimization is achieved using **mathematical methods** and these methods are implemented using computers and **computer science**. Although operations research can be applied to different domains, this book is focused on **supply chain management**. As illustrated in the figure below, in this book, supply chain management, applied mathematics, and computer science overlap and interact in a number of ways.



Supply chain management is the process of coordinating the flow of goods, services, and information from suppliers, through production and distribution, to customers. It involves managing the relationships between different parties in the supply chain, as well as coordinating the activities and resources needed to move goods and services from one place to another. Some of the key activities involved in supply chain management include sourcing raw materials and components, manufacturing and assembling products, warehousing and storing finished goods, and transportation and distribution. The goal of supply chain management is to ensure that goods and services are produced and delivered efficiently and effectively, while minimizing costs and risks. Operations research and computer science can both play important roles in the design and optimization of supply chain systems.

Supply chain models provide a framework for understanding and analyzing the various components and processes involved in the supply chain, and operations research techniques (a branch of applied mathematics) can be used to design and analyze algorithms for optimizing the performance of these systems. Once the algorithms have been designed and tested, computer science can be used to implement the solution, either by building a custom software application or by using a pre-existing tool or platform.

The next section describes the management framework used in the book, consisted of **management levels**, and **business processes** modelling supply chain management.

## 1.2 Management Framework: Supply chain models

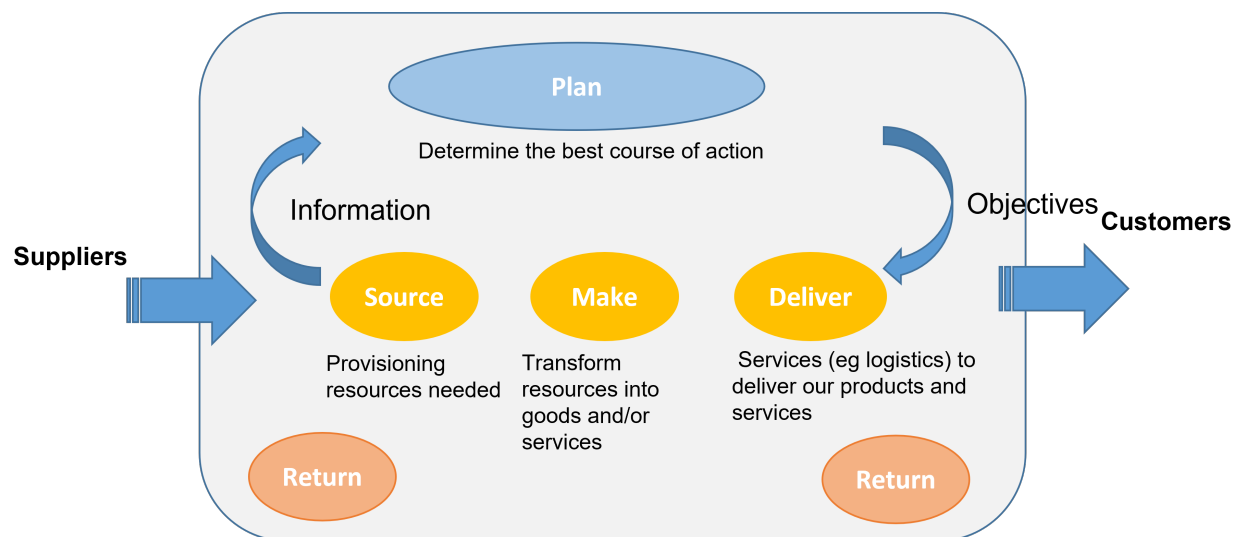
### 1.2.1 Business processes

Business processes represent the group of activities needed to produce goods or services. From a very fundamental point of view, any company produces goods or service and delivers these good or services to customers. Likewise, the company has to purchase the goods or services that are needed to produce and deliver their offer from a set of suppliers. Obviously, every organisation needs to make good decisions in these processes to remain to be competitive. Normally, companies are organised into different business units or departments. Some examples are the sourcing department, in charge of purchases, the engineering department in charge of product design and production, sales, financing, marketing, human resources. Managing these departments means making decisions, and these decisions must be aligned to meet the objectives of the company.

The Supply Chain Operations Reference model (SCOR) provides a model for the management processes involved in a company, the Supply Chain Reference Model, In the SCOR model, we identify six management processes:

- **Plan:** The first is the Plan, which groups all processes that determines the best course of action to meet the requirements of the rest of the processes.
- **Source:** The second group are source processes, which are in charge of provisioning resources needed to meet the demand for our products.
- **Make:** The third are Make processes, which transform the goods and services of the suppliers into the good or services that meet the demand of our customers.
- **Deliver:** Then, we have the deliver processes which imply the needed transportation, and logistics necessary to deliver our products and services.
- **Return:** The return processes deal with the returning of products or services from customers because they do not meet the requirements or the level of service that was agreed, but also the return of good or services to our suppliers for the same reason.
- **Enable:** And finally, enable processes which are associated with the management of the supply chain, that is the management of the relationships with our collaborators, both customers and suppliers.

Operations Research provides tools to support decision making in these management processes. All the problems included in this interactive book can be mapped to the processes in the SCOR model.



## 1.2.2 Management levels

Management levels divide management and therefore decision making into three planes that are traversal to the entire company: The strategic level, the tactical level and the operational level.

Each level has its own responsibilities and all work towards the same goals, to achieve the same objectives. The main differences between levels is that they deal with different time scales.

- **Strategic level:** The strategic level deals with long term business decisions and assurance of business objectives in the future. Therefore, decisions at this level deal with long time periods or time horizons, in the range of several years. Consequently, they need to be revisited or updated with a periodicity in the range of months.
- **Tactical level:** The tactical level on the other hand deals with business decisions to ensure the fulfillment of the objectives in the near future, for instance, in the next 1 or two years. Tactical level decisions need to be revisited every few months to ensure mid term performance. Some examples are for instance client acquisition or investment selection.

- **Operational level:** Finally, the operational level deals with intermediate-term decisions in processes, rather than business decisions. Operational management controls how the different processes, or the different units of an organisation implement the company's strategic and tactical objectives. The operational level articulate specific processes and spans management decisions in smaller periods of time, from months to minutes. Consequently, they need to be revisited rather frequently.



Operations Research provides tools to support decision making in any of the three levels, although the O in OR clearly refers to the operational level.

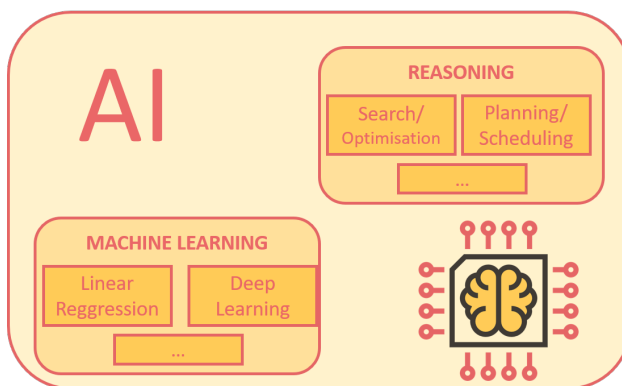
### 1.3 Operations Research and Artificial Intelligence

**Operations Research is strongly connected to Artificial Intelligence (AI). The European Union defines AI as:**

"Artificial intelligence (AI) refers to systems that display intelligent behaviour by analysing their environment and taking actions - with some degree of autonomy - to achieve specific goals"

Therefore, AI systems sense their environment and take optimal decisions to achieve specific goals. Clearly, from this definition, AI and OR are also interlinked and can be used together to solve complex optimization problems. Fundamentally, both disciplines make use of mathematics, primarily in the field of optimisation, to find the best course of action. For this reason, and given the importance and impact of AI in solving decision problems, this book will also cover some of the basics of modern artificial intelligence.

Moreover, as described in the same document referenced above, broadly speaking, all such artificial intelligence techniques can be grouped in two main groups, related to the capabilities of **reasoning** and **learning**.



- **Reasoning:** These techniques allow an AI system to take as input the data coming from different sources and propose an action to take, given the goal to achieve.
- **Machine learning:** These techniques allow an AI system to learn how to solve problems that cannot be precisely specified, or whose solution method cannot be described by symbolic reasoning rules.

Operations research techniques, such as mathematical modeling and optimization algorithms, can be used to represent and solve reasoning problems in a way that is amenable to automation and solution by an AI system. In turn, machine learning can be used to improve the performance and efficiency of operations research models and algorithms. For example, an AI system might be used to analyze large amounts of data and identify patterns and relationships that are not immediately apparent to humans. This analysis can then be used to inform the development of more accurate and effective operations research models, which can be used to optimize the performance of a system. Similarly, an AI system might be used to automate the solution of operations research models, allowing for faster and more accurate decision making. Overall, the combination of AI and operations research can be a powerful tool for solving complex optimization problems in a wide range of fields.

Let us emphasize again that this book in OR is mainly focused on reasoning techniques, and not so on learning techniques (although it will cover the basics). Reasoning techniques will allow us to find the best decision from a set of alternative options. The course will discuss extensively **optimisation** which is an entire field of mathematics dedicated to finding the **optimal** (normally the minimum or the maximum) solution among all **feasible** solutions. Due to the importance of these definitions, they are formally defined in a section below. The course will also cover **searching problems** and **Combinatorial Optimization**, which is the sub-field of optimisation dedicated to finding an optimal solution from a finite set of solutions. The course will describe how complex problems in OR, like **planning** (eg finding the optimal allocation of resources to accomplish a specific objective) or **scheduling** (find the optimal order of activities that minimise the execution of an activity plan, given a limited set of resources), can be modeled and solved using advanced analytic techniques.

Although it is not a fundamental part of this course, the course will also describe how machine learning is strongly related to the same techniques and will discuss specific applications of machine learning in OR.

### 1.3.1 Reasoning

The reasoning capability can be further divided into four basic steps needed to be able to choose the best action (make a rational decision) to achieve one objective:

- **Problem modelling:** Obtain a mathematical representation of the decision making problem.
- **Input data collection:** Transform the available data into a format that can be input to the problem model
- **Reason with the input data:** (eg find the optimal solution among all possible alternatives)
- **Make a decision:** based on the provided result (feedback)

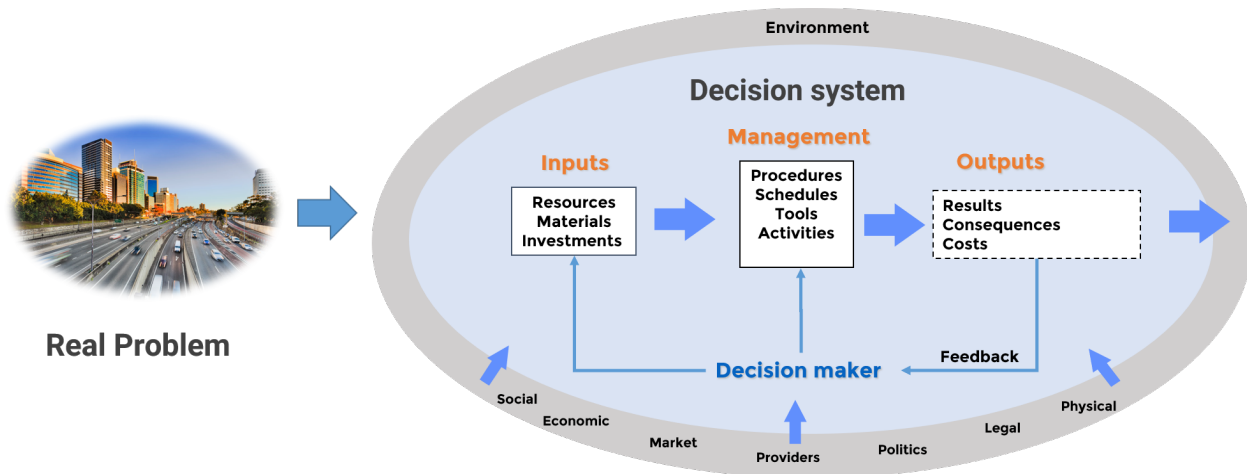
### 1.3.2 Problem modelling

The first part of the problem modelling process is to provide a clear definition of the objectives and the scope of management. In other words, the first question that one needs to answer is:

What are the goals? And, what is the relationship between these goals and the decision or decisions under analysis?

The answer to these questions will be in the form of a set of variables, and mathematical expressions between these variables that will express the decision problem as a mathematical problem.

System theory is useful at this point to provide a clear framework to illustrate problem modelling.



The main components of the decision system are:

- **Inputs:** Representation of the (limited) resources managed by the decision maker. Management problems deal with scarce or limited resources that need to be managed efficiently.
- **Management:** Represent the decisions taken on **controllable decision variables**. That is, there will be a set of variables that quantify the management of the input resources and the values these variables can take represent the different decision alternatives available to the decision maker.
- **Environment:** There are also **uncontrollable variables**, meaning variables that have an effect in the decision making, but are determined by the environment, and therefore, they are not under the control of the decision maker.
- **Output:** Represent the results of the decision. Each different alternative will have a different effectiveness, ie will achieve the objective at a certain degree. The optimal solution is the most effective alternative. Measures of effectiveness are metrics used to determine effectiveness of the output and normally the optimal solution will be the maximum possible value (eg maximise profits) or the minimum possible value (eg minimise cost) of such metrics.

And depending on the model chosen, there will be different types of interactions between these model components. At this point, it is possible to classify models into two categories:

- **Normative models:** Normative models are designed to select the best alternative, by deriving a set of mathematical expressions that characterise the relationship between model components (output, input, decision variables, uncontrollable variables). Depending on the type of model, we will be able to use different techniques to find the optimal solution. Some examples are Linear or non-linear mathematical programming.
- **Descriptive models:** On the other hand, descriptive models are designed to describe the system behaviour. They allow us to experiment with the system and explore the effectiveness of the different alternatives, thus gaining in-depth knowledge of the problem that can be actioned in the decision making process. The most representative example of descriptive models is simulation.

The main model components are:

- **Objective function:** Mathematical relationship between result variables and decision variables. The objective function models the effectiveness of decisions.
- **Constraints:** Mathematical relationships between decision variables and uncontrollable variables. The constraints model other effects of the environment that are not accounted for in the objective function.

### 1.3.3 Data collection

Data collection is the process that determines the values of the (technological) coefficients and uncontrollable variables in our model. In other words, data collection is the process that determines the value of the parameters of our model. Before data collection, our model is an abstract model that represents *any* decision problem instance it has been designed for. For instance, we can derive an abstract model to determine the optimal investments subject to a set of constraints that model the market. But if we want to use this model to determine the optimal investments for our company, we need to input into this model our company data and give values to its parameters.

In this sense, there are different types of models depending on the nature of their parameters:

- **Deterministic models:** All model parameters are known with certainty
- **Stochastic models:** All model parameters are stochastic variables (ie there is some degree of uncertainty in their value)
- **Mixed models:** Models that combine deterministic and stochastic parameters

### 1.3.4 Reasoning

Finally, once we have a specific instance of a model with our collected data, we are able to apply a technique to find the solution to the problem.

#### Types of solutions

There are however different types of solutions that we will be able to find:

- **Feasible solution:** It meets all the requirements and constraints set by the environment, otherwise the solution is **unfeasible**
- **Optimal solution:** It provides the best result of all feasible solutions. Normally, our objective is to find the optimal solution, but depending on the complexity of our problem, at times this might just not be possible
- **Sub-optimal solution:** It provides a bounded feasible solution, that is, a solution that is at a known maximum distance to the optimal solution.

#### Types of techniques

Depending on the problem model, we will have different mathematical techniques to find a solution to our problem. These techniques can be classified into the following categories:

- **Numerical Techniques:** Numerical techniques test the effectiveness of the different alternatives to find the optimal solution. Depending on the strategy used to explore the solution space, numerical techniques can be classified into the following categories:
  - **Enumeration (Brute force):** This is the simplest technique: Test ALL possible solutions.
  - **Algorithms:** Step-by-step procedure to find optimal solution
  - **Heuristic:** Step-by-step procedure to find a sub-optimal solution with a limited number of operations
  - **Simulation:** Test alternatives in a computer representation of the system
  - **Machine learning:** Configure a mathematical model to find a solution based on previous known results
- **Analytical Techniques** Analytical techniques provide a one-step method to find the solution. They can be further classified into:
  - **Optimal:** One step method to determine the optimal solution
  - **Sub-Optimal:** One step method to find sub-optimal solutions

Depending on the complexity of the problem, or the type of model, there will be different techniques available. If there are no techniques that can be applied, then the problem is not workable and we need to find an alternative model. In other words, workability is a property of our model that determines whether it is possible to solve the problem with available techniques or not.

### 1.3.5 Making a decision

Finally, through a technique, we are able to obtain a solution that represents the best alternative of the decision problem. But, before we are able to implement this decision into the real world, it is important to validate that the model exhibits some fundamental properties:

- **Consistency:** verify that the model is valid when parameters are changed to the extreme
- **Sensitivity Analysis:** How is the response affected by changes in the parameters?
- **Plausibility:** Is the model consistent with available data?

If we are satisfied with the answer to these questions, we are ready to implement the best alternative in the real world and monitor the results. This implementation can be automatic (taken by an autonomous system) or supervised (with human supervision).

## 1.4 Brief introduction to Jupyter Notebooks

Jupyter Notebooks are great to share software and scientific results. In my opinion, the main advantages are:

1. **Explainability:** Jupyter Notebooks combine software code together with rich multimedia descriptions. This allows researchers to **explain** their results, show exactly the code they have used, describe its fundamentals and its results. This narrative format helps the reader understand and in extension reuse scientific results.
2. **Composability:** You can easily combine different Jupyter Notebooks into larger project.
3. **Community Support:** There is a very large user community worldwide supporting Jupyter Notebooks.

Also, there are a growing number of platforms and tools that support it off-the-shelf. Some examples that we will use during the course are: [Anaconda](#), [Google Colabs](#), [JupyterLab](#), or [Pycharm](#).

From a very basic point of view, a Jupyter Notebook is a file that contains a sequence of **cells**. There are primarily two types of cells:

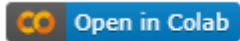
- **Code Cells:** Code cells contain code in a given programming language (for instance Python) that will be executed by an **interpreter** when we execute the Notebook. Code cells have an output section where we can see the result of the code cell. From this, there are two ways we can use a Notebook. We can *view* a Notebook using a viewer, which allows us to render the Notebook, but not to run the code cells. If we have an Interpreter at hand, we can also **run** the Notebook to let the interpreter run the code and update the results of code cells.
- **Markdown Cells:** Markdown cells use [Markdown Syntax](#) to explain code cells using a rich language that allows us to include links, equations, images, any kind of resource we can think of to support our presentation.



## 1.5 Introduction to Python and Markdown

### 1.5.1 Introduction to Python

Try me



Python is a high level programming language that is particularly suited for data science and operations research. Python has three main versions labelled 1, 2, and 3. Within each version, there are several minor releases, like 2.2 and 3.3. Therefore, there are many Python versions labelled 1.x, 2.x, 3.x, where the x is the minor version number. This course covers 3.x, but it is not difficult to change from one version to another. This notebook provides a quick introduction to Python in Notebooks. If you are not familiar with this environment, check the partner book on computer programming in this [link](#)

#### Your first code cell

Let us start easy with our first code block. In this cell, we will **assign variables** and use a **function**. We will use the Python function `print()` that prints a message in the output of the code cell. The message is passed as a **string** variable. Let us introduce briefly these two concepts in the context of Jupyter Notebooks.

#### Basic variables

Variables allow us to store a *value* and use it in our Notebook through a *name*. To assign value to a variable, we first type the name, then the equal sign (=) and then the value:

```
[1]: x=1
      y=0.4
      msg='Hello world'
```

There are different types of variables. In the example above, *x* and *y* are two numeric variables. We will talk about numeric variables later in this Notebook. The variable *msg* is a string variable (text). You can either use quotes or double quotes to assign a string variable.

#### Basic functions

In Python, a function takes variables as arguments and return a value. The standard Python library provides many convenient functions. Let us start with a simple example.

```
[2]: # This is your first Python cell
      print(msg)
      print('Hello again!')

      Hello world
      Hello again!
```

The first line in the example above is a comment. Comments help us humans read the code and start with a '#'. Therefore, the code cell above has only one comment and two lines of code. The first line of code calls the function `print` with the string variable *msg*. Note that we can use variables in separate code cells of a Notebook. Note also that variables are declared when we Run a code cell. Cells should be run in the order they are presented in a Notebook. The second line of code takes a string value as argument, because we do not want to store this value to use it later. You can play around with the `print` function to print a different message or several messages, just give it a try! Do not be afraid to change the values of variables. Variables are mutable, meaning that you can change the value of a variable in your code. But before

you do, note that Python documentation normally includes references to the group comedy Monty Python and the words “spam” and “eggs” are normally used as fill-in variables. The example below reassigns the value of the variable `msg` to print the word “spam”. You can try to add a new line of code to print “spam” and “and eggs!” in two separate lines of the output in the next cell code.

```
[3]: msg="spam"
print(msg) #Tip: I can also add a comment to a line of code
# Add a new line to print "spam" and "and eggs!" to the output of this code cell

spam
```

### Numeric Variables

In Python, you can declare variables and do simple operations with them. We are going to do mathematical operations with variables, so you need to get familiar with the operands and the results, depending on the type of arguments used. Let us start with numerical variables. Python uses standard variable notations, so the operands `+`, `-`, `*`, `/` and `**` represent sum, rest, multiply, divide and power operations. You can also use parenthesis to control the order of operations and declare negative or decimal variables using the `-` sign or a `.` (eg `-4` or `-4.5`). In the following block of code you can find some examples.

```
[29]: x = 2 #x is a numeric variable, equal to 2.
y = x+4 #y is a numeric variable, equal to x+4 = 6.
z = (x+3)*(y**2)/4 #z is a numeric variable equal to (2+3)*(6**2)/4 = 5
print(x)
print(y)
print(z)

2
6
45.0
```

Notice that `z` is different from `x` and `y`. While `x` and `y` are integers, `z` is a floating variable. One of the main characteristics of Python is that it is a **weakly typed** programming language. Roughly, this means that you can do operations with variables of different types and Python will make the operation to its best effort to avoid a compilation error. That is why Python generated a floating variable from the division operation. Also, as in math, the order of the operands is important, and we can use parenthesis to control it.

### Errors

We cannot use any combination of variables. For instance, adding a numeric variable to a string could result in a compilation error. Another important piece of information is that in Jupyter, you can use variables declared in different code cells as long as you run them first. If you have run the previous cell, you can test the following cell.

```
[7]: ms = 'x is ' + x
print(ms)

-----
TypeError                                Traceback (most recent call last)
<ipython-input-7-3ada229f534a> in <module>()
----> 1 ms = 'x is ' + x
      2 print(ms)

TypeError: can only concatenate str (not "int") to str
```

Notice that the output of the cell now shows an error message. Do not panic. The message gives you information to interpret and correct the error. While you program, specially when you are a beginner, it is normal to get errors and very important to get familiar with the error messages. In this message, Python warns you of a `TypeError` and the reason why (you cannot concatenate an `int` to a string). There are other things that you are not allowed. For instance, you cannot

divide by zero. The only person that can divide by zero is Chuck Norris. If you are not Chuck Norris, you will get a different error if you try the following cell:

```
[8]: z = y/0

-----
ZeroDivisionError                                Traceback (most recent call last)
<ipython-input-8-d81c8fc02de0> in <module>()
----> 1 z = y/0

ZeroDivisionError: division by zero
```

In this case, the error message is different. In both cases, you get enough context information to identify the line of code that gave error and the reason why that line resulted in an error. ## Printing results to output Now some tips, if you want to print a msg with values, you can pass more variables to the print function as comma separated values. As we saw in the previous example, the print function can also take numeric values, so this is a way to get the expected result. We could also use the function str to convert a number to a string and use the + operator to concatenate the strings on a message. We can also use the format function to a string with two parameters to tell the function print how do you want to see them:

```
[14]: print('x is ', x , ', y is ', y, ', z is ', z)
print('x is ' + str(x) + ', y is ' + str(y) + ', z is ' + str(z))
print('x is {x:d}, y is {y:d} and z is {z:.1f}'.format(x=x,y=y,z=z))

x is 2 , y is 6 , z is 5.0
x is 2, y is 6, z is 5.0
x is 2, y is 6 and z is 5.0
```

In Python, there are many ways to achieve the same result, so it is very important to read the docs and experiment to understand the different trade-offs!.

## Lists

We will work quite a lot with lists in this course. Lists are used to store an indexed list of objects. To assign a list variable, we use brackets, surrounding the values of the objects we want to store. Indexed means that we can get the value of a specific item in the list using its position in the list. The index is specified again using brackets. Let us see a basic example:

```
[5]: words = ["Hello!", "spam", "eggs"]
print(words[0])
print(words[1])
print(words[2])

Hello!
spam
eggs
```

## Boolean variables

Another important type in Python is the Boolean type. Boolean variables can take two values, False or True.

```
[8]: b = True
```

### Boolean logic

Normally, in Python we used logical operands to do operations with boolean variables and build logic into our code. Basic boolean operators are *logic and* (and), *logic or* (or) and *negate* (not):

```
[15]: c = not b
print(c)
print(c & b)
print(c and b)
print(c | b)
print(c or b)
```

```
False
False
False
True
True
```

Comparisons are also important operators that return boolean values. Comparators are less (<), less or equal (<=), exactly equal (==), greater or equal (>=), and greater (>). Normally, they take numerical values:

```
[21]: x = 5
y = 6
z = 6
print(x < y)
print(y <= z)
print(y == z)
print(y >= z)
print(y > x)
```

```
True
True
True
True
True
```

The *in* operator is important in the context of lists. It allows us to check if any element of a list contains a specific value. It can be combined with the *not* operator if we want the opposite logic.

```
[25]: print("tomato" in words)
print("spam" in words)
print("tomato" not in words)
```

```
False
True
True
```

## Control Structures

Control structures allow us to implement complex logic into our programs, creating different execution pathways or repeating code blocks to make our code more efficient. The next subsections describe basic selection structures (`if` `else` clauses) and repetition structures (`for` loops) in Python.

### If and else clauses

If clauses allow us to execute some code if a condition is true. This is the notation of the if clause:

```
if expression:
    sentence(s)
```

The *sentences* inside the if clause are only executed if the *expression* is true. On the other hand, else clauses execute sentences if the previous clause does not terminate normally. In general, we will use else clauses after if clauses to execute other clauses when the expression is not true. Therefore, an if clause combined with an else clause looks something like this:

```
if expression:
    sentence(s)
else:
    other_sentence(s)
```

Let us see an example:

```
[20]: if b:
        print("b is:")
        print(c)
    else:
        print("False!")

    print("This message is always shown")

b is:
False
This message is always shown
```

Note that there is an *indent* (a space from the left margin) in the sentences inside the if and else clauses. This is very important because it is the way we have to tell Python which lines of codes are inside a clause and which are not.

### For loops

*For loops* iterate over iterable variables like lists (we will see other iterable types later in the course). *For loops* allow us to declare a variable and use it inside it to control the sentences it contains. For loops are primarily used to write very efficient code with few lines. The structure of a for loop is

```
for var in vars:
    sentence(s)
```

The for loop declares a (*var*) variable that can only be used inside the sentences it contains. It will run the *sentence(s)* it contains as many times as elements are contained in the (*vars*) list used in the clause.

Take as example the code we used when we defined lists. We could rewrite it as:

```
[22]: for word in words:
        print(word)
```

```
Hello!  
spam  
eggs
```

### List comprehension

One of the main features of Python lists we will use in the course is list comprehension. List comprehension allows us to assign list variables very efficiently using for loops inside a bracket. Let us see how it works by example:

```
[32]: base = [0, 1, 2, 3]  
squares = [i**2 for i in base]  
print(squares)  
variable_names = ['x_'+str(i) for i in base]  
print(variable_names)
```

```
[0, 1, 4, 9]  
['x_0', 'x_1', 'x_2', 'x_3']
```

### Libraries

In Python, we can import libraries using the keyword `import`. Libraries encapsulate functions and declare variables that provide additional functionalities and type declarations to provide a specific functionality. In next tutorials, we are going to introduce two libraries, `numpy` and `matplotlib`, but before that, you need to know how to import a library. Imports work like this:

```
import library as lib
```

in the code snippet above, we use the keyword `import` to ask python to import the *library* and use the alias *lib*. Later on our code, we will be able to use the functions in the library using the alias. Python and Conda allow you to import many useful libraries through import commands, but not every available library. For some libraries, first we will need to install the library in our environment, normally using Conda (but we could also use other tools like `pip`). This is however everything you need to know for now. You can now check the following tutorials to learn more about **`numpy`** and **`matplotlib`**.

### 1.5.2 The NumPy Library

#### Try me



The `Numpy` (Numerical Python) is a package of numerical functions to effectively work with multidimensional data structures in Python. In Python, it is possible to work with nested lists to work with multidimensional structures (arrays and matrix), but this is not efficient. The `Numpy` library defines the `numpy` array object to provide an efficient and convenient object to define multidimensional structures.

To use `Numpy` in your Notebooks and programs, you first need to import the package (in this example we use the alias `np`):

```
[3]: import numpy as np
```

## The Numpy Array

The numpy array uses a similar structure to a Python list, although as mentioned above, it provides additional functionalities to easily create and manipulate multidimensional data structures. The data in an array are called elements, and they are accessed using brackets, just as with Python lists. The dimensions of a numpy array are called **axes**. The elements within an axis are separated using commas and surrounded by brackets. Axes are also separated by brackets, so that a numpy array is represented as a nested python list. The **rank** is the number of axis of an array. The **shape** is a list representing the number of elements in each axis. The elements of a numpy array can be of any numerical type.

```
[6]: b = np.array([[1,2,3,4],[5,6,7,8]]) #This creates a 2-dimensional (rank 2) 2x4 array
print("My first Numpy array:")
print(b)

print("element in position (1,2) is:")
print(b[1,2])

print("Number of dimensions:")
print(b.ndim) #number of dimensions or rank

print("Shape of array:")
print(b.shape) #shape (eg n rows, m columns)

print("Total number of elements:")
print(b.size) #number of elements
```

```
My first Numpy array:
[[1 2 3 4]
 [5 6 7 8]]
element in position (1,2) is:
7
Number of dimensions:
2
Shape of array:
(2, 4)
Total number of elements:
8
```

## Create Numpy Arrays

Numpy includes several functions for creating numpy arrays initialized with convenient ranks, shapes, or elements with constant or random values.

### Some examples:

```
[8]: o = np.ones((3,2)) # array of 3x2 1s
print(o)

b=np.zeros((3,4)) # array of 3x4 zeroes
print(b)

c=np.random.random(3) #array of 3x1 random numbers
print(c)

d=np.full((2,2),12) # array of 2x2 12s
print(d)

id =np.eye(3,3) # identity array of size 3x3
```

(continues on next page)

(continued from previous page)

```
print(id)

[[1. 1.]
 [1. 1.]
 [1. 1.]]
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
[0.71574091 0.54968971 0.72723399]
[[12 12]
 [12 12]]
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

### Creating sequences

Some useful functions for creating lists are **arange** and **linspace**:

- **arange(start, end, step)**: creates a numpy array with elements ranging from **start** to **end** incrementing by **step**. Only end is required, using only end will create an evenly spaced range from 0 to end.
- **linspace(start,end,numvalues)**: creates a numpy array with **numvalues** elements with evenly distributed values ranging from **start** to **end**. The increment is calculated by the function so that the resulting number of elements matches the numvalues input parameter.

```
[10]: a = np.arange(0, 10, 2)
print(a)

b=np.linspace(0,10,6)
print(b)

[0 2 4 6 8]
[ 0.  2.  4.  6.  8. 10.]
```

### Arithmetic operations

You can apply element-wise **arithmetic** and **logical** calculations to numpy arrays using arithmetic or logical operators. The functions **np.exp()**, **np.sqrt()**, or **np.log()** are other examples of functions that operate in the elements of a numpy array. You can check the entire list of available functions in the official [Numpy documentation](#). **Some examples:**

```
[11]: x =np.array([[1,2,3,4],[5,6,7,8]])
y =np.array([[9,10,11,12],[13,14,15,16]])
print(x+y)
print(y-x)
print(np.sqrt(y))
print(np.log(x))
print(x**2)
print(x+5)

[[10 12 14 16]
 [18 20 22 24]]
[[8 8 8 8]
 [8 8 8 8]]
[[3.          3.16227766 3.31662479 3.46410162]
```

(continues on next page)



(continued from previous page)

```

[3.60555128 3.74165739 3.87298335 4.          ]]
[[0.          0.69314718 1.09861229 1.38629436]
 [1.60943791 1.79175947 1.94591015 2.07944154]]
[[ 1  4  9 16]
 [25 36 49 64]]
[[ 6  7  8  9]
 [10 11 12 13]]

```

Note that in the last examples, we are adding a scalar value to a numpy array. In general, we can apply arithmetic operations on array of different dimensions, given that the smallest dimension between the operands is one, or that the arrays have the same dimensions. When this condition is met, numpy will expand the smaller array to match the shape of the larger array with an operation called **broadcasting**.

## Array functions

Numpy also provides an extensive list of array functions:

- **sum()**: Returns the sum of all elements.
- **min()**: Returns the minimum value within the array
- **max()**: Returns the maximum value within the array
- **mean()**: Returns the mean of an array
- **median()**: Returns the median value of the array
- **cumsum()**: Returns the cumulative sum of the elements of the array.

All the functions above support the additional **axis** parameter to work on a specific dimension.

```

[13]: x =np.array([[1,2,3,4],[5,6,7,8]])
      y =np.array([[9,10,11,12],[13,14,15,16]])

print("sum of all elements in x:")
print(np.sum(x))

print("mean value of y:")
print(np.mean(y))

sum of all elements in x:
36
mean value of y:
12.5

```

Other functions take two arrays as arguments and perform element wise operations:

- **minimum()**: Returns an array with the minimum value in each position of the array
- **maximum()**: Returns an array with the maximum value in each position of the array

```

[14]: b=np.linspace(0,1,10)
      r = c=np.random.random(10)
print(np.minimum(b,r))

[0.          0.11111111 0.22222222 0.0069694  0.44444444 0.3403326
 0.19167794 0.71257103 0.78045669 0.64287305]

```

```
[ ]:
```

### 1.5.3 The Matplotlib Pyplot Library

Try me



The **Matplotlib** (Mathematical plotting Library) is a Python charting package that allows us to render rich interactive graphics from Python data structures, including Numpy arrays.

To use Matplotlib in your Notebooks and programs, you first need to import the Pyplot package (in this example we use the alias `plt`):

```
[1]: import matplotlib.pyplot as plt
```

#### Matplotlib modes

We can use two different modes in Notebooks: - **Notebook**: This will create interactive graphics where you can zoom-in and move around the graph as well as export. You can use this mode for experiments, to interact with data directly in the book, but it is not recommended to use it for presentation.

- **Inline**: This will create a static graphic with the selected properties. We will work with this mode in this course

```
[2]: #We set the mode inline of matplotlib to get the result at the output of the cell code
%matplotlib inline
```

#### Main Concepts

Matplotlib introduces the following concepts:

- **Figure**: A figure is the canvas where we will plot our data. Note that a figure is not a graph, but the container that allows us to arrange one or more graphs for presentation.
- **Plots**: Plots or axes are the actual graphs included in the figure. A graph has a **title**, a **legend** and 2 or more **axis**.
- **Axis**: Conform the limits of the plot.

#### Creating bidirectional charts

In this course we will work mainly with bidirectional charts, therefore we will create figures with one axe (or plot) and two axis. There is a convenient Matplotlib function for this exact purpose:

- **plot(x,y,[fmt],[kwargs])**: This function takes an optional parameter x, a parameter y and optional parameters to format the line and pass additional formatting parameters to matplotlib.

For instance,

```
plot (y)
```

will plot the data in y versus the corresponding indexes.

```
plot (x, y)
```

will plot y versus x

```
plot (x, y, 'o--')
```

will plot y versus x using a dotted line and circular markers

```
plot(x,y, 'o--', label='random data')
```

will plot y versus x using a dotted line and circular markers and create a label for y with the text “random data”. Note that for the label parameter we can use expressions, for instance to write equations. For instance, using:

```
\python label='$x_{1}=f(y, \phi)$'
```

will create a label displayed as  $x_1 = f(y, \phi)$

Note that the plot function will not render the title or the legend of the plot. We need to use the functions:

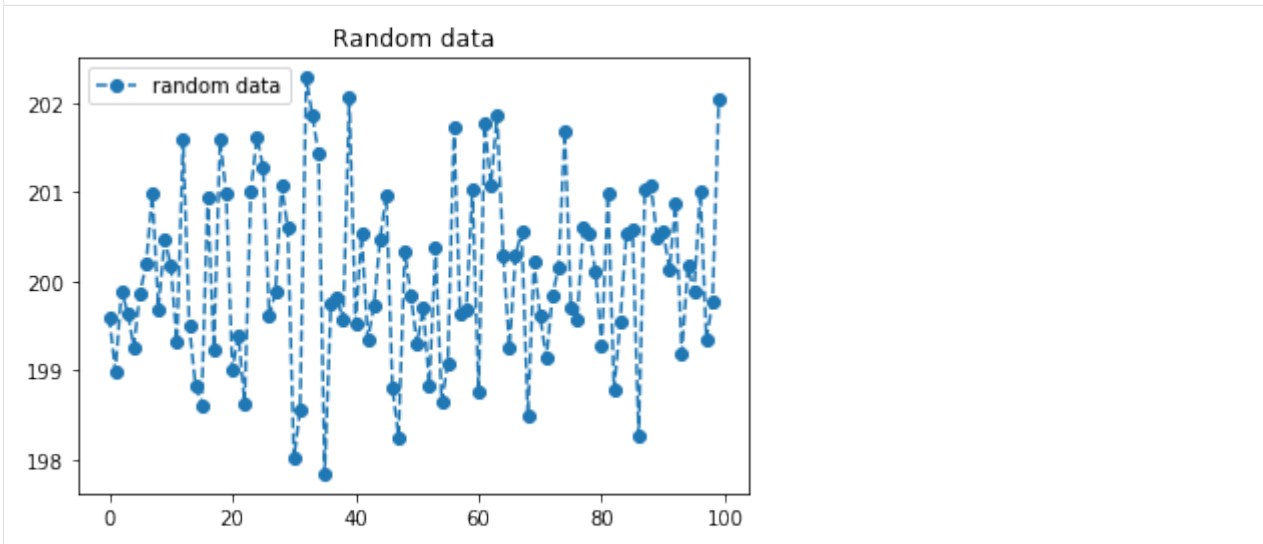
- **legend([kwargs]):** will render the legend, we can provide optional parameters to format the legend.
- **title(label):** will render the title with the text provided in the label parameter.

There are other signatures and many formatting parameters defined in the matplotlib library, but this is outside the scope of this tutorial. Check the documentation whenever you want to try to apply a different style.

```
[3]: import numpy as np
y = 200 + np.random.randn(100)
x = np.arange(y.size)

plt.plot(x,y, 'o--', label='random data')
#Add legend
plt.legend()
plt.title('Random data')
```

```
[3]: Text(0.5, 1.0, 'Random data')
```



The plot function allows us to effectively chart different lines into the same plot, just adding as many calls to the function as we want in the same code cell. For instance, in the example below we add two lines by calling twice the plot function with different data.

```
[35]: import numpy as np
y = 200 + np.random.randn(100)
y2 = 100 + np.random.randn(100)
x = np.arange(y.size)

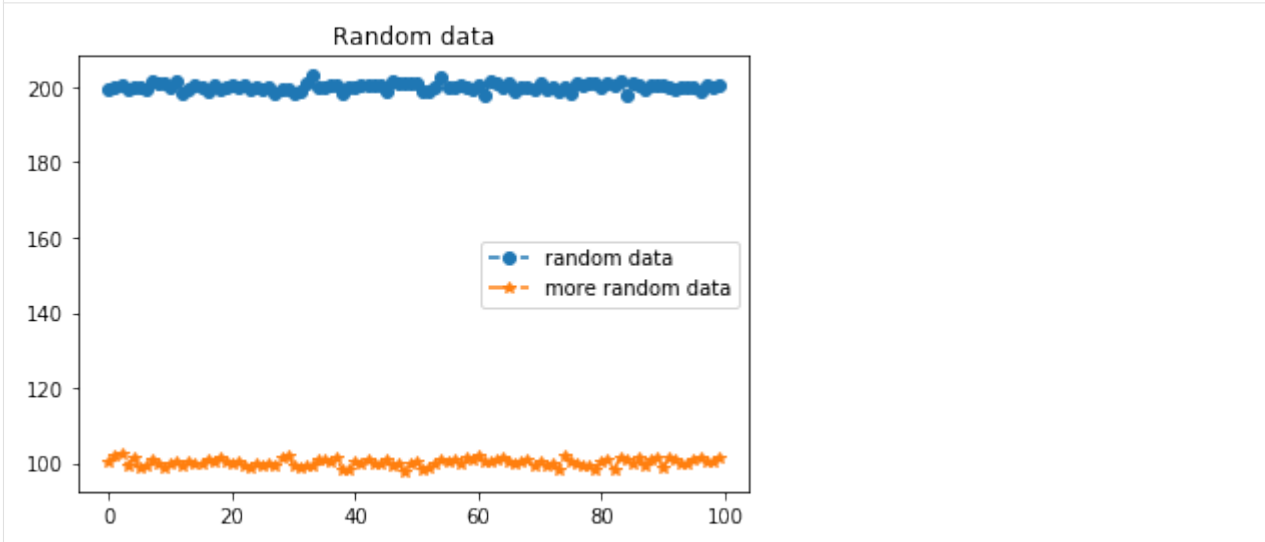
plt.plot(x,y, 'o--', label='random data')
plt.plot(x,y2, '*-.', label='more random data')
```

(continues on next page)

(continued from previous page)

```
#Add legend
plt.legend()
plt.title('Random data')
```

```
[35]: Text(0.5, 1.0, 'Random data')
```



## Formatting the axes

There are many different ways to format the axis. We will use the following convenient functions:

```
plt.xlim((0, 20))
plt.ylim((0, 15))
plt.xlabel(r'$x_{1}$')
plt.ylabel(r'$x_{2}$')
```

- **xlim(left, right)**: sets the limits of the x axis to the values provided. The limits are provided as a tuple of integer values, first the limit to the left of the plot (lower value) and then the limit to the right (higher value).
- **ylim(bottom, top)**: sets the limits of the y axis to the values provided. The limits are provided as a tuple of integer values, first the limit to the bottom (lower value) and then the limit to the top (higher value).
- **xlabel(label)**: sets the label for the x axis, using the text provided in the label parameter. We can provide expressions for instance to use equations in the labels (see example below).
- **ylabel(label)**: sets the label for the y axis, using the same parameter as the xlabel function describe above.

```
[10]: import numpy as np
y2 = 200 + 50*np.random.randn(100)
x = np.arange(y.size)

plt.plot(x,y2,label='random data')
#Add legend
plt.legend()
plt.title('Random data')

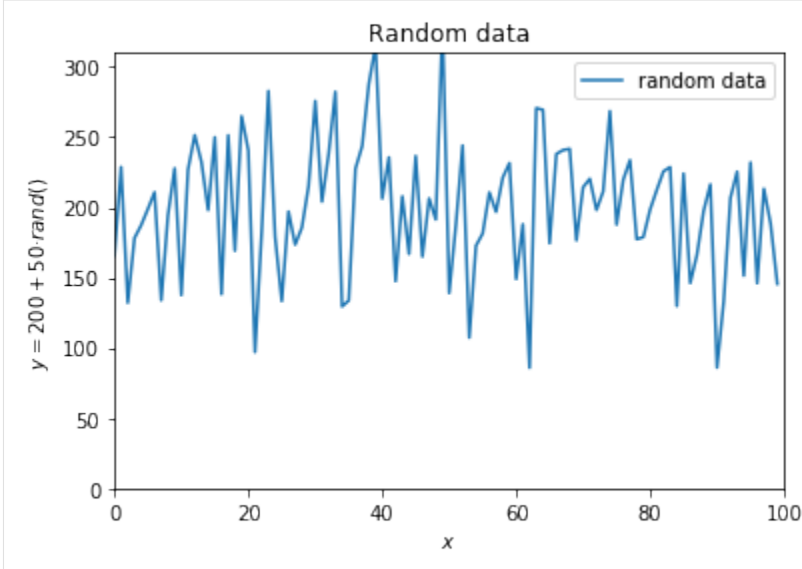
# sets limits
plt.xlim((0,100))
plt.xlabel(r'$x$')
```

(continues on next page)

(continued from previous page)

```
plt.ylim((0,310))
plt.ylabel(r'$y=200 + 50\cdot\text{rand}()$')
```

```
[10]: Text(0, 0.5, '$y=200 + 50\cdot\text{rand}()$')
```



## Filling areas

The function `fill_between` allows us to fill the area between any two lines:

**`fill_between(x, y, y2=0, where=None, kwargs)`:** Fills the area between lines `y` (upper limit of the area) and `y2` (bottom limit of the area), both versus `x`. Also, both `y` and `y2` can either be a line or a numerical value. If a scalar value is used, a horizontal line crossing the `y` axis at the numerical value provided is used as the bottom limit of the area. The optional `where` parameter is used to provide an expression in `x` to limit the area in the horizontal axis. Additional parameters are used to format the area. For instance, in the example below, we set up the color of the area and its transparency.

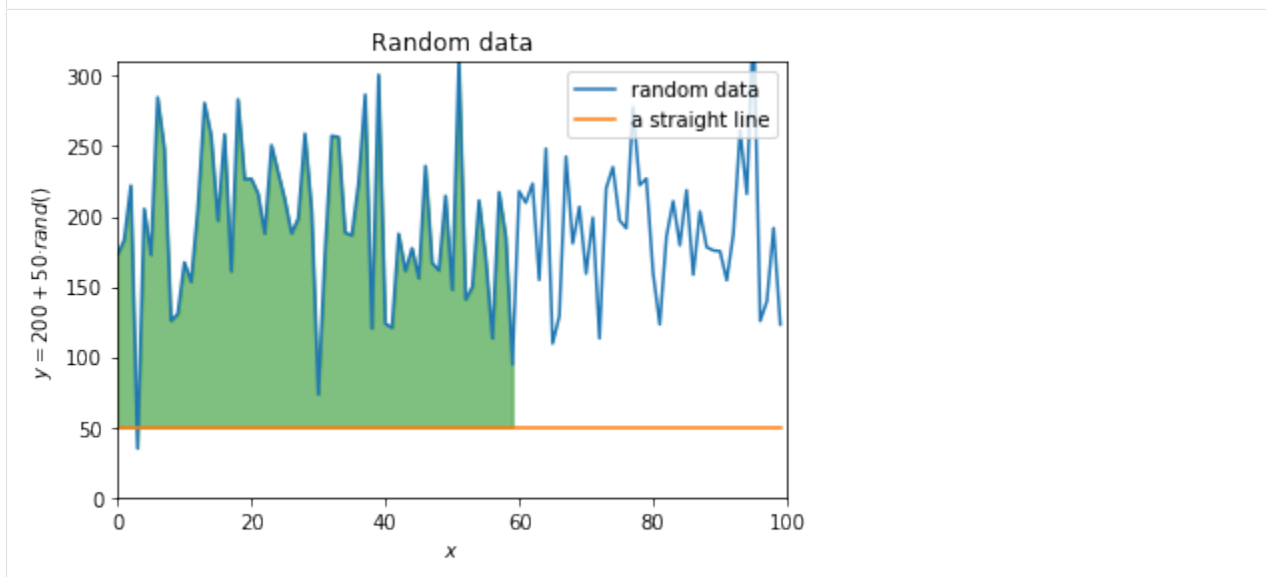
```
[37]: import numpy as np
y_rand = 200 + 50*np.random.randn(100)

x = np.arange(y.size)
y_const = 50 + 0*x
plt.plot(x,y_rand,label='random data')
plt.plot(x,y_const, label='a straight line')
#Add legend
plt.legend()
plt.title('Random data')

# sets limits
plt.xlim((0,100))
plt.xlabel(r'$x$')

plt.ylim((0,310))
plt.ylabel(r'$y=200 + 50\cdot\text{rand}()$')
plt.fill_between(x, y_rand, y2=y_const, where=x<60, color='green', alpha=0.5)
# since y2 is a constant, the line above is equivalent to
#plt.fill_between(x, y_rand, 50, where=x<60, color='green', alpha=0.5)
```

```
[37]: <matplotlib.collections.PolyCollection at 0x1e8fc64ccc0>
```



Note that in the last example, we are drawing the area between a line and a horizontal line and therefore we could just use `y2=50` in the `fill_between` function call.

### Drawing vertical lines

In order to draw a vertical line, we use a different pyplot function:

- `axvline(x=0, ymin=0, ymax=1,kwargs)**`: Draws a horizontal line at the scalar position `x` of the horizontal axis, the additional `ymin` and `ymax` parameters allow to set up the lower and upper limits of the line. Additional parameters are used to format the line.

Similarly, the function `axhline` can be used to draw horizontal lines.

```
[43]: import numpy as np
y_rand = 200 + 50*np.random.randn(100)

x = np.arange(y.size)
plt.plot(x,y_rand,label='random data')

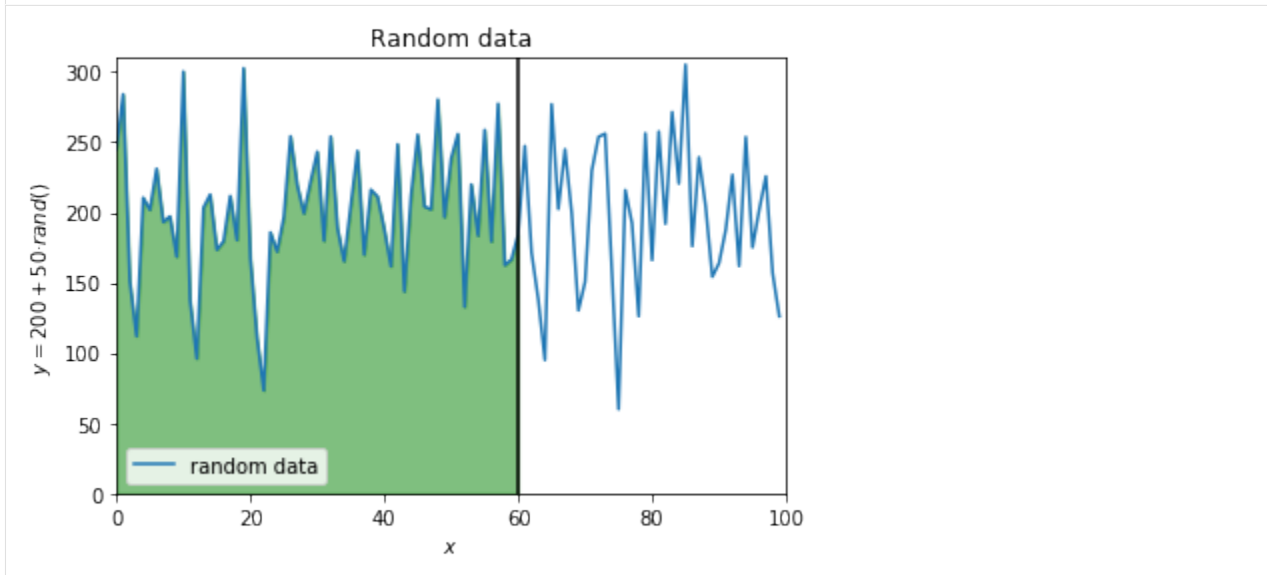
#Draw horizontal line
plt.axvline(60,0,310,color='black')

#Add legend
plt.legend()
plt.title('Random data')

# sets limits
plt.xlim((0,100))
plt.xlabel(r'$x$')

plt.ylim((0,310))
plt.ylabel(r'$y=200 + 50 \cdot rand()$')
plt.fill_between(x, y_rand, 0, where=x<=60, color='green', alpha=0.5)
# since y2 is a constant, the line above is equivalent to
#plt.fill_between(x, y_rand, 50, where=x<60, color='green', alpha=0.5)
```

```
[43]: <matplotlib.collections.PolyCollection at 0x1e8fc7cecf8>
```



### Extra: Annotating a point of the graph

As an extra (not needed to complete assignments, but used in the slides), we can annotate a point of the graph with the function `annotate`: - `**annotate(s, xy, *args, kwargs)`: Annotates the point `xy` with the string `s`

See the example below:

```
[63]: import numpy as np
y_rand = 200 + 50*np.random.randn(100)

x = np.arange(y.size)

#Calculate minimum coordinates
ymin = np.min(y_rand)
xmin=x[np.where(y_rand==ymin)]

# Calculate maximum coordinates
ymax = np.max(y_rand)
xmax = x[np.where(y_rand == ymax)]
plt.plot(x,y_rand,label='random data')

#Annotate minimum
plt.annotate('Min Value', (xmin[0],ymin))

#Annotate maximum
plt.annotate('Max value', (xmax[0], ymax))

#Add legend
plt.legend()
plt.title('Random data')

# sets limits
plt.xlim((0,100))
plt.xlabel(r'$x$')
```

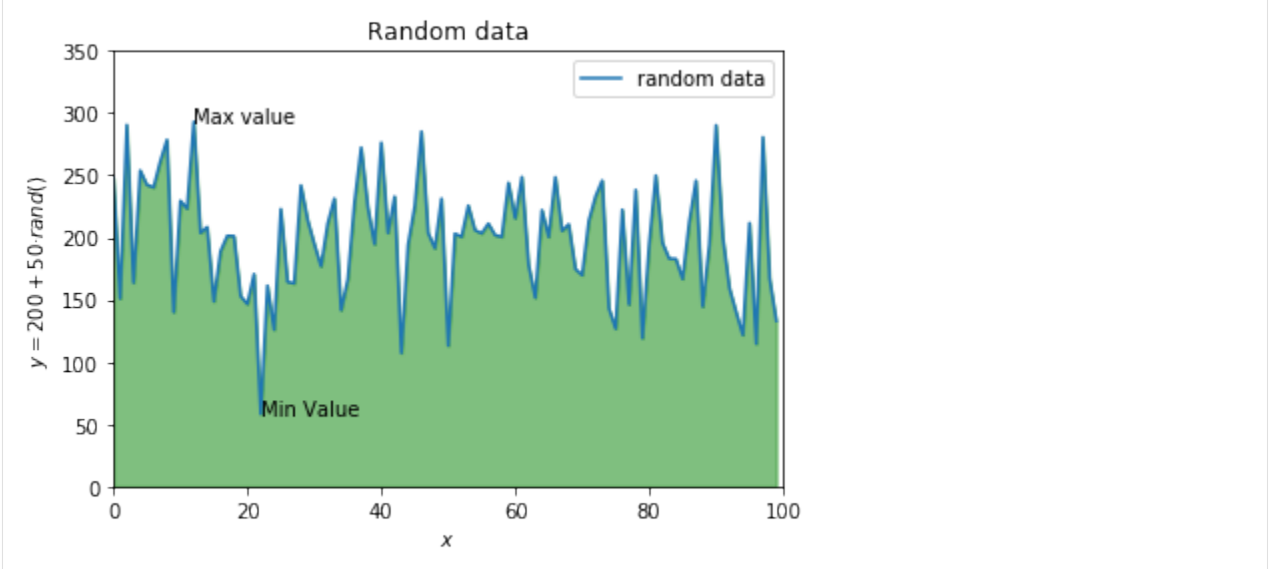
(continues on next page)

(continued from previous page)

```
plt.ylim((0, 350))
plt.ylabel(r'$y=200 + 50 \cdot \text{rand}()$')

plt.fill_between(x, y_rand, 0, color='green', alpha=0.5)
# since y2 is a constant, the line above is equivalent to
# plt.fill_between(x, y_rand, 50, where=x<60, color='green', alpha=0.5)
```

[63]: <matplotlib.collections.PolyCollection at 0x1e8fdfa0860>



[ ]:

### 1.5.4 Markdown Cells

Try me



Text can be added to IPython Notebooks using **Markdown cells**. Markdown is a popular **markup language**. In this context, markup refers to the information used to control how the document must be presented, rather than the content of the document itself. Markdown is therefore a **descriptive language** in the sense that we use the markup specification to control how text is presented and do not worry how the web browser actually renders it. Markdown is a superset of the HyperText Markup Language **HTML** which is used to build web pages, and in that sense, you will find that you can create the same type of documents you find over the Internet using Markdown. The specification can be found here:

<http://daringfireball.net/projects/markdown/>

🔗 Since you can view the source of a cell just by double-clicking on it, this guide is basically build using examples, and you can check the code just by clicking on the cell or while the cell is selected in command mode, press **Enter** to edit it. Once a cell has been edited, use **Shift-Enter** to re-render it.




## Markdown basics

### Code blocks

You can enter block codes in Markdown cells surrounding the text you want to display as code using triple quotes (`'''`) and adding the language name. For instance, the following code block will display some python code styled using Python language syntax:

```
# This is a comment
print("Hello, world!")
```

Note that you cannot execute the code in a markdown cell, but still, you can render the code in the web-browser. This is convenient to write documentation for your code embedded in the Markdown cell, or to explain bits of code written in a programming language not supported by the Notebook Kernel (e.g. SQL).

 In this Notebook, we will use code blocks with the `Markdown` language name so that we can render Markdown syntax in the web-browser. Keep this in mind when you check the source code of the Markdown cells below!

### Bold text and italics

You can make text *italic* or **bold**, surrounding the text with either `*` or double `**`. For instance, the code:

```
*italics*
```

will write down the word ‘italics’ in italics, and the code:

```
**bold text**
```

will write the text ‘bold text’ in bold.

### Headings

You can use the `#` character to create a heading. For instance, the code:

```
# This is a heading
```

Creates a heading with the text ‘This is a heading’. Headings are organized in levels, with level 1 being the largest and level 6 the smallest. Therefore, the code:

```
#### 1.1.1.1 Heading level 4 example
This is a paragraph in the heading level 4 example.

### 1.1.1.1.1 Heading level 5 example
This is a paragraph in the heading level 5 example.
```

will create styled headings of level 4 and 5.

### Lists

You can build itemized lists using the – character, and nested sub-lists using the tabulator or insert some spaces before the – character. For instance, the code:

```
- Item list number one
  - Sublist number one
    - This is a sublist item
  - Sublist number two
    - This is a sublist item
    - And this is another
- Item list number two
  - This is another sublist item
  - Here, just one more sublist:
    - One item
    - And one more!
```

Creates the following list:

- Item list number one
  - Sublist number one
    - \* This is a sublist item
  - Sublist number two
    - \* This is a sublist item
    - \* And this is another
- Item list number two
  - This is another sublist item
  - Here, just one more sublist:
    - \* One item
    - \* And one more!

You can also create numbered nested lists just using numbers before a ‘.’ character. For instance:

```
1. Here we go
  1. Sublist item 1
  2. Sublist item 2
2. Here we go again
  1. Sublist item 1
  2. Sublist item 2
```

Will create this sublist: 1. Here we go 1. Sublist item 1 2. Sublist item 2 2. Here we go again 1. Sublist item 1 2. Sublist item 2

## Additional Markdown syntax

While we covered the basics on the section above, this section includes some syntax tips that you can use to enhance the presentation of your document.

### Horizontal rules

Horizontal rules introduce a horizontal separator in your document, for instance: For instance, the code:

```
---
```

will create a horizontal rule like this:



### Block quotes

Block quotes are handy to highlight some text. For instance, the code:

```
> **Ben Parker**: With great power comes great responsibility.
```

will generate the following block quote:

**Ben Parker:** With great power comes great responsibility.

### Inline links

You can create links to other pages or to external resources using the following syntax:

```
This is a link to [IPython's website] (https://ipython.org)
```

will create the following link:

This is a link to [IPython's website](https://ipython.org)

### Inline images

You can insert images in your Markdown document using the following syntax:

```
![alternative text] (https://ipython.org/\_static/IPy\_header.png "Title: Logo")
```



Note that syntax includes an alternative text for the image, which is used when the image is not displayed (for instance when the content is transcribed by a screen reader). It is important to include a rich description of your image in the alternative text, so that it can be used by screen readers.

### Images with links

You can also insert images with links. For instance, the code:

```
[![alternative text] (https://ipython.org/_static/IPy_header.png "Title: Logo")] (https://ipython.org)
```

inserts an image that links to the iPython website, as shown below:



Note that basically, the only differences between previous examples is that we wrapped the image in the link description!

### LaTeX equations

You can include mathematical expressions using LaTeX notation wrapped using the dollar symbol as a delimiter. You can insert equations inline (in the same line) as text like:

```
This is an inline equation  $e^{i\pi} + 1 = 0$ 
```

This is an inline equation  $e^{i\pi} + 1 = 0$

You can also use the dollar symbol to include equations standalone in a new line. For instance, the code:

```

$$e^x = \sum_{i=0}^{\infty} \frac{1}{i!} x^i$$

```

Will introduce an equation in a new line, like this:

$$e^x = \sum_{i=0}^{\infty} \frac{1}{i!} x^i$$

### Latex cheat sheet

You can find a comprehensive cheat sheet for LaTeX equations [here](#). You can also use [Overleaf](#) to find tutorials and edit equations online. However, we hope this table contains the most common LaTeX commands and symbols.

Symbol	LaTeX command	Description	Example
$x^y$	<code>^{\text{x}}</code>	Exponent. The expression in the exponent is passed between <b>curly brackets</b> ( <code>{}</code> )	$x^{y+1}$
$x_i$	<code>_{\text{i}}</code>	Subscript. The expression in the subscript is passed between curly brackets	$x_{i+1}$
$\sum$	<code>\sum</code>	Summation. You can add subscripts and exponents to determine the range of the summation	$\sum_{i=0}^{10} i!$
$\frac{x}{y}$	<code>\frac</code>	Fraction. Pass first the numerator and then the denominator using curly brackets	$\frac{1}{2}$
$\int$	<code>\int</code>	Integration. Pass the expression to integrate and the variable to integrate over	$\int x^2 dx$
$\sqrt{x}$	<code>\sqrt</code>	Square root. Pass the expression to square root using curly brackets	$\sqrt{x^2}$
$\geq$	<code>\geq</code>	Greater than or equal to.	$\geq$
$\leq$	<code>\leq</code>	Less than or equal to.	$\leq$
$\neq$	<code>\neq</code>	Not equal to.	$\neq$
$\alpha$	<code>\alpha</code>	Any greek symbol, using its english name (e.g. alpha).	$\alpha$
$\beta$	<code>\beta</code>	Beta is another popular greek symbol.	$\beta$
$\gamma$	<code>\gamma</code>	Gamma is another great greek symbol.	$\gamma$
$\omega$	<code>\omega</code>	But Omega as in Omega level mutant is my favourite.	$\omega$

### Matrices and vectors in LaTeX

You can just use brackets to insert vectors in LaTeX, which is quite convenient. For instance, the code:

```
x = [1, 2, 3]
```

Defines a row vector  $x$  with the values 1, 2, 3.

Writing a matrix in LaTeX is a bit more complicated. You can use the following syntax:

```
\begin{bmatrix}
  1 & 2 & 3 \\
  4 & 5 & 6 \\
  7 & 8 & 9
\end{bmatrix}
```

To write a matrix like:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Note that we write the matrix row-by-row, using two backslashes to separate rows, and the `&` symbol to separate cells in a row.

## Tables

You can add a table like this in your document:

```
| This | is |  
|-----|-----|  
| a | table|
```

will display the following table:

This	is
a	table

The table cells are delimited by the (|) character. The first row and first column are used to determine the width of the table, so it is important to insert spaces if you want to control the table width and text alignment across the table. Also, note that the first row is used as headers, and that there is a separator between the header and the table body.

Note that the text in the table is automatically aligned to the left. You can control the alignment of the text using colons (: ) after the column delimiter. For instance, the code:

```
`` `markdown  
| Right aligned | centered aligned |  
|-----:|:-----:|  
| This is a right aligned column | and this one is centered |  
`` `
```

see:

Right aligned	centered aligned
This is a right aligned column	and this one is centered

## Local files

If you have local files in your Notebook directory, you can refer to these files in Markdown cells directly:

```
[subdirectory/]<filename>
```

For example, you can use this feature to include images that are not publicly available on the internet, but in your local directory. Say you have an image named `my_image.png` in a folder named `images` your notebook directory. You can then use the following code to include this image in your Markdown document:

```
![my_image.png] (images/my_image.png)
```

You can also embed a video and video controls with the HTML5 video tag:

```
<video controls src="images/animation.m4v" />
```

Note that, since videos are not natively supported in Markdown, we are using HTML5 notation, since Markdown also supports native HTML.

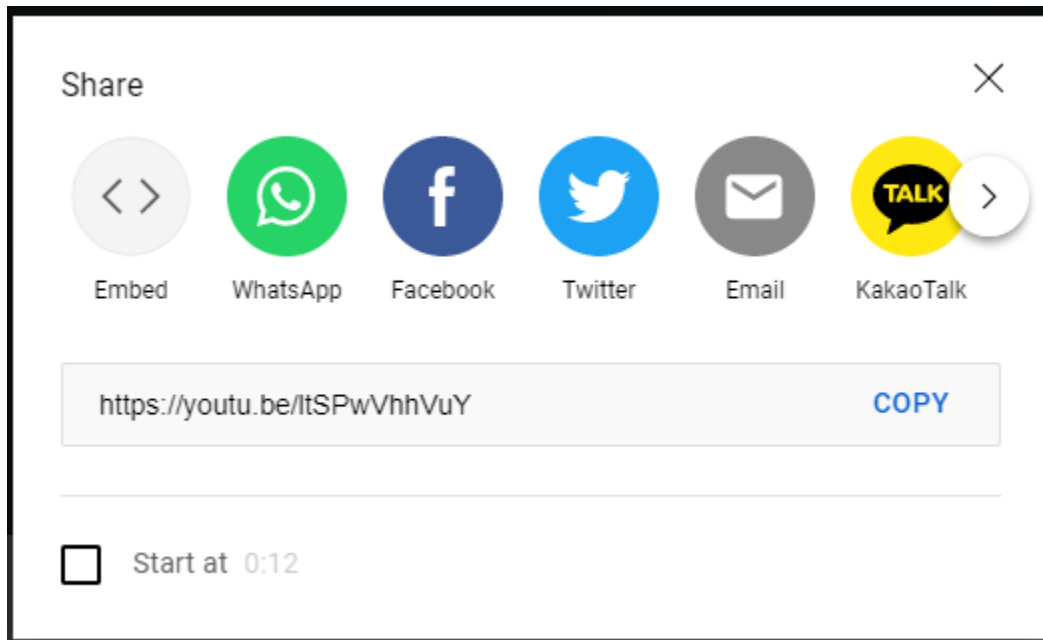
## Security of local files

Note that this means that the Notebook server also acts as a generic file server, and that all the files inside the same folder as your Notebook are available through the Notebook server. Although we will be mostly using sandbox environments like Google Colabs or Binder, bear this in mind if you ever use Notebook server in your system and do not run the Notebook server in a high level directory of your file system, like your home directory).

## Extra tips

### Youtube videos

You can embed YouTube videos using the *share* feature of YouTube. In the video you want to share in YouTube, just click on share and select the embed option:



As mentioned above, since you can embed HTML notation in the notebook, you can just copy it and paste it in your notebook like this:

```
<iframe width="560" height="315" src="https://www.youtube.com/embed/ltSPwVhhVuY"
↵title="YouTube video player" frameborder="0" allow="accelerometer; autoplay;
↵clipboard-write; encrypted-media; gyroscope; picture-in-picture" allowfullscreen></
↵iframe>```
```

These do not embed the data into the notebook file, and require that the files exist when you are viewing the notebook.

### ASCII symbols as icons

Some Notebook clients do support ASCII symbols, which are symbols that are encoded (converted into bits) and included into the set of characters (ASCII) just as letters. For example, the symbol for the heart is the *letter* ♥, or a pizza slice is the *letter* 🍕. You can just copy this special symbols and paste them in your Notebook to use them as icons.

### 1.5.5 Python for Operations Research II

Try me



#### Introduction

In previous tutorials, we covered few tips to work with lists and **Numpy** Arrays in Python. This tutorial presents two python types that will be used in the following tutorials, **tuples** and **dictionaries**.

#### Tuples

Tuples are very similar to lists, except that they are **immutable**. This means that, once you declare a tuple variable, you cannot make any changes to it (unless you are Chuck Norris). Tuples are created using parenthesis instead of brackets, like in the example below:

```
[33]: unknowns = ("x", "y", "z")
```

Since they are immutable, we can use tuples to store values that cannot change in our program. Tuples are **iterable**. We can access the values of a tuple using indexes and use them in control functions like *for* loops.

```
[34]: # Let's print the first unknown
print("The first unknown is: " + unknowns[0])

# Let's print all unknowns
for i in unknowns:
    print("Printing unknown: " + i)

The first unknown is: x
Printing unknown: x
Printing unknown: y
Printing unknown: z
```

#### Dictionaries

Dictionaries allow us to *map keys to values*. Note that both tuples and lists are *ordered* and that we can access the values of the elements using indexes. In dictionaries, we map a value to a key, so that we can use the key to access the value of an element, instead of its index. The syntax of dictionaries is:

```
my_dict = {key1: value, key2: value2}
```

Then, we can access the values in the dictionary using the keys. For instance:

```
[35]: my_dict = {'x': 1, 'y':2, 'z':3}
print(my_dict['x'])

1
```

You can access dictionary keys with the method `keys()`. This is useful to iterate on the keys of a dictionary:

```
[36]: for key in my_dict.keys():
    print('key is: ' + key + ' , value is: ' + str(my_dict[key]))
```



```
key is: x , value is: 1
key is: y , value is: 2
key is: z , value is: 3
```

Dictionaries also support comprehension. As with list comprehension, dictionary comprehension allows us to assign keys and values very efficiently using for loops. Let us see how it works by example:

```
[37]: keys = ('a', 'b', 'c', 'd')
      values = (1, 2, 3, 4)

      new_dict = {key:value for key in keys for value in values}
      print('key a has value ' + str(new_dict['a']))

      key a has value 4
```

## 1.5.6 Introduction to data manipulation and analysis with Pandas

Try me



Pandas is a powerful library that provides convenient data structures and functions to work with data. Throughout the course, we will use Pandas for data manipulation and analysis. To use Pandas in your project, you first need to install it in your environment. Additionally, in this tutorial we will import the display and Markdown libraries to display the dataframes as tables.

```
[ ]: !pip install pandas
      !pip install ipython
```

```
[1]: # Import pandas
      import pandas as pd

      # Import display and Markdown
      from IPython.display import display, Markdown
```

### Introduction to Pandas data structures

This introduction covers the fundamentals of Pandas data structures used in this course. It is however recommended that you also take a look to the [official documentation](#).

There are primarily two types of Pandas data structures:

- **Dataframes:** Dataframes are two-dimensional data structures, similar to a two-dimensional numpy array, a spreadsheet or an SQL table.
- **Series:** Series are one-dimensional data structures, similar to one-dimensional numpy arrays

### Dataframes

Data frames facilitates the manipulation and analysis of two-dimensional data structures, combining convenient functions to:

- **Extract** data from different sources, like CSV files, Excel sheets, or SQL databases
- **Transform** data, applying functions to change the values, or combining different data sources to prepare the data for our application
- **Load** data into output CSV files, Excel sheets or SQL databases
- **Analyse** Analyse the data

Pandas columns in a dataframe are not necessarily of the same type. In the course, we will primarily work with numeric types, string types and the datetime type, so we will focus on those two types in this tutorial. We will also store complex objects, and find convenient Pandas functions to manipulate the dataframe using them, so you can see that Pandas is going to be very useful throughout the course!

You can create a dataframe from different structures: DataFrame accepts many kinds. Among others, in this course, we will primarily use:

- A dictionary of one dimensional numpy arrays, lists, dicts, or Series
- CSV, or Excel files
- Another Pandas dataframe
- A Pandas series

Let us start with a simple example, using a dictionary.

```
[2]: # This is a dictionary
dict = {"one": [1, 2, 3, 4, 5], "two": [5, 4, 3, 2, 1], "three":["Awesome", "Brutal",
↪ "Cool", "Divine", "Exquisite"], "date": ['2021-01-01', '2021-01-02', '2021-01-02',
↪ '2021-02-03', '2021-02-05'] }

# And this is a dataframe
dataframe = pd.DataFrame.from_dict(dict)

display(dataframe)
```

	one	two	three	date
0	1	5	Awesome	2021-01-01
1	2	4	Brutal	2021-01-02
2	3	3	Cool	2021-01-02
3	4	2	Divine	2021-02-03
4	5	1	Exquisite	2021-02-05

Our Pandas dataframe, Pandas, has three columns, with names, 'one', 'two', and 'three'. Notice the integer number in the column to the left. This is the **index**, and by default, the index has no name.

Another convenient way to create a dataframe is to use a set of records in an array:

```
[3]: # This is an array of dict
records = [{"one": 1, "two": 5, "three": "Awesome", "date": "2021-01-01"},
          {"one": 2, "two": 4, "three": "Brutal", "date": "2021-01-02"},
          {"one": 3, "two": 3, "three": "Cool", "date": "2021-01-03"},
          {"one": 4, "two": 2, "three": "Divine", "date": "2021-02-04"},
          {"one": 5, "two": 1, "three": "Exquisite", "date": "2021-02-05"}
]
```

(continues on next page)

(continued from previous page)

```
# This is a dataframe
dataframe_2 = pd.DataFrame.from_records(records)
display(dataframe)
```

	one	two	three	date
0	1	5	Awesome	2021-01-01
1	2	4	Brutal	2021-01-02
2	3	3	Cool	2021-01-02
3	4	2	Divine	2021-02-03
4	5	1	Exquisite	2021-02-05

Another convenient function to create a dataset is loading a csv file:

```
online_retail = pd.read_csv('datasets/Online Retail_2.csv', usecols=['InvoiceNo',
↪ 'StockCode', 'Quantity', 'InvoiceDate'])
```

This line of code will create a dataset from the file 'datasets/Online Retail\_2.csv', loading only data from the columns labeled as 'InvoiceNo', 'StockCode', 'Quantity', and 'InvoiceDate'.

### Accessing data

There are multiple ways to access the data in a dataframe. Dataframe columns are **series**, the other important data structure in Pandas. You can access columns in a dataframe using column names as keys:

```
[4]: names = dataframe['three']
display(names)
```

```
0      Awesome
1      Brutal
2         Cool
3      Divine
4    Exquisite
Name: three, dtype: object
```

Notice that series are also indexed, you can access *elements* or *slices* just as in numpy:

```
[5]: print(names[0])
print(names[1])
print(names[0:2])
```

```
Awesome
Brutal
0      Awesome
1      Brutal
Name: three, dtype: object
```

Obviously, you can access the elements directly from the data frame:

```
[6]: print(dataframe['three'][0])
```

```
Awesome
```

### Queries

You can use queries to access parts of your dataframe, that is, to *filter* the data, using python expressions. The function `query()` is used to issue the query. For instance, the following query finds all rows where the column 'one' is less or equal than two:

```
[7]: filtered_df = dataframe.query('one <= 2')
display(filtered_df)
```

	one	two	three	date
0	1	5	Awesome	2021-01-01
1	2	4	Brutal	2021-01-02

You can combine different columns in the expressions, for instance, the following example filters data where the column 'two' is greater than 3 and the column 'three' is equal to "Awesome". The second example shows values where the column 'one' is equal to the column 'two'

```
[8]: filtered_df2 = dataframe.query('two > 3 and three == "Awesome"')
display(filtered_df2)
filtered_df3 = dataframe.query('one == two')
display(filtered_df3)
```

	one	two	three	date
0	1	5	Awesome	2021-01-01

	one	two	three	date
2	3	3	Cool	2021-01-02

### Vectorized functions

Pandas series have a huge set of vectorized functions you can apply to the entire series. For instance, the different functions in the numpy tutorial:

```
[9]: display(dataframe['two'].sum())
display(dataframe['one'].cumsum())
display(dataframe['two'].mean())
```

15

0	1
1	3
2	6
3	10
4	15

Name: one, dtype: int64

3.0

One interesting function is `describe()`, which returns another dataframe with statistical information about the series or dataframe:

```
[10]: dataframe.describe()
```

```
[10]:
```

	one	two
count	5.000000	5.000000
mean	3.000000	3.000000
std	1.581139	1.581139

(continues on next page)

(continued from previous page)

```
min    1.000000  1.000000
25%    2.000000  2.000000
50%    3.000000  3.000000
75%    4.000000  4.000000
max    5.000000  5.000000
```

Another important vectorized function is `unique()`. We will use `unique()`

All string type functions are vectorised and can be accessed through `str`. For instance, the following example gets the first character of the string:

```
[11]: upper_three = dataframe['three'].str.upper()
display(upper_three)
```

```
0    AWESOME
1    BRUTAL
2     COOL
3    DIVINE
4  EXQUISITE
Name: three, dtype: object
```

`str` provides vectorized access to the string value of the series, and thus you can access parts of every string in a series using indexes:

```
[12]: first_character = dataframe['three'].str[0]
display(first_character)
```

```
0    A
1    B
2    C
3    D
4    E
Name: three, dtype: object
```

Similar to `str`, `dt` implements vectorized functions to work with dates. The following examples first transforms column “date” into a datetime, and then calculates the month.

```
[13]: month = pd.to_datetime(dataframe['date']).dt.to_period("M")
display(month)
```

```
0    2021-01
1    2021-01
2    2021-01
3    2021-02
4    2021-02
Name: date, dtype: period[M]
```

The function `pd.to_datetime` converts the series in column ‘date’ into a datetime. Then, `dt.to_period()` transforms this column into a period, in this case, the letter “M” represents *months*, and therefore, the period obtained is months.

Finally, `apply` allows us to apply and `lambda` function taking all the values of the dataframe or just a series:

```
[14]: plus_one = dataframe['one'].apply(lambda x: x + 1)
display(plus_one)
```

```
0    2
1    3
2    4
3    5
```

(continues on next page)

(continued from previous page)

```
4      6
Name: one, dtype: int64
```

In the code above, we apply a newly created lambda function that takes one parameter *x* and just adds one. `apply()` will apply this function to the series in column 'one'. There is a huge set of functions available, and many things you can do which are not covered in this tutorial, so you better check the documentation at [Pandas Series API Reference](#)

### Indexing

By default, the dataframes we have created have an integer *index* to access data in a row, but we can define any type of index. Let us create another dataframe with another type of index:

```
[15]: data_2 = {"2021-01-01": ['003', '46021'], "2021-01-02": ['005', '46022'], "2021-01-03":
↳: ['007', '46022'],
        "2021-02-04": ['003', '46021'], "2021-02-05": ['003', '46027']}

dataframe_3 = pd.DataFrame.from_dict(data_2, orient='index', columns=['client_id',
↳: 'postal_code'])
display(dataframe_3)
```

	client_id	postal_code
2021-01-01	003	46021
2021-01-02	005	46022
2021-01-03	007	46022
2021-02-04	003	46021
2021-02-05	003	46027

We have used two new parameters to create our dataframe from a dictionary (`from_dict`), *orient*, and *columns*. When *orient* is `index`, the dictionary keys are used as the index of the created dataframe. In this case, we are using a string. The columns parameters allow us to give a name to the columns of our dataframe. Now, we can access the data using the new index:

```
[16]: print(dataframe_3['client_id']['2021-01-01'])

003
```

We can also set the index of an existing dataframe using its columns. Let us change the index of our first dataframe:

```
[17]: new_dataframe = dataframe.set_index(['one'])
display(new_dataframe)
```

	two	three	date
one			
1	5	Awesome	2021-01-01
2	4	Brutal	2021-01-02
3	3	Cool	2021-01-02
4	2	Divine	2021-02-03
5	1	Exquisite	2021-02-05

Notice, that since our column had a name, now our index also has a name. We can set the name of an index like:

```
[18]: dataframe_3.index.set_names(['date'], inplace=True)
display(dataframe_3)
```

	client_id	postal_code
date		

(continues on next page)

(continued from previous page)

2021-01-01	003	46021
2021-01-02	005	46022
2021-01-03	007	46022
2021-02-04	003	46021
2021-02-05	003	46027

We can use different columns as index to create a dataframe similar to a **pivot table**:

```
[19]: dataframe_4 = dataframe.set_index(['one', 'two', 'three'])
display(dataframe_4)
print(dataframe_4['date'][1][5]['Awesome'])
```

```

                date
one two three
1  5  Awesome  2021-01-01
2  4  Brutal   2021-01-02
3  3  Cool     2021-01-02
4  2  Divine   2021-02-03
5  1  Exquisite 2021-02-05
```

2021-01-01

### New columns

To add a new column, you just need to define the column name. For instance, let us to the original dataframe a new column containing the month:

```
[20]: dataframe['month'] = pd.to_datetime(dataframe['date']).dt.month
display(dataframe)
```

```

   one  two  three  date  month
0    1    5  Awesome 2021-01-01    1
1    2    4  Brutal  2021-01-02    1
2    3    3    Cool  2021-01-02    1
3    4    2  Divine  2021-02-03    2
4    5    1  Exquisite 2021-02-05    2
```

### Grouping

Grouping allows us to group by values that have the same index value:

```
[21]: ones_in_month = dataframe.set_index(['month']).groupby(['month']).sum()
display(ones_in_month)
```

```

      one  two
month
1         6   12
2         9    3
```

Note that, first, we define the index month, and then we group by this index. Finally, we apply the vectorized function sum to sum all values of the columns with the same value of the index month. Column three silently disappears since its type does not support this function.

### Joining

Indexes allow us to join different dataframes that have an index with the same name:

```
[22]: dataframe_5 = dataframe.set_index(['date'])
dataframe_6 = dataframe_5.join(dataframe_3, how="inner")
display(dataframe_6)
```

date	one	two	three	month	client_id	postal_code
2021-01-01	1	5	Awesome	1	003	46021
2021-01-02	2	4	Brutal	1	005	46022
2021-01-02	3	3	Cool	1	005	46022
2021-02-05	5	1	Exquisite	2	003	46027



## CONTINUOUS LINEAR PROGRAMMING

In this chapter, we will cover both theoretical and practical topics related to Continuous Linear Programming (CLP) using Python. CLP allows us to model, solve and analyse the solution of optimization problems using linear equations. “The world is not linear” but still, CLP is a really powerful tool to solve a plethora of real world optimisation problems.

### 2.1 Tutorials

This section contains some basic tutorials:

#### 2.1.1 Continuous Linear Programming

##### Definition

Continuous Linear Programming (CLP) spans a wide set of OR problems that can be modeled based on the following assumptions:

- **Unknown variables are continuous:** The result variable and the decision variables can take any real value. Also, all the different coefficients can take any real value.
- **Objectives and constraints are linear expressions:** The relationships between the variables are expressed as linear expressions.

Clearly, CLP establishes a rather simplified model of real world problems. Many assumptions are needed to represent real problems as a set of linear expressions. Yet, CLP is a very powerful tool in OR and it can provide valuable insights to decision-making in different fields. The main reason is that, once these assumptions are taken, linear algebra provides valuable tools to find an optimal solution and analyse it.

The main types of problems that will be covered in the CLP exercises belong to two problem types:

- **Production mix:** Production mix problems help us determine the optimal allocation of resources in the production of good or services to either minimise costs or maximise profits
- **Blending problems:** In blending problems, the objective is to find the optimal combination of components in a set of products that either minimise costs or maximise profits

## Set up

In CLP, the objective variable, noted as  $z$ , can take any real value:

$$z \in \mathbb{R}$$

The value of this unknown variable  $z$  is a function of another set of unknowns, which are called decision variables and that represent our decisions. These decision variables are noted as  $x_j$ , where  $j$  is an integer (sub)index that goes from 1 to  $n$ :

$$z = f(x_1, x_2, \dots, x_n)$$

$$x_1, x_2, \dots, x_n \in \mathbb{R}$$

Therefore, we will have  $n$  different decision variables, that are going to be continuous. And, in extension,  $z$  is also going to be continuous. So now we know from where does the  $C$  in CLP comes from. Next, in CLP, the function  $f$  is a linear function. What we want is to find the **optimal value** of  $z$ , which, depending on the type of problem will be either the maximum value (e.g. maximise profits) or the minimum value (e.g. minimise costs).

Hence, we want to maximise or minimise  $z$ , which as we said is a linear function and therefore can be expressed as the sum of the product of the decision variables multiplied by a set of coefficients, which are noted as  $c_1$  to  $c_n$ .

$$\max \text{ or } \min z = f(x_1, x_2, \dots, x_n) = c_1 \cdot x_1 + c_2 \cdot x_2 + \dots + c_n \cdot x_n$$

Now, we know that linear functions do not have any absolute minimum or maximum, but the objective function is subject to a set of constraints, which are also linear functions of the decision variables, that is, the sum product of the decision variables times a set of coefficients noted as  $a_{ij}$  must be less than, greater than, or equal to another coefficient noted as  $b_j$ .

s.t.

$$a_{11} \cdot x_1 + a_{12} \cdot x_2 + \dots + a_{1n} \cdot x_n \leq b_1$$

$$a_{21} \cdot x_1 + a_{22} \cdot x_2 + \dots + a_{2n} \cdot x_n \leq b_2$$

...

$$a_{m1} \cdot x_1 + a_{m2} \cdot x_2 + \dots + a_{mn} \cdot x_n \leq b_m$$

Note that coefficients  $a$  have two sub-indexes, the second is equal to the index of the corresponding decision variable, and the first is equal to the sub-index of  $b$ . These new index  $i$  ranges from 1 to  $m$  and therefore, our optimisation function is subject to a set of  $m$  constraints. We will refer to the expressions to the left of the inequality as the Left Hand Side (LHS), and to the other side of the relationship as the Right Hand Side (RHS).

Now, note that we can use the sum operator to represent the objective function in a more compressed form, that is, the LHS is equal to the sum of the product of  $c_j$  times  $x_j$  for  $j$  equal to 1 up to  $n$ . Note that this is can also be represented as the dot product of two vectors, the vector  $x$  with the decision variables and the vector  $c$  of the coefficients.

$$\max \text{ or } \min z = c_1 \cdot x_1 + c_2 \cdot x_2 + \dots + c_n \cdot x_n = \sum_{j=1}^n c_j \cdot x_j = c \cdot x$$

$$x = [x_1, x_2, \dots, x_n]^T$$

$$c = [c_1, c_2, \dots, c_n]^T$$

We can do the same transformations to the left hand sides of the constraints, expressing them as the sum product of the decision variables times the left hand side coefficients. And, we may as well express all the left hand sides of the constraints as the product of a matrix  $A$  that contains all the different coefficients times the decision variable vector.

$$\max \text{ or } \min z = \sum_{j=1}^n c_j \cdot x_j$$

s.t.

$$\sum_{j=1}^n a_{1j} \cdot x_j \leq b_1$$

$$\sum_{j=1}^n a_{2j} \cdot x_j \leq b_2$$

...

$$\sum_{j=1}^n a_{mj} \cdot x_j \leq b_m$$

or

$$\max \text{ or } \min z = c \cdot x$$

s.t.

$$A \cdot x \leq b$$

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

$$b = [b_1, b_2, \dots, b_m]$$

These alternative compact forms allow us to deal with problems with an arbitrary number of decision variables and an arbitrary number of constraints.

## Problems

This section further explains the types of problems that can be solved with CLP.

### Production Mix Problems

In production mix problems, the decision maker needs to create products to satisfy demand using limited resources. The objective is to maximise the profits, taking into account the demand and the limited availability of resources. Production mix problems represent the simplest form of optimization problem in operations research, where the objective is to maximise a utility function (in this case profits) bounded by a set of constraints. Normally, production mix problems are not found in real world applications, but rather, they are part of larger problems.

#### Objective function

Let us note the objective variable profits as  $z$ , expressed in a monetary unit (e.g. €).

- $z$ : profits in €.

The decision maker needs to decide how many units of each product type to manufacture. Let us assume there are  $n$  different products. Now, let us note the decision variables  $x_j \in \mathbb{R}$  as the amount of product  $j \in [1, \dots, n]$  to be produced, in a production unit:

- $x_j$ : Quantity of product  $j$  to be produced in product units.

The objective is to maximize profits. Let us note the profit per unit of product  $j$  as  $p_j$ :

- $p_j$ : Profit per unit of product  $j$  to be produced in €/unit.

Now, the objective function is to maximise the sum of the product of the number of units produced times the profit per unit, for each product type:

$$\max z = x_1 * p_1 + x_2 * p_2 + x_3 * p_3 + \dots + x_n * p_n = \sum_{j=1}^n x_j * p_j$$

### Demand constraints

As mentioned above, there might be requirements on the demand. Let us note the demand of product  $j$  as  $d_j$ . The demand constraints will be of the type:

$$x_1 \geq d_1$$

$$x_2 \geq d_2$$

...

$$x_n \geq d_n$$

We can write the demand constraints in a compact form like:

$$x_j \geq d_j \quad \forall j \in [1, \dots, n]$$

### Availability constraints

Different limited resources are used to produce each product. Let us assume that we have  $m$  different resources. Let us note as  $b_i$  as the quantity available of resource  $i$  ( $i \in [1, \dots, m]$ ), expressed in a given unit:

- $b_i$ : Availability of resource  $i$  in resource unit.

Further, to produce a unit of a product  $j$  we will require a given amount of resource  $i$ . Let us note as  $a_{ij}$  the requirement of resource  $i$  to produce a unit of product  $j$ :

- $a_{ij}$ : Requirement of resource  $i$  to produce a unit of product  $j$  in resource unit / product unit.

Then, for each resource  $i$ , we need to ensure that the total amount used is less or equal the total amount available. The amount used to produce a specific number of products will be the number of units products (e.g.  $x_j$ ) multiplied by the amount required to produce one unit  $x_{ij}$ . If we sum across all products, we obtain the total amount used, which needs to be lower than the availability. Let's take the resource indexed 1 as an example:

$$x_1 * a_{11} + x_2 * a_{12} + \dots + x_n * a_{1n} \leq b_1$$

Note that we can write this down this constraint in a compact form like:

$$\sum_{j=1}^n x_j * a_{1j} \leq b_1$$

Now, we need to take into account the other resources:

$$\sum_{j=1}^n x_j * a_{2j} \leq b_2$$

$$\sum_{j=1}^n x_j * a_{3j} \leq b_3$$

...

$$\sum_{j=1}^n x_j * a_{mj} \leq b_m$$

Note that we can write all this  $m$  equations in a compact form like:

$$\sum_{j=1}^n x_j * a_{ij} \leq b_i \quad \forall i \in [1, \dots, m]$$

That is, for every resource, we need to ensure that the sum of the quantities used to produce each type of product are less or equal than the total amount available.

## Summary

Putting it all together, the generic formulation of the production mix problem as a CLP is:

$$\max z = \sum_{j=1}^n x_j * p_j$$

s.t.

$$\sum_{j=1}^n x_j * a_{ij} \leq b_i \quad \forall i \in [1, \dots, m]$$

$$x_j \geq d_j \quad \forall j \in [1, \dots, n]$$

## Examples

The following exercises are instances of the production mix problem:

- Production mix
- Ski Patrol
- The good carpenter

## Blending Problems

In blending problems, the decision maker needs to find the optimal blending (mixing or combination) of materials or resources to create **products** that satisfy a given demand. Each resource or material will have a different costs and therefore, in its primary form, the objective is to minimise the overall costs of the materials needed to satisfy the demand. Additionally, the required materials or resources needed may be subject to availability constraints (i.e. we cannot consider that the amount of materials or resources available is unlimited).

### Objective function

Let us assume that we have  $m$  different materials or resources to be considered in the problem and  $n$  different products.

Let us note as  $c_i \quad i \in [1, \dots, m]$  the cost of product  $i$ . Let us also note  $x_{ij}$  the amount of resource or material  $i$  used to produce products  $j \quad j \in [1, \dots, n]$ .

Note that these variables  $x_{ij}$  represent our decision variables: We are looking for the optimal combination of resources or materials to manufacture our products. Note also that we have  $m * n$  different decision variables (the combination of products and materials).

With these definitions, our objective function can be defined as:

$$\min z = \sum_{i=1}^m \sum_{j=1}^n c_i * x_{ij}$$

That is, the objective is to minimize the overall costs, which is the sum of the cost of the different materials needed to manufacture all our products.

### Availability constraints

Since the amount of resources or materials available is limited, let us plug in the availability constraints, let us note as  $a_m$  as the availability of the material or resource  $m$ . The total amount used to manufacture all products cannot exceed the availability of that specific material or resource. For instance, for material or resource 1, the sum of the amount used to manufacture the different products cannot exceed the availability  $a_1$ , that is:

$$\sum_{j=1}^n x_{1j} \leq a_1$$

The same applies to other material or resources:

$$\sum_{j=1}^n x_{2j} \leq a_2$$

...

$$\sum_{j=1}^n x_{mj} \leq a_m$$

We can write this set of constraints in a compact form as:

$$\sum_{j=1}^n x_{ij} \leq a_i \quad i \in [1, \dots, m]$$

### Demand constraint

As mentioned above, we need to satisfy the demand for every product, which means that the total amount of a product that we need to manufacture needs to be greater or equal than its demand. Let us note  $d_j$  as the demand for product  $j$ .

Note that we do not have explicitly defined a decision variable for the quantity of each product to manufacture, but in terms of the blending problem, if all the materials are considered, it is equivalent to think of the total amount of a product as the total amount of the materials needed to manufacture a product. Hence, the demand constraints can be expressed as:

$$\sum_{i=1}^m x_{i1} \geq d_1$$

$$\sum_{i=1}^m x_{i2} \geq d_2$$

...

$$\sum_{i=1}^m x_{in} \geq d_n$$

Note that we can express this set of constraints in a compact form as:

$$\sum_{i=1}^m x_{ij} \geq d_j \quad j \in [1, \dots, n]$$

### Additional requirements: Minimum amount

We may find additional requirements in the problem definition, for instance, we may find a requirement for a minimum amount of an ingredient. Sometimes the requirement is expressed as a percentage or ratio of the total composition of the product. As an example, let us define that the minimum amount of material or resource 2 for product 3 needs to be 20%, or 0.2. If we are dealing with all the materials and resources used, we can again assume that the amount of material or resource  $x_{23}$  needs to be at least 20% of the sum of all ingredients used to manufacture product 3, that is:

$$x_{23} \leq 0.2 * \sum_{i=1}^m x_{i3}$$

We could bring all the dependant coefficients to the left-hand side and express this constraint as:

$$0 \leq -1 * x_{23} + 0.2 * \sum_{i=1}^m x_{i3}$$

$$0 \leq -0.8 * x_{23} + 0.2 * \sum_{i=1, i \neq 2}^m x_{i3}$$

$$0.8 * x_{23} - 0.2 * \sum_{i=1, i \neq 2}^m x_{i3} \geq 0$$

## Combination of production mix and blending problems

Instead of just minimizing the overall cost, our objective can be to maximise profit. This can be regarded as a combination of the production mix and blending problems. Let us define the unitary selling prices of our  $n$  products as  $[p_1, p_2, \dots, p_n]$

If we focus on one product, e.g. product 1, the profits can be calculated as the difference between the selling price and the cost obtained for the total amount of the products sold. Again, if we can assume that the total amount of ingredients is equal to the total amount of products, expressed in the same unit as the unitary selling price, the overall profits for product 1 can be expressed as:

$$p_1 * \sum_{i=1}^n x_{i1} - \sum_{i=1}^n c_i * x_{i1}$$

Our objective is to maximize the overall profits, so we need to factor in the profits of the rest of the products with a summatory:

$$\max z = \sum_{j=1}^n p_j * \sum_{i=1}^n x_{ij} - \sum_{i=1}^n c_i * x_{ij}$$

Now, we can re-arrange the terms of the summations as:

$$\max z = \sum_{j=1}^n \sum_{i=1}^n (p_j - c_i) * x_{ij}$$

## Examples

The following exercises represent blending problem instances:

- [Blending Problem](#)
- [Petroleum Blending](#)
- [Blending Craft Beer](#)

## Standard Form

Some algorithms like the Simplex need to operate with problems formulated in a standard form. Every CLP problem described in the set-up section can be converted to a standard form where:

- The objective function is of type **maximise**
- All the constraints are of type **equal**

That is, any CLP problem can be expressed in the standard form as:

$$\max z = \sum_{i=1}^n c_i * x_i = c^T$$

$$\text{s.t. } a_{11} \cdot x_1 + a_{12} \cdot x_2 + \dots + a_{1n} \cdot x_n + s_1 = b_1$$

$$a_{21} \cdot x_1 + a_{22} \cdot x_2 + \dots + a_{2n} \cdot x_n + s_2 = b_2$$

...

$$a_{m1} \cdot x_1 + a_{m2} \cdot x_2 + \dots + a_{mn} \cdot x_n + s_n = b_m$$

which can be expressed using a matrix product as:

$$A \cdot x = b$$

and

$$x_i \geq 0 \quad \forall i$$

### Conversion of minimization problems

If we need to convert a problem where the objective function is of type minimize to the standard form, we need just need to make a change of variable:

$$z = -z^*$$

With this change of variable the objective function:

$$\min z = c \cdot x$$

is equivalent to:

$$\max z^* = -c \cdot x$$

And now the objective function is of type maximise.

### Conversion of constraints and slack variables

Note that we have introduced a new set of decision variables  $s = [s_1, s_2, \dots, s_n]$  used to make all the constraints of type equal. These are new variables that are introduced to change the type of constraint to a constraint of type equal. Imagine for instance that our problem has a constraint of type less or equal:

$$a_{11} \cdot x_1 + a_{12} \cdot x_2 + \dots + a_{1n} \cdot x_n \leq b_1$$

Since the left hand side is less or equal than the right hand side, we can make the left hand side equal to the right hand side by adding a non-negative variable  $s_1$  to the left hand side:

$$a_{11} \cdot x_1 + a_{12} \cdot x_2 + \dots + a_{1n} \cdot x_n + s_1 = b_1$$

Obviously, we need to add a non-negativity constraint to the new variable to make sure that both constraints are equivalent:

$$s_1 \geq 0$$

Equivalently, if we have a constraint of type greater or equal, we can make the same modification, but now, the new variable needs to be lower or equal than zero to make the problem equivalent. For instance, let us consider without loss of generality that we have the following constraint in our problem:

$$a_{21} \cdot x_1 + a_{22} \cdot x_2 + \dots + a_{2n} \cdot x_n \geq b_2$$

This is equivalent to:

$$a_{21} \cdot x_1 + a_{22} \cdot x_2 + \dots + a_{2n} \cdot x_n + s_2 = b_2$$

where:

$$s_2 \leq 0$$

Also, if our constraint is of type equal, we do not need to introduce any change, but just to make sure we treat all constraints in the same way, let us introduce again a new variable:

$$a_{31} \cdot x_1 + a_{32} \cdot x_2 + \dots + a_{3n} \cdot x_n = b_3$$

This is equivalent to:

$$a_{31} \cdot x_1 + a_{32} \cdot x_2 + \dots + a_{3n} \cdot x_n + s_3 = b_3$$

where:

$$s_3 = 0$$



## Slack variables

These new variables are called **slack variables**, and they represent the difference between the left hand side and the right hand side. When slack variables are non-negative (meaning that the constraint is of type less or equal), they are called **surplus** variables. Note that normally these constraints represent limited resources, so the slack value will represent the surplus of resource that we have available in the optimal solution. When slack variables are non-positive (meaning that the constraint is of type greater or equal), they are called **excess** variables. Note that these types of constraints will normally represent additional requirements, so the value of the slack variable will represent the excess of the solution with respect to the requirement. When slack variables are zero, they are referred to **artificial variables**. Since they must be zero, they need to be treated differently by the solvers.

### Conversion of negative decision variables

Finally, note that in the standard form, all the decision variables need to be non-negative, we can achieve this with a change of variable:

$$x = x^i - x^{ii}$$

$$x^i, x^{ii} \in [0, \text{inf}]$$

That is, we replace the decision variable  $x$  by two non-negative decision variables, when  $x^{ii}$  is greater than  $x^i$ ,  $x$  takes negative values.

### Summary

Now, in order to the standard form to hold, we need to plug in the new variables into the model. We first introduce the slack variables into our decision variable vector, so that it becomes:

$$x = [x_1, x_2, \dots, x_n, s_1, s_2, \dots, s_n]$$

Now, to make sure that there is no impact in the objective function, we multiply the non-zero slack variables by a factor of 0. Since artificial variables are equal to zero, we multiply them by an arbitrarily large negative number  $-M$ . This number needs to be introduced by algorithms such as the Simplex. Let us assume without loss of generality that we have  $p$  non-zero slack variables and  $k$  artificial variables. Let us note artificial variables as  $a_1, a_2, \dots, a_k$ . Our objective function becomes:

$$\max z = \sum_{i=1}^n c_i \cdot x_i + 0 \cdot s_1 + 0 \cdot s_2 + \dots + 0 \cdot s_p - M \cdot a_1 - M \cdot a_2 + \dots - M \cdot a_k$$

And the constraints:

$$a_{11} \cdot x_1 + a_{12} \cdot x_2 + \dots + a_{1n} \cdot x_n + s_1 + 0 \cdot s_2 + \dots + 0 \cdot s_p + 0 \cdot a_1 + 0 \cdot a_2 + \dots + 0 \cdot a_k = b_1$$

$$a_{21} \cdot x_1 + a_{22} \cdot x_2 + \dots + a_{2n} \cdot x_n + 0 \cdot s_1 + 0 \cdot s_2 + \dots + 0 \cdot s_p + 0 \cdot a_1 + 0 \cdot a_2 + \dots + 0 \cdot a_k = b_2$$

...

$$a_{p1} \cdot x_1 + a_{p2} \cdot x_2 + \dots + a_{pn} \cdot x_n + 0 \cdot s_1 + 0 \cdot s_2 + \dots + s_p + 0 \cdot a_1 + 0 \cdot a_2 + \dots + 0 \cdot a_k = b_p$$

$$a_{(p+1)1} \cdot x_1 + a_{(p+1)2} \cdot x_2 + \dots + a_n \cdot x_n + 0 \cdot s_1 + 0 \cdot s_2 + \dots + 0 \cdot s_p + a_1 + 0 \cdot a_2 + \dots + 0 \cdot a_k = b_{p+1}$$

$$a_{(p+2)1} \cdot x_1 + a_{(p+2)2} \cdot x_2 + \dots + a_{(p+2)n} \cdot x_n + 0 \cdot s_1 + 0 \cdot s_2 + \dots + 0 \cdot s_p + 0 \cdot a_1 + a_2 + \dots + 0 \cdot a_k = b_{p+2}$$

...

$$a_{m1} \cdot x_1 + a_{m2} \cdot x_2 + \dots + a_{mn} \cdot x_n + 0 \cdot s_1 + 0 \cdot s_2 + \dots + 0 \cdot s_p + 0 \cdot a_1 + 0 \cdot a_2 + \dots + 1 \cdot a_k = b_m$$

### Standard tabular representation

Note that we can express the standard form as stated at the beginning of this section as:

$$\max z = c \cdot x$$

s.t.

$$A \cdot x = b$$

where

$$x = [x_1, x_2, \dots, x_n, s_1, s_2, \dots, s_p, a_1, a_2, \dots, a_k]^T$$

$$c = [c_1, c_2, \dots, c_n, 0, \dots, 0, -M, \dots, -M]^T$$

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} & 1 & 0 & \dots & 0 \\ a_{21} & a_{22} & \dots & a_{2n} & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} & 0 & 0 & \dots & 1 \end{bmatrix}$$

$x$  and  $c$  are two  $n + m$  vectors and  $A$  is a  $m \times (n+m)$  matrix where the right most  $n$  columns represent the original LHS coefficients and the right most  $n$  columns represent the  $n \times n$  identity matrix.

Now, if we move all the dependent terms to the LHS, the objective function becomes:

$$z - c_1 * x_1 - c_2 * x_2 - \dots - c_n * x_n - 0 * s_1 - 0 * s_2 + \dots - 0 * s_p + M * a_1 + M * a_2 + \dots + M * a_k = 0$$

We can write all the coefficients in a data frame where each column represents the coefficients of a variable and each row represents an equation:

z	$x_1$	$x_2$	...	$x_n$	$s_1$	$s_2$	...	$s_p$	$a_1$	$a_2$	...	$a_k$	RHS
1	$-c_1$	$-c_2$	...	$-c_n$	0	0	...	0	-M	-M	...	-M	0
0	$a_{11}$	$a_{12}$	...	$a_{1n}$	1	0	0	...	0	0	...	0	$b_1$
0	$a_{21}$	$a_{22}$	...	$a_{2n}$	0	1	0	...	0	0	...	0	$b_2$
...	...	...	...	...	...	...	...	...	...	...	...	...	...
0	$a_{m1}$	$a_{m2}$	...	$a_{mn}$	0	0	0	...	0	0	...	1	$b_m$

This representation of the problem is called the standard tabular form.

## 2.1.2 CLP Graphic solution example

Try me



## Introduction

In this notebook we analyse the graphic method to solve CLP problems of two variables. First, we will derive a CLP model of our problem, and then, we will use the Python plotting library matplotlib to obtain a graphical representation that we will use to find the optimal solution.

The graphic method is not really a practical method to solve CLP problems, but rather, a quite didactic mean to gain understanding on the CLP. Later in the course, we will cover the Simplex algorithm, one of the most successful algorithms in history, which is an extension of this method. To put it into practice, we will use a simple blending problem defined below.

## Problem Definition

The cost per gram of  $M_1$  is 45 €cents

The cost per gram  $M_2$  is 12 €cents

The final product has two nutritional properties that depend on  $M_1$  and  $M_2$ , the non-saturated fat and the proteins.

To meet the quality requirements, the final product must have no more than 0.5 percent of non-saturated fat and at least 8 percent proteins.

The recip must have at least 2% of  $M_2$ .

$M_1$  is 3 percent non-saturated fat and 80 percent protein.

$M_2$  is 3 percent non-saturated fat and 20 percent protein.

**> How many grams of M1 and M2 should be used in every 100 grams of product?**

## Model

The problem can be modeled as:

$$\min Z = 45x_1 + 12x_2$$

where:

$x_1$  : grams of M1 per 100 grams of product

$x_2$  : grams of M2 per 100 grams of product

The objective function is subject to the following constraints:

$$x_1 > 0$$

$$x_2 \geq 2$$

$$x_1 + x_2 \leq 100$$

$$3x_1 + 3x_2 \leq 50$$

$$8x_1 + 2x_2 \geq 80$$

### Solution

We will use the [numpy](#) library to create the decision space and the [matplotlib](#) library to represent the decision space. Numpy provides useful functions to create multidimensional arrays. Below, some tips of the functions we are going to use.

### Numpy Functions

The following subsections describe some functions of the [Numpy](#) Python package that are used in this example. The [Numpy Tutorial](#) includes comprehensive information about this package, but as a reminder and for the sake of completeness, here you can find a brief description of the functions used in this tutorial.

#### Linspace

We will use the function `linspace` to create the x-axis that represents the  $x_1$  decision variable. Linspace receives as arguments the minimum and maximum values of the axis and the number of elements that we want in our axis.

#### maximum and minimum

We will use the functions `maximum` and `minimum` to obtain the lines that delimit the feasibility region. `Minimum` and `maximum` take two (multidimensional) arrays and provide an array with the element-wise minimum or maximum at each position.

### Pyplot Functions

Likewise, the following subsections describe some functions of the [Matplotlib](#) Python package that we have used. The [Matplotlib Tutorial](#) provides more information and sample codes of the library, but here is a brief guide of the main functions used in this tutorial.

#### plot

We will use the function `plot` to draw lines. `Plot` takes two arrays as parameters and uses the values of each position of the array as coordinates. We can add additional parameters as described [here](#). We will use one parameter to represent the labels.

#### fill\_between

We will use the function `fill_between` to fill the area between two lines. Since we can only use two lines to call `fill`, we will need to use the numpy functions `maximum` and `minimum` to create the lines that we want to draw

#### xlim, ylim

We will use these functions to set the limits of both coordinates

ylabel, xlabel, legend

We will use these functions to add labels to the different axis and a legend to the figure

### Feasibility region

Let us determine the feasibility region by iteratively representing the different constraints in the decision space. We start by representing the possible values of  $x_1$  and  $x_2$  just taking into account the physical constraints:

$$x_1 > 0$$

$$x_2 > 0$$

$$x_1 + x_2 \leq 100$$

```
[1]: #Import the numpy and pyplot libraries, we set the aliases np and plt so that it is
      ↪easier to use
import numpy as np
import matplotlib.pyplot as plt

#We set the mode inline of matplotlib to get the result at the output of the cell code
%matplotlib inline

# Construct lines, in our coordinate system x represents our decision variable x1 and
  ↪y represents our decision variable x2
# x > 0, we use the function linspace to create a linear space for x
x = np.linspace(0, 100, 2000) #2000 numbers from 0 to 100
# y1 represents the line of constraint y <= 100 - x
y1 = 100 - x

#y2 represents the line of constraint y > 0 (we need this function to call fill
  ↪between)
y2 = x*0

#1. Make plot
plt.plot(x, y1, label=r'$x_{1}+x_{2} \leq 100$')

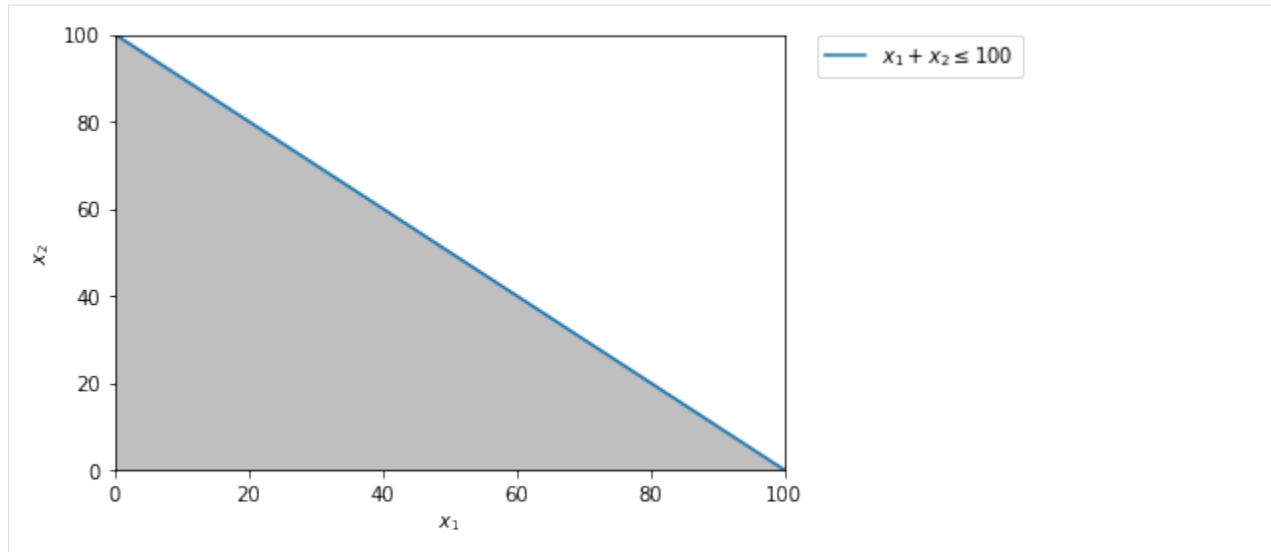
#2. Adjust axis
plt.xlim((0, 100))
plt.ylim((0, 100))
plt.xlabel(r'$x_{1}$')
plt.ylabel(r'$x_{2}$')

#3. Fill feasible region

plt.fill_between(x, y1, y2, where=y1>y2, color='grey', alpha=0.5) #fill where y1 is
  ↪greater than y2

#4. plot legend
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)

[1]: <matplotlib.legend.Legend at 0x1fd6b115460>
```



Introducing constraints

Now we enter the first constraint:

$$x_1 > 0$$

$$x_2 \geq 2$$

$$x_1 + x_2 \leq 100$$

We can see how this affects the feasibility area by representing again the equations

```
[2]: import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# Construct lines, x represents our decision variable x1 and y represents our
↳ decision variable x2
# x > 0
x = np.linspace(0, 100, 2000) #2000 numbers from 0 to 100
# y2 represents the line of constraint y ≤ 100 - x
y1 = 100 - x
y2 = x*0+2;

#1. Make plot
plt.plot(x, y1, label=r'$x_{1}+x_{2} \leq 100$')
plt.plot(x, y2, label=r'$x_{2} \geq 2$')

#2. Adjust axis
plt.xlim((0, 100))
plt.ylim((0, 100))
plt.xlabel(r'$x_{1}$')
plt.ylabel(r'$x_{2}$')

#3. Fill feasible region

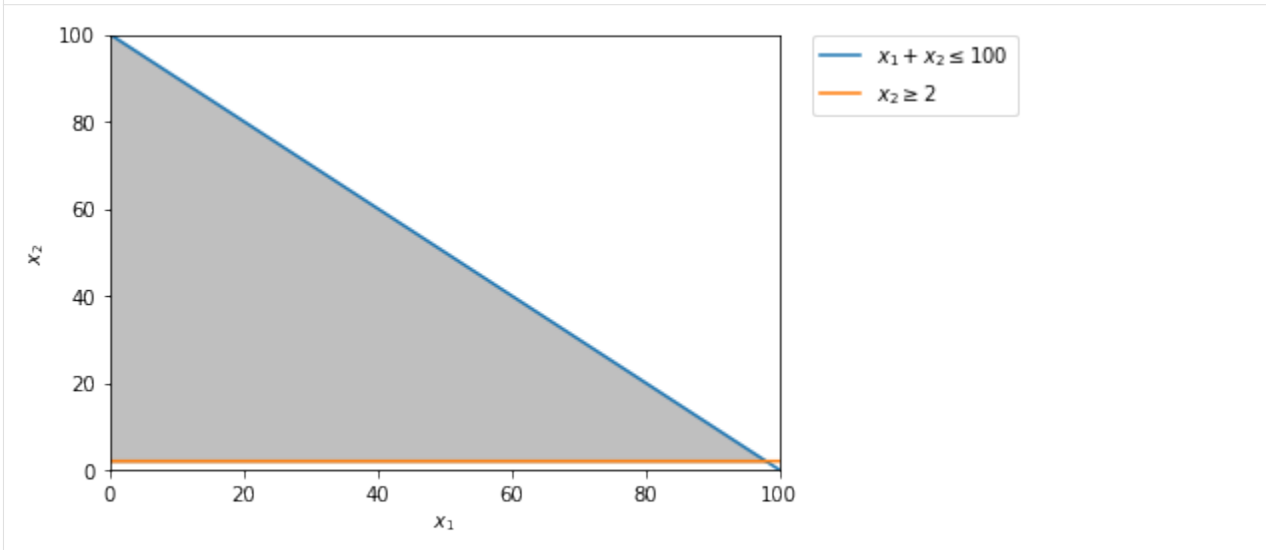
plt.fill_between(x, y1, y2, where=y1>y2, color='grey', alpha=0.5) #fill where y1 is
↳ greater than y2
```

(continues on next page)

(continued from previous page)

```
#4. plot legend
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
```

```
[2]: <matplotlib.legend.Legend at 0x1fd6c1628e0>
```



Now we enter the next constraint:

$$x_1 > 0$$

$$x_2 \geq 2$$

$$x_1 + x_2 \leq 100$$

$$3x_1 + 3x_2 \leq 50$$

Again, we can check the effect on the feasibility region

```
[3]: import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# Construct lines, x represents our decision variable x1 and y represents our_
↳ decision variable x2
# x > 0
x = np.linspace(0, 100, 2000) #2000 numbers from 0 to 100
# y2 represents the line of constraint y <= 100 - x
y1 = 100 - x
y2 = x*0+2;
y3 = (50-3*x)/3.0

#1. Make plot
plt.plot(x, y1, label=r'$x_{1}+x_{2} \leq 100$')
plt.plot(x, y2, label=r'$x_{2} \geq 2$')
plt.plot(x, y3, label=r'$3x_{1} + 3x_{2} \leq 50$')

#2. Adjust axis
plt.xlim((0, 100))
plt.ylim((0, 100))
plt.xlabel(r'$x_{1}$')
```

(continues on next page)

(continued from previous page)

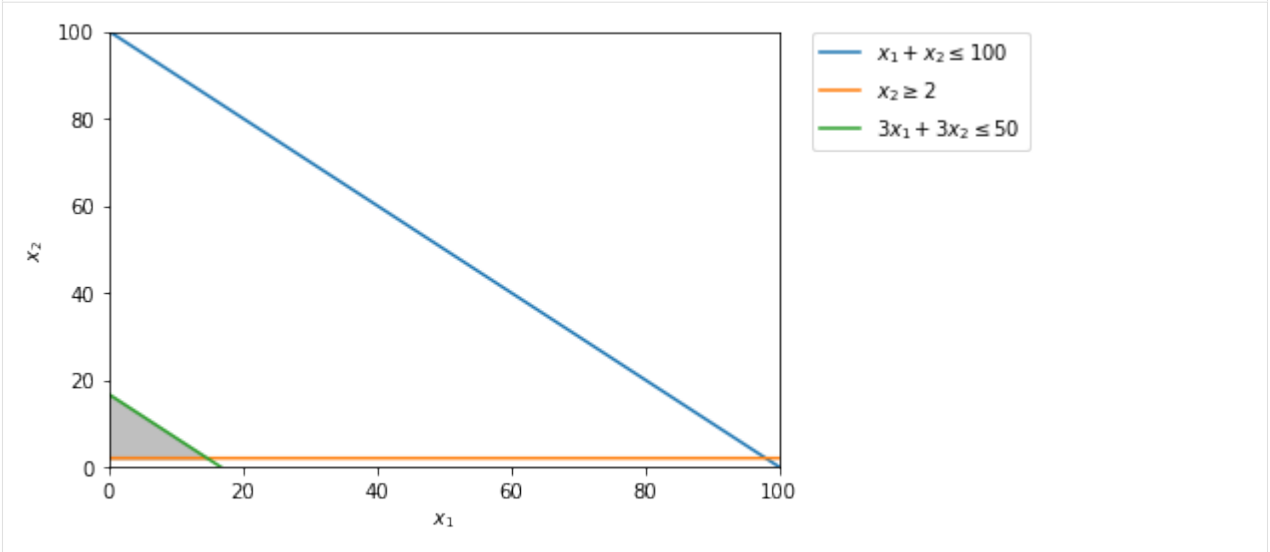
```
plt.ylabel(r'$x_{2}$')

#3. Fill feasible region

plt.fill_between(x, y3, y2, where=y3>y2, color='grey', alpha=0.5) #fill where y3 is_
↳greater than y2

#4. plot legend
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
```

[3]: <matplotlib.legend.Legend at 0x1fd7e444eb0>



We can see that now the physical constraint  $x_1 + x_2 \leq 100$  does not have any impact on the feasibility region. We can focus on narrower intervals of  $x_1$  and  $x_2$  to search for the best solution. Let's also introduce the next constraint.

- $x_1 > 0$
- $x_2 \geq 2$
- $x_1 + x_2 \leq 100$
- $3x_1 + 3x_2 \leq 50$
- $8x_1 + 2x_2 \geq 80$

Now the feasibility region is:

```
[4]: import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# Construct lines, x represents our decision variable x1 and y represents our_
↳decision variable x2
# x > 0
x = np.linspace(0, 100, 2000) #2000 numbers from 0 to 100

# y1 represents the physical constraint x + y <= 100
y1 = 100 - x

# y2 represents the line of constraint y >= 2
```

(continues on next page)



(continued from previous page)

```

y2 = x*0+2

# y3 represents the constraint 3x + 3y <= 50
y3 = (50-3*x)/3.0

# y4 represents the line of the constraint 6x + 2y >= 4
y4 = (80 - 8*x)/2.0

#1. Make plot
plt.plot(x, y1, label=r'$x_{1}+x_{2} \leq 100$') #y4
plt.plot(x, y2, label=r'$x_{2} \geq 2$') #y1
plt.plot(x, y3, label=r'$3x_{1} + 3x_{2} \leq 50$') #y2
plt.plot(x, y4, label=r'$8x_{2} + 2x_{1} \geq 80$') #y3

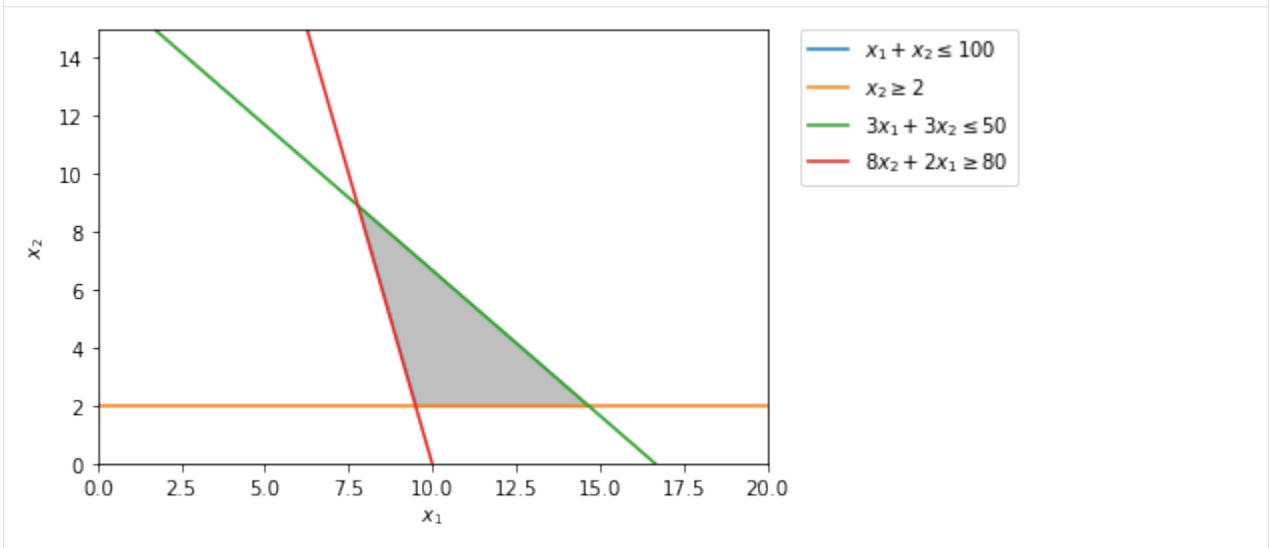
#2. Adjust axis
plt.xlim((0, 20))
plt.ylim((0, 15))
plt.xlabel(r'$x_{1}$')
plt.ylabel(r'$x_{2}$')

#3. Fill feasible region
y5=np.maximum(y4, y3) #line representing the maximum between y4 and y3
y6=np.maximum(y2,y4) #line representing the maximum between y2 and y4
plt.fill_between(x, y5, y6, where=y5>y6, color='grey', alpha=0.5) #fill where y5 is
    ↪ greater than y6

#4. plot legend
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)

```

[4]: <matplotlib.legend.Legend at 0x1fd6c15d100>



The feasible region (in grey) contains all possible solutions to the objective function.

Minima and maxima of linear programming problems are found at the vertices of the feasible region. Therefore, the solution is the minimum of the objective function between the three vertices.

The three corners are the intersections between the lines:

Line 1	Line 2
$3x + 3y \leq 50$	$8x + 2y \geq 80$
$8x + 2y \geq 80$	$y \geq 2$
$y \geq 2$	$3x + 3y \leq 50$

Considering that the objective function is:

$$Z = 45x + 12y$$

We can calculate Z for each corner:

1)  $3x + 3y \leq 50$  and  $8x + 2y \geq 80$

$$(50 - 3x)/3 = (80 - 8x)/2$$

$$x = 7.78$$

$$y = 8.86$$

$$Z = 456.42$$

2)  $8x + 2y \geq 80$  and  $y \geq 2$

$$(80 - 8x)/2 = 2$$

$$x = 9.5$$

$$y = 2$$

$$Z = 451.5$$

3)  $y \geq 2 - x$  and  $3x + 3y \leq 50$

$$2 = (50 - 3x)/3$$

$$x = 14.67$$

$$y = 2$$

$$Z = 684$$

```
[5]: import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# Construct lines, x represents our decision variable x1 and y represents our
↳ decision variable x2
# x > 0
x = np.linspace(0, 100, 2000) #2000 numbers from 0 to 100

# y1 represents the physical constraint x + y <= 100
y1 = 100 - x

# y2 represents the line of constraint y >= 2
y2 = x*0+2

# y3 represents the constraint 3x + 3y <= 50
y3 = (50-3*x)/3.0

# y4 represents the line of the constraint 6x + 2y >= 4
y4 = (80 - 8*x)/2.0

y6 = (451.5 - 45*x)/12

#1. Make plot
plt.plot(x, y1, label=r'$x_{1}+x_{2} \leq 100$') #y4
```

(continues on next page)

(continued from previous page)

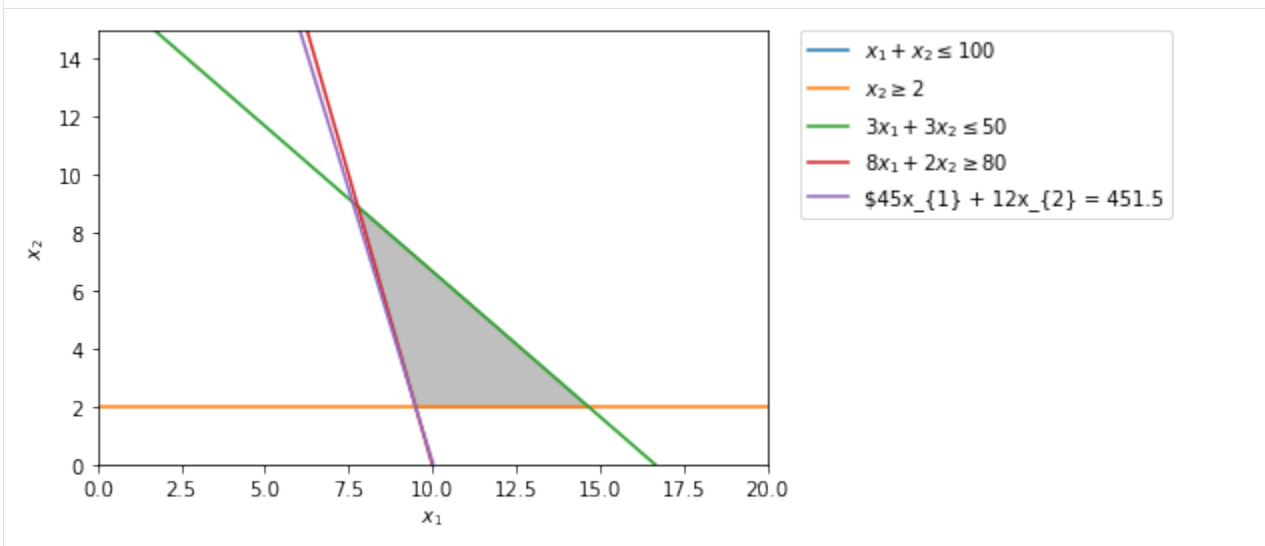
```
plt.plot(x, y2, label=r'$x_{2} \geq 2$') #y1
plt.plot(x, y3, label=r'$3x_{1} + 3x_{2} \leq 50$') #y2
plt.plot(x, y4, label=r'$8x_{1} + 2x_{2} \geq 80$') #y3
plt.plot(x, y6, label=r'$45x_{1} + 12x_{2} = 451.5$')

#2. Adjust axis
plt.xlim((0, 20))
plt.ylim((0, 15))
plt.xlabel(r'$x_{1}$')
plt.ylabel(r'$x_{2}$')

#3. Fill feasible region
y5=np.maximum(y4, y3) #line representing the maximum between y4 and y3
y6=np.maximum(y2,y4) #line representing the maximum between y2 and y4
plt.fill_between(x, y5, y6, where=y5>y6, color='grey', alpha=0.5) #fill where y5 is
->greater than y6

#4. plot legend
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
```

[5]: <matplotlib.legend.Legend at 0x1fd7e5474f0>



We have successfully calculated that the minimum value for Z (451.5), when  $x_1$  is 9.5 and  $x_2$  is 2.

This method of testing every vertex is only feasible for a small number of variables and constraints. As the numbers of constraints and variables increase, it becomes far more difficult to graph these problems and work out all the vertices. For example, if there were a third variable:

$$Z = Ax + By + Cz$$

We would have to graph in three dimensions (x, y and z).

In the next few notebooks, we'll take a look at how we can use the graphical method to solve other CLP problems:

- Making Chappie solved with the graphical method
- Making weapons solved with the graphical method

### 2.1.3 Interactive Graphic Method

Try me



#### Introduction

In this notebook, you can experiment with the graphic representation of a linear programming problem. The objective is to gain understanding on the graphic method, how constraints affect the boundaries of the feasibility region and why the solution to a problem is always at a vertex of the feasibility region. In this notebook you will be able to change the objective variable and add or remove constraints to experiment and reflect on these questions.

#### Requirements

This script uses [Numpy](#), [Matplotlib](#), and [Ipywidgets](#). They are provided off-the-shelf in Colabs. Otherwise, run the following cell to install in your Kernel:

```
[ ]: !pip install numpy
      !pip install matplotlib
      !pip install ipywidgets
```

#### Problem model

The problem model is taken from the [Making weapons](#) problem. You can find the solution here: ([Making weapons](#))

$$\max z = 40x_1 + 5x_2$$

s.t.

$$x_1 + 0.5x_2 \leq 125 \text{ (Beskar constraint)}$$

$$x_1 + x_2 \leq 225 \text{ (Tungsten constraint)}$$

$$3x_1 + x_2 \leq 300 \text{ (Titanium constraint)}$$

$$x_2 \geq 100 \text{ (Demand constraint)}$$

The code cell below allows you to add the constraints one by one to see how to modify the feasibility region. The slider allows to modify the objective variable  $z$  to find the optimal value in each case. Remember, the optimal value will be always the highest value of  $z$  where the objective function intersects with the feasibility region.

#### Solution with the graphical method

```
[3]: #Import the numpy and pyplot libraries, we set the aliases np and plt so that it is_
      ↪easier to use
      import numpy as np
      import matplotlib.pyplot as plt
      import ipywidgets as widgets
      from ipywidgets import interactive
      #We set the mode inline of matplotlib to get the result at the output of the cell code
      %matplotlib inline

      #Construct lines, in our coordinate system x represents our decision variable x1 and_
```

(continues on next page)

(continued from previous page)

```

↪y represents our decision variable x2
x=np.linspace(0,300,2000) #2000 numbers from 0 to 100

y1=(125-x)/0.5 # Beskar constraint
y3=225-x      # Tungsten constraint
y4=100+x*0    # Demand constraint
y5=300-3*x    # Titanium constraint

#1. Interactive plot
def show_solution(z, beskar_constraint, tungsten_constraint, titanium_constraint, ↵
↪demand_constraint):
    y_min=x*0      # absolute min: non negativity constraint
    y_max = 250 + 0*x # A very high maximum value
    plt.figure(1) #Create a figure
    # 1. Check if we need to add constraints and recalculate max and min
    if beskar_constraint:
        plt.plot(x,y1,label=r'$x_{1}+0.5x_{2}\leq125$')
        y_max = np.minimum(y1, y_max)

    if tungsten_constraint:
        plt.plot(x,y3,label=r'$x_{1}+x_{2}\leq225$')
        y_max = np.minimum(y3, y_max)

    if titanium_constraint:
        plt.plot(x,y5,label=r'$3x_{1}+x_{2}\leq300$')
        y_max = np.minimum(y5, y_max)

    if demand_constraint:
        plt.plot(x,y4,label=r'$x_{2}\geq100$')
        y_min = y4

    # 2. Adjust axis
    plt.xlim((0,150))
    plt.ylim((0,250))
    plt.xlabel((r'$x_{1}$'))
    plt.ylabel((r'$x_{2}$'))

    # 3. fill between max and min
    plt.fill_between(x, y_max, y_min, where=y_max>y_min, color='grey', alpha=0.5)

    # 4. Plot objective function
    obj_func= (z - 40*x)/5 # we clear the x2 with the value of z
    plt.plot(x, obj_func, label=r'$z = 40*x_1 + 5*x_2$')

    # 5. Plot legend
    plt.legend(bbox_to_anchor=(1.05,1), loc=2, borderaxespad=0.)
    plt.show()

interactive_plot = interactive(show_solution, z=(0,6000,10), beskar_constraint = ↵
↪False, tungsten_constraint = False, titanium_constraint = False, demand_constraint ↵
↪= False)
output = interactive_plot.children[-1]
output.layout.height = '350px'
interactive_plot

```

```
interactive(children=(IntSlider(value=3000, description='z', max=6000, step=10), ↵  
↵Checkbox(value=False, descrip..
```

### 2.1.4 Simplex example

Try me



#### Graphic and algebraic correspondences

In this notebook we will analyse the Simplex algorithm alternating between the graphical and the algebraic representation and using a problem as an example:

##### Problem Definition

Production Mix A company produces two kinds of products, P1 and P2.

The selling price of P1 is 300€ The selling price of P2 is 250€

- P1 requires 2 operator man hours and 1 hour of machine operating time
- P2 requires 1 operator man hours and 3 hours of machine time
- Operator man hours cannot be higher than 40 hours per day
- Machine operating time cannot be higher than 45 hours per day
- Marketing states that P1 units per day cannot be higher than 12

**How many units of P1 and P2 should the company produce every day to maximise income?**

##### Model

We want to find the maximum solution to:

$$\max Z = 300x_1 + 250x_2$$

This is our objective function. The decision variables are:

$x_1$  : units of Product 1

$x_2$  : units of Product 2

The objective function is subject to the following constraints:

$$2x_1 + x_2 \leq 40$$

$$x_1 + 3x_2 \leq 45$$

$$x_1 \leq 12$$

We define the problem in its standard form as:

$$\max Z = 300x_1 + 250x_2 + 0s_1 + 0s_2 + 0s_3$$

Where:

$s_1$  : Slack variable of the operator man hour constraint ( $s_1 \geq 0$ )

$s_2$  : Slack variable of the machining time constraint ( $s_2 \geq 0$ )

$s_3$  : Slack variable of the operator man hour constraint ( $s_3 \geq 0$ )

And thus, by entering the slack variables, we can now express the constraints as:

$$2x_1 + x_2 + s_1 = 40$$

$$x_1 + 3x_2 + s_2 = 45$$

$$x_1 + s_3 = 12$$

## Solution

We will use the Simplex Method and iterate between the graphical and the algebraic form of the problem to describe the results. First let us represent the problem:

```
[1]: # Alias to use numpy library functions
import numpy as np
import matplotlib
# Alias to use matplotlib
import matplotlib.pyplot as plt
# Inline is used to get the output of matplotlib to the output of the cell
%matplotlib inline

# Construct lines, x represents our decision variable x1 and y represents our
↳decision variable x2
# x > 0
x = np.linspace(0, 100, 2000) #2000 numbers from 0 to 100
# y1 represents the line of constraint 2x1 + x2 <= 40
y1 = -2*x + 40
# y2 represents the line of the constraint x1 + 3x2 <= 45
y2 = (45-x)/3.0
# x2 represents a constraint at y2 <12 used for convenience
x2 = (12 - x*0)

matplotlib.rcParams.update({'font.size': 18})

#1. Make plot
plt.plot(x, y1, label=r'$2x_{1} + x_{2} \leq 40$') #Plot y1
plt.plot(x, y2, label=r'$x_{1} + 3x_{2} \leq 45$') #Plot y2
plt.axvline(x=12, label=r'$x_{1} \leq 12$', c='g') #plot y3

#2. Adjust axis
plt.xlim((0, 50))
plt.ylim((0, 45))
plt.xlabel(r'$x_{1}$')
plt.ylabel(r'$x_{2}$')

#3. Fill feasible region
y4=np.minimum(y1, y2) #line representing the maximum between y3 and y2
plt.fill_between(x, y4, 0, where=x<12, color='grey', alpha=0.5) #fill where y5 ys
↳greater than y6
```

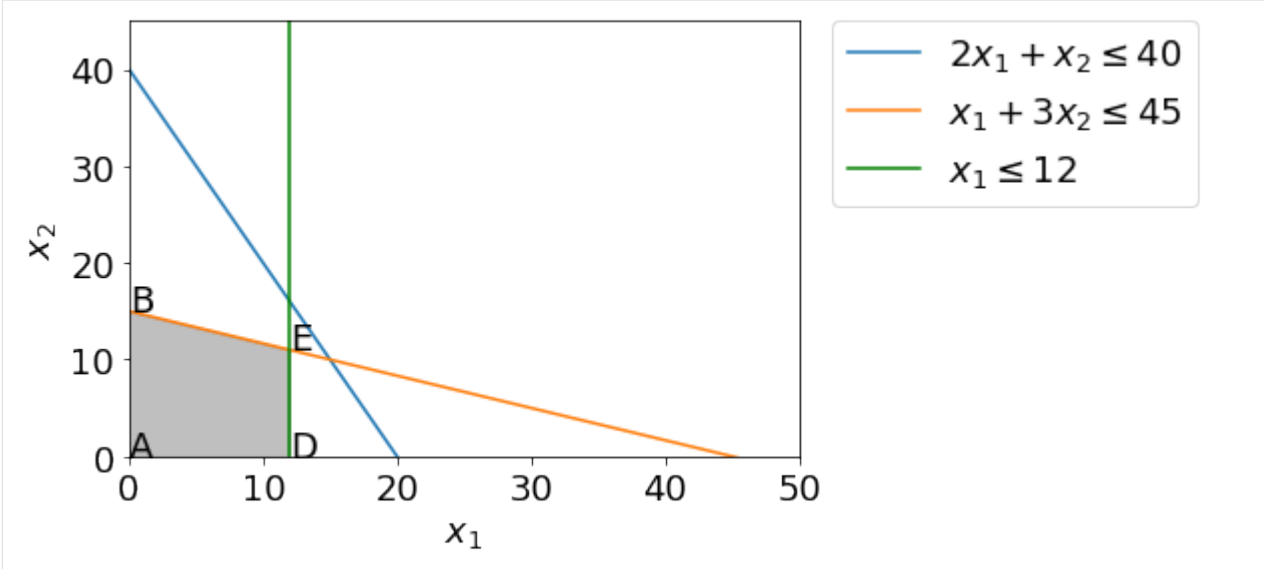
(continues on next page)

(continued from previous page)

```
#4. plot legend
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)

#5 Annotate graph
plt.annotate('A', xy=(0, 0))
plt.annotate('B', xy=(0, 15))
plt.annotate('D',xy=(12, 0))
plt.annotate('E', xy=(12,11))
```

[1]: Text (12, 11, 'E')



The feasible region (in grey) contains all possible solutions to the objective function.

Let us now represent the same problem in a basic Tableau:

Z	$x_1$	$x_2$	$s_1$	$s_2$	$s_3$	RHS
1	-300	-250	0	0	0	0
0	2	1	1	0	0	40
0	1	3	0	1	0	45
0	1	0	0	0	1	12

From the Tableau, we can easily find a first Feasible Basic Solution (FSB), by making  $x_1$  and  $x_2$  non basic (=0). The three equations of the constraints provide us the value of the slack variables in this solution:

$$\begin{aligned} x_1 &= 0 \\ x_2 &= 0 \\ s_1 &= 40 \\ s_2 &= 45 \\ s_3 &= 12 \end{aligned}$$

Let us note this solution as (0,0,40,45,12) and  $z=0$ . Note that this FSB corresponds to Vertex A in the figure. How can we know if it is optimal? We have 2 non basic variables  $x_1$  and  $x_2$  which have negative coefficients in the objective function and positive values of this variables would increase  $z$ . If we increase  $x_1$  in one unit,  $z$  would increase 300 units and if we increase  $x_2$  in one unit  $z$  would increase 250. Therefore, the solution is not optimal, since we could find a higher  $z$  for a positive value of these variables in the objective function. Hence, this is not the optimal solution and we need to look



for it in another vertex. From point A, we could either move to vertex B or to vertex D. Moving towards D would mean making  $x_1$  higher than zero, thus making it a **basic variable**. Likewise, moving towards B would mean making  $x_2$  higher than zero and basic (enter the base). We select  $x_1$  as the **entering** basic variable, since its coefficient (300) is higher and therefore  $z$  will increase at a higher rate when we move towards D. We have just selected the pivot variable and the pivot column in the Tableau.

Now, we are going to move towards D, which is equivalent to make  $x_1 = \delta$ . As this value increases, the objective value will also increase, but we have to bear in mind that all variables must be positive and therefore  $\delta$  cannot take any given value: we have to make sure the solution is **feasible**. If we write down all the constraints in algebraic form, we can find the possible values of  $\delta$

$$\begin{aligned} 2 * \delta + 1 * 0 + s_1 &= 40 \\ 1 * \delta + 3 * 0 + s_2 &= 45 \\ 1 * \delta + s_3 &= 12 \end{aligned}$$

To ensure that all the surplus variables are positive  $\delta$  cannot exceed the maximum positive value that guarantees that all the surplus variables are non negative. Our strategy to find a feasible solution is to make on slack variable **non-basic** (equal to zero), but making sure that all the constraints are met. Since we only have  $\delta$  and a slack variable in each expression, making the corresponding slack variable equal to zero is equivalent to making  $\delta$  equal to the RHS divided by the coefficient, which means that the maximum value it can take is  $\delta = \min(40/2, 45/1, 12/1)$ . Any value above this makes the solution unfeasible. This is called the **Ratio test**. If we make  $x_1 = \delta = 12$  then that implies that  $s_3$  is now a non-basic variable. With the Ratio test, we have selected  $s_3$  as the **leaving** variable which will no longer be basic in the FSB. Let us add two new columns to the basic Tableau, one to the right to indicate which variable is the basic variable in the equation and another one to the left to express this Ratio test.

Basic	z	$x_1$	$x_2$	$s_1$	$s_2$	$s_3$	RHS	Ratio
.	1	-300	-250	0	0	0	0	.
$s_1$	0	2	1	1	0	0	40	40/2=20
$s_2$	0	1	3	0	1	0	45	45/1=45
$s_3$	0	1	0	0	0	1	12	12/1=12

Now, notice that from the Ratio test, we have easily determined that the variable that leaves the basic solution is  $s_3$  and using the last equation we can easily determine that  $x_1 = 12$  in the new solution. We will move to another solution (12,0,?,?,0) which corresponds exactly to point D. Now notice that in the first iteration, the problem was expressed in a very convenient way since we had only a non-zero (1) value for each basic variable in every row. To accomplish this, we are going to transform the different rows. Since the equations are linearly independent, we can make linear operations in the different equations and still obtain a linear independent system of equations. We want to obtain a system like this:

Basic	z	$x_1$	$x_2$	$s_1$	$s_2$	$s_3$	RHS	Ratio
.	1	0	?	0	0	?	?	.
$s_1$	0	0	?	1	0	?	?	?
$s_2$	0	0	?	0	1	?	?	?
$x_1$	0	1	?	0	0	?	?	?

Note that when the problem is in this form, we will have each basic variable in a row with coefficient 1 and the rest with coefficient 0 and we will be able to obtain the value for each basic variable rather easily. First we transform the **leaving** equation, by dividing the entire row by the coefficient of the entering variable in that equation (in this case is 1) and the row remains unaltered:

Basic	z	$x_1$	$x_2$	$s_1$	$s_2$	$s_3$	RHS	Ratio
•	1	0	?	0	0	?	?	•
$s_1$	0	0	?	1	0	?	?	?
$s_2$	0	0	?	0	1	?	?	?
$x_1$	0	1/1=1	0/1=0	0/1=0	0/1=0	1/1=1	12/1=12	?

Now, for the rest of the row, we need to make a linear operation that gives a zero in the cell corresponding to the entering variable. To accomplish this, we subtract the **entering equation** multiplied by the value of the coefficient that multiplies the entering variable  $x_1$ .

Basic	z	$x_1$	$x_2$	$s_1$	$s_2$	$s_3$	RHS	Ratio
•	1	$-300-(-300 \cdot 1)$	$-250-(-300 \cdot 0)$	$0-(-300 \cdot 0)=0$	$0-(-300 \cdot 0)=0$	$0-(-300 \cdot 1)=300$	$0-(-300 \cdot 12)$	•
$s_1$	0	$2-2 \cdot 1=0$	$1-2 \cdot 0=1$	$1-2 \cdot 0=1$	$0-2 \cdot 0=0$	$0-2 \cdot 1=-2$	$40-12 \cdot 2=16$	?
$s_2$	0	$1-1 \cdot 1=0$	$3-1 \cdot 0=3$	$0-1 \cdot 0=0$	$1-1 \cdot 0=1$	$0-1 \cdot 1=-1$	$45-12 \cdot 1=33$	?
$x_1$	0	1	0	0	0	1	12	?

We end up with the following Tableau:

Basic	z	$x_1$	$x_2$	$s_1$	$s_2$	$s_3$	RHS	Ratio
•	1	0	-250	0	0	300	3600	•
$s_1$	0	0	1	1	0	-2	16	?
$s_2$	0	0	3	0	1	-1	33	?
$x_1$	0	1	0	0	0	1	12	?

And the new FSB solution is (12, 0, 16, 32, 0) which as noted corresponds to point D.

At this point, z has increased from 0 to 3600, but we still have a coefficient in  $x_2$  that is lower than zero and therefore we could find a higher value if again we move from D to E by increasing  $x_2$ , just as we did with  $x_1$ . We have selected the next entering variable and the pivot column and we can perform the Ratio test again to select the variable leaving the basic solution:

Basic	z	$x_1$	$x_2$	$s_1$	$s_2$	$s_3$	RHS	Ratio
•	1	0	-250	0	0	300	3600	•
$s_1$	0	0	1	1	0	-2	16	16
$s_2$	0	0	3	0	1	-1	33	11
$x_1$	0	1	0	0	0	1	12	12

The minimum value that makes all variables positive is 11. Therefore, the variable leaving the solution is  $s_2$ . We first transform the new entering row:

Basic	z	$x_1$	$x_2$	$s_1$	$s_2$	$s_3$	RHS	Ratio
.	1	?	0	0	?	?	?	.
$s_1$	0	0	0	1	?	?	?	?
$x_2$	0	0	1	0	1/3	-1/3	11	?
$x_1$	0	1	0	0	?	?	?	?

And then the rest of the rows using linear transformations as we did before we obtain.

Basic	z	$x_1$	$x_2$	$s_1$	$s_2$	$s_3$	RHS	Ratio
.	1	$0 - (-250 \cdot 0) = -250$	$0 - (-250 \cdot 0) = -250$	$0 - (-250 \cdot 0) = 0$	$0 - (-250 \cdot 1/3) = 300$	$0 - (-250 \cdot -1/3) = 3600$	$0 - (-250 \cdot 0) = 0$	.
$s_1$	0	0	$1 - 1 = 0$	$1 - 0 = 0$	$0 - 1/3 = -1/3$	$-2 - (-1/3) = -5/3$	$16 - 11 = 5$	.
$x_2$	0	0	1	0	1/3	-1/3	11	.
$x_1$	0	1	0	0	0	1	12	.

Now we obtain a new solution (12, 11, 5, 0, 0) where  $z = 6350$  and we know for sure that this is a maximum because there are no negative coefficients for any of the non-basic variables that could yield to a higher value of  $z$ .

```
[2]: # Alias to use numpy library functions
import numpy as np
import matplotlib
# Alias to use matplotlib
import matplotlib.pyplot as plt
# Inline is used to get the output of matplotlib to the output of the cell
%matplotlib inline

# Construct lines, x represents our decision variable x1 and y represents our
↪ decision variable x2
# x > 0
x = np.linspace(0, 100, 2000) #2000 numbers from 0 to 100
# y1 represents the line of constraint 2x1 + x2 <= 40
y1 = -2*x + 40
# y2 represents the line of the constraint x1 + 3x2 <= 45
y2 = (45-x)/3.0

matplotlib.rcParams.update({'font.size': 18})

#1. Make plot
plt.plot(x, y1, label=r'$2x_{1} + x_{2} \leq 40$') #Plot y1
plt.plot(x, y2, label=r'$x_{1} + 3x_{2} \leq 45$') #Plot y2
plt.axvline(x=12, label=r'$x_{1} \leq 12$', c='g') #plot y3

#2. Adjust axis
plt.xlim((0, 50))
plt.ylim((0, 45))
plt.xlabel(r'$x_{1}$')
```

(continues on next page)

```
plt.ylabel(r'$x_{2}$')

#3. Fill feasible region
y4=np.minimum(y1, y2) #line representing the maximum between y3 and y2
plt.fill_between(x, y4, 0, where=x<12, color='grey', alpha=0.5) #fill where y5 ys.
↳greater than y6

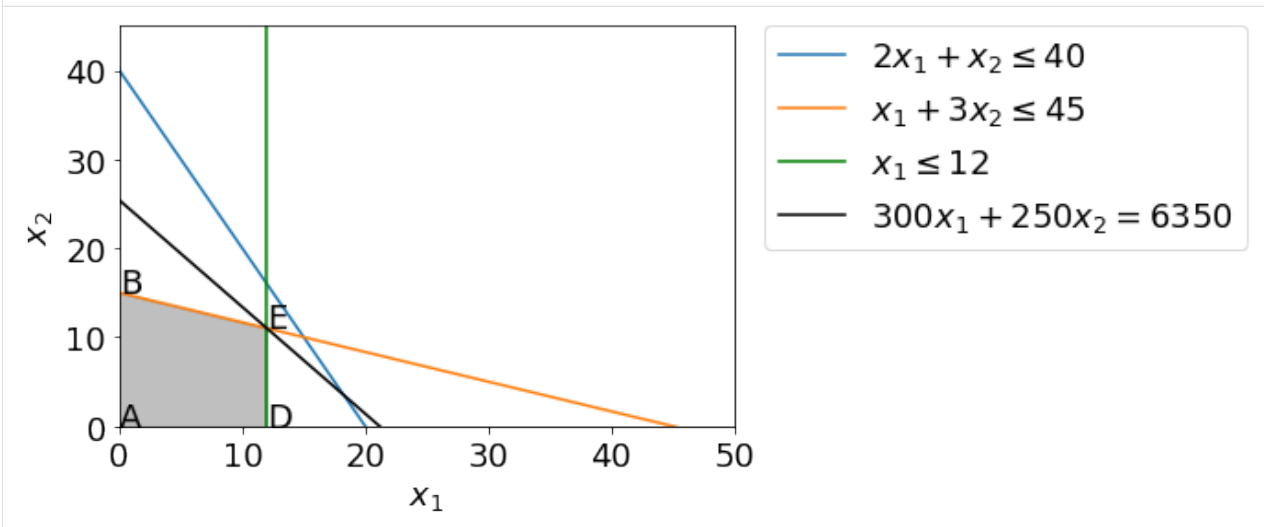
y5 = (6350-300*x)/250 #line representing the solution

plt.plot(x, y5, label=r'$300x_{1} + 250x_{2} = 6350$', c='k')

#4. plot legend
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)

#5 Annotate graph
plt.annotate('A', xy=(0, 0))
plt.annotate('B', xy=(0, 15))
plt.annotate('D',xy=(12, 0))
plt.annotate('E', xy=(12,11))
```

[2]: Text(12, 11, 'E')



## 2.1.5 Duality

### Introduction

The duality principle states that every optimization problem can be represented and analysed in two different formulations, the **primal** and the **dual**. This means that every problem that we have analysed so far has a **dual** problem formulation with the following characteristics:

- The dual has an optimal solution **if and only if** the primal has an optimal solution (and vice versa)
- The **optimal value** of the objective function both cases (respective optimal solutions) **is the same** (strong duality principle)
- The dual of dual is the primal
- The solution of the dual problem can be obtained from the solution of the primal and vice versa (using algorithms like the like Simplex)

### Basic rules for obtaining the dual

The dual of a problem can be obtained applying the following rules:

- If the **primal problem objective function** is of type **maximise**, the **dual problem objective function** is of type **minimise** (and vice versa)
- If the primal has  $m$  constraints, the dual has  $m$  decision variables, noted as  $u_1, u_2, \dots, u_m$
- The coefficients of the objective function of the dual are the independent terms (RHS) of the constraints of the primal (and vice versa)
- The coefficients of the dependent terms (LHS) of the constraint of the dual can be represented by the transposed matrix  $A^T$  that contains the LHS coefficients of the primal (and vice versa)
- All the constraints are of type greater or equal ( $\geq$ )
- For maximization problems:
  - If the  $i^{th}$  constraint of the primal is of type less or equal ( $\leq$ ), then  $u_i$  is non-negative ( $\geq 0$ )
  - If the  $i^{th}$  constraint of the primal is of type greater or equal ( $\geq$ ), then  $u_i$  is non-positive ( $\leq 0$ )
- For minimization problems:
  - If the  $i^{th}$  constraint of the primal is of type less or equal ( $\leq$ ), then  $u_i$  is non-positive ( $\leq 0$ )
  - If the  $i^{th}$  constraint of the primal is of type greater or equal ( $\geq$ ), then  $u_i$  is non-negative ( $\geq 0$ )
- If any case, if the  $i^{th}$  constraint of the primal is of type equal ( $=$ ), then  $u_i$  is unrestricted (can be positive or negative)

### Summary

These rules are summarised in the following table:

Primal problem	Dual problem
Objective of type maximize	Objective of type minimize
Objective of type minimize	Objective of type maximize
Constraint limits, RHS	Decision Variable coefficients
Decision Variable coefficients	Constraint limits, RHS
Constraint coefficient of decision variable $i$ ( $a_{ij}$ )	Constraint coefficient of constraint $i$ ( $a_{*ji}$ )

### Simple example

Let us consider the following generic problem with two decision variables:

$$\max z = c_1 * x_1 + c_2 * x_2$$

s.t.

$$a_{11} * x_1 + a_{12} * x_2 \leq b_1$$

$$a_{21} * x_1 + a_{22} * x_2 \leq b_2$$

$$a_{31} * x_1 + a_{32} * x_2 \leq b_3$$

Applying the rules above, the objective function of the dual is of type minimise, there are 3 decision variables, as the primal has 3 constraints, and the coefficients of the objective function of the dual are the RHS of the primal, hence, the objective function of the dual becomes:

$$\min z = b_1 * u_1 + b_2 * u_2 + b_3 * u_3$$

Now, we have two constraints, one per every decision variable. The RHS are the coefficients of the objective function, and the dependent coefficient are obtained by transposing the coefficients of the primal, hence the dual problem is subject to the following constraints:

$$a_{11} * u_1 + a_{21} * u_2 + a_{31} * u_3 \geq c_1 \quad a_{12} * u_1 + a_{22} * u_2 + a_{32} * u_3 \geq c_2$$

Now, since the constraints of the primal are of type *leq*, the three decision variables are non-negative:

$$u_1, u_2, u_3 \geq 0$$

### Bounding explanation

One way to deduct the dual is by acknowledging that, if the objective function of the primal is of type maximize and all the constraints are of type less or equal, linear combinations of the constraints provide **upper bounds** for the objective function. For instance, let us consider the following problem:

$$\max z = 4 * x_1 + x_2 + 5 * x_3 + 3 * x_4$$

s.t.

$$x_1 - x_2 - x_3 + 4 * x_4 \leq 1$$

$$5 * x_1 + x_2 + 3 * x_3 + 8 * x_4 \leq 55$$

$$-x_1 + 2 * x_2 + 3 * x_3 + 5 * x_4 \leq 3$$

Note that, if we multiply the second constraint by a factor of 2, we obtain an equivalent constraint:

$$10 * x_1 + 2 * x_2 + 6 * x_3 + 16 * x_4 \leq 110$$

Note also that all the LHS coefficients of this constraint are higher or equal than the coefficients of the objective function, and therefore:

$$\max z = 4 * x_1 + x_2 + 5 * x_3 + 3 * x_4 \leq 10 * x_1 + 2 * x_2 + 6 * x_3 + 16 * x_4 \leq 110$$

This constraint is an upper bound for the objective function, we do not know yet which is the maximum value of z, but we know that it cannot be higher than 110.

Note that we can find another upper bound using a linear combination of the second of third constraint, just by adding both constraints, we obtain:

$$4 * x_1 + 3 * x_2 + 6 * x_3 + 13 * x_4 \leq 58$$

Again, this linear combination is an upper bound of the objective function, since all the coefficients are higher or equal the coefficients of the objective value, for any value of the decision variables, we have:

$$\max z = 4 * x_1 + x_2 + 5 * x_3 + 3 * x_4 \leq 4 * x_1 + 3 * x_2 + 6 * x_3 + 13 * x_4 \leq 58$$

Therefore, even if we do not know which is the optimal value of the objective function, 58 is an upper bound, the optimal value cannot exceed this bound.

Note that finding the maximum value of z, is equivalent to finding the **minimum upper bound**, hence, the minimum linear combination of the constraints, or equivalently, the minimum linear combination of the RHSs:

$$\min z^* = 1 * u_1 + 55 * u_2 + 4 * u_3$$

which yields coefficients that are higher or equal the coefficients of the objective function for every decision variable:

$$u_1 + 5 * u_2 - u_3 \geq 4$$

$$-u_1 + u_2 + 2 * u_3 \geq 1$$

$$-u_1 + 3 * u_2 + 3 * u_3 \geq 5$$

$$4 * u_1 + 8 * u_2 + 5 * u_3 \geq 3$$

Note that this is the dual of the primal problem.

### Complementary slackness

Following the reasoning of the previous section, note that the decision variables of the dual are equivalent to the **Lagrangian multipliers** of the constraints in the Lagrangian of the primal:

$$\max z = \sum_{i=1}^n c_i * x_i + u_1 * (a_{11} * x_1 + a_{12} * x_2 + \dots + a_{1n} * x_n) + u_2 * (a_{21} * x_1 + a_{22} * x_2 + \dots + a_{2n} * x_n) + \dots$$

Maximizing the objective function is equivalent to minimizing the part of the Lagrangian multipliers.

But most importantly, Dual variables represent changes in the primal objective function per unit of constraint, that is, if an RHS of a constraint  $i$  is modified in 1 unit, the objective function is going to change in  $u_i$  units.

For this reason, the decision variables of the dual are referred to as the **shadow prices** of the constraints. The term shadow highlights that they are not factored in the objective function, and the term price means that they represent the impact in the objective function.

It follows that if a constraint is non-binding, meaning that the corresponding slack variable is non-zero, then the corresponding decision variable, or shadow price must necessarily be zero (that is, the shadow price must be zero, because we have a surplus or excess, and a change in the RHS of the constraint is not going to change the objective variable, just the value of the surplus or excess). Conversely, if a constraint is binding (meaning that the corresponding slack variable is zero), then the corresponding dual variable (shadow price) is basic (non-zero), because a change in the RHS of the constraint can cause a change in the objective variable.

Same applies for the dual: If a constraint of the dual is binding (meaning that the slack variable is zero, or non-basic), then the corresponding primal variable is basic. Also, if a constraint of the dual is non-binding (meaning that the slack variable is non-zero, or basic), then the corresponding primal variable is non-basic.

This principle is called **complementary slackness** meaning that the slack values or *slackness* of dual and primal are complementary.

## 2.1.6 Sensitivity: Algebraic explanation

### Changes in the coefficients of the objective function

Let us start with the Production Mix Problem example, which in the standard form is:

$$\max Z = 300x_1 + 250x_2 + 0s_1 + 0s_2 + 0s_3$$

subject to:

$$2x_1 + x_2 + s_1 = 40$$

$$x_1 + 3x_2 + s_2 = 45$$

$$x_1 + s_3 = 12$$

The final Tableau is:

Basic	z	$x_1$	$x_2$	$s_1$	$s_2$	$s_3$	RHS	Ratio
•	1	0	0	0	$250/3$	$650/3$	6350	•
$s_1$	0	0	0	1	$-1/3$	$-5/3$	5	•
$x_2$	0	0	1	0	$1/3$	$-1/3$	11	•
$x_1$	0	1	0	0	0	1	12	•

Let us assume we want to make a change in coefficient  $c_1$  of  $\Delta c_1$ . The objective function becomes:

$$\max Z = (300 + \Delta c_1) \cdot x_1 + 250x_2$$

It can be proved that the final Tableau now becomes:

Basic	z	$x_1$	$x_2$	$s_1$	$s_2$	$s_3$	RHS	Ratio
•	1	$-\Delta c_1$	0	0	$250/3$	$650/3$	6350	•
$s_1$	0	0	0	1	$-1/3$	$-5/3$	5	•
$x_2$	0	0	1	0	$1/3$	$-1/3$	11	•
$x_1$	0	1	0	0	0	1	12	•

The Simplex method requires that the Tableau is in canonical form. Thus, we need to add the last row times  $\Delta c_1$  to the first row to fulfill this:

Basic	z	$x_1$	$x_2$	$s_1$	$s_2$	$s_3$	RHS	Ratio
•	1	0	0	0	$250/3$	$650/3 + \Delta c_1$	$6350 + 12 \cdot \Delta c_1$	•
$s_1$	0	0	0	1	$-1/3$	$-5/3$	5	•
$x_2$	0	0	1	0	$1/3$	$-1/3$	11	•
$x_1$	0	1	0	0	0	1	12	•

And taking into account that the coefficient of  $s_3$  needs to be positive for  $x_1$  to remain in the basis,  $\Delta c_1$  cannot be lower than  $-650/3$  and can increase up to infinity.

Let us now assume we want to make a change in coefficient  $c_2$  of  $\Delta c_2$ . The objective function becomes:

$$\max Z = 300 \cdot x_1 + (250 + \Delta c_2) \cdot x_2$$

The final Tableau now becomes:



Basic	z	$x_1$	$x_2$	$s_1$	$s_2$	$s_3$	RHS	Ratio
•	1	0	$-\Delta c_2$	0	250/3	650/3	6350	•
$s_1$	0	0	0	1	-1/3	-5/3	5	•
$x_2$	0	0	1	0	1/3	-1/3	11	•
$x_1$	0	1	0	0	0	1	12	•

Again, the Simplex method requires that the Tableau is in canonical form. Thus, we need to add the second last row times  $\Delta c_2$  to the first row to fulfill this:

Basic	z	$x_1$	$x_2$	$s_1$	$s_2$	$s_3$	RHS	Ratio
•	1	0	0	0	$\frac{250}{3} + \frac{\Delta c_2}{3}$	$\frac{650}{3} - \frac{\Delta c_2}{3}$	$6350 + 11 \cdot \Delta c_2$	•
$s_1$	0	0	0	1	-1/3	-5/3	5	•
$x_2$	0	0	1	0	1/3	-1/3	11	•
$x_1$	0	1	0	0	0	1	12	•

And taking into account that the coefficients of  $s_2$  and  $s_3$  need to be positive for  $x_2$  to remain in the basis,  $\Delta c_2$  cannot be lower than -250 nor higher than 650.

### Changes in the constraint independent terms

Now let us see what is the effect of changing b, for instance decreasing  $b_1$  an amount equal to  $\Delta b_1$ . In the original problem formulation, this means:

$$2x_1 + x_2 + s_1 = 40 - \Delta b_1$$

$$x_1 + 3x_2 + s_2 = 45$$

$$x_1 + s_3 = 12$$

Changing the Right Hand Side by  $\Delta b_1$  is equivalent to changing  $s_1$  an amount of  $\Delta b_1$ . Note that  $s_1$  is in the basis, thus for it to remain in the basis, it must satisfy:

$$s_1 = 5 + \Delta b_1 \geq 0$$

$$\Delta b_1 \geq -5$$

Now, take the second constraint and let us apply the same change:

$$2x_1 + x_2 + s_1 = 40$$

$$x_1 + 3x_2 + s_2 = 45 - \Delta b_2$$

$$x_1 + s_3 = 12$$

Again, this is equivalent to:

$$2x_1 + x_2 + s_1 = 40$$

$$x_1 + 3x_2 + s_2 + \Delta b_2 = 45$$

$$x_1 + s_3 = 12$$

Which can be regarded as a change of  $\Delta b_2$  in  $s_2$ . Now, if we change  $s_2$  from  $s_2 = 0$  to  $s_2 = \Delta b_2$  in the final Tableau:

Basic	z	$x_1$	$x_2$	$s_1$	$s_2$	$s_3$	RHS	Ratio
.	1	0	0	0	$250/3 \cdot \Delta b_2$	$650/3$	6350	.
$s_1$	0	0	0	1	$-1/3 \cdot \Delta b_2$	$-5/3$	5	.
$x_2$	0	0	1	0	$1/3 \cdot \Delta b_2$	$-1/3$	11	.
$x_1$	0	1	0	0	$0 \cdot \Delta b_2$	1	12	.

The basis remains unchanged as long as all the constraints are still met:

$$s_1 - 1/3 \cdot \Delta b_2 = 5$$

$$x_2 + 1/3 \cdot \Delta b_2 = 11$$

$$x_1 + 0 \cdot \Delta b_2 = 12$$

$$s_1 = 5 + 1/3 \cdot \Delta b_2 \geq 0 (\Delta b_2 \geq -15)$$

$$x_2 = 11 - 1/3 \cdot \Delta b_2 \geq 0 (\Delta b_2 \leq 33)$$

$$x_1 = 12 - 0 \cdot \Delta b_2 \geq 0 (\Delta b_2 \leq Inf)$$

## 2.1.7 Interactive Sensitivity Analysis

Try me



### Introduction

In this notebook, you can experiment with the graphic representation of a linear programming problem. The objective is to assess the impact that the changes in the problem coefficient have in the optimal solution.

## Problem model

The problem model is taken from the Production Mix problem:

$$\max Z = 300x_1 + 250x_2 + 0s_1 + 0s_2 + 0s_3$$

And the constraints as:

$$2x_1 + x_2 + s_1 = 40$$

$$x_1 + 3x_2 + s_2 = 45$$

$$x_1 + s_3 = 12$$

## Changes in the objective function coefficient

The code cell below allows you to modify the coefficients of the objective function through sliders, to see how these changes modify the slope of the objective function. Depending on the changes, the optimal solution might be at another vertex, so the objective function slider allows to modify the objective variable z to find the optimal value in these cases. Remember, the optimal value will be always the highest value of z where the objective function intersects with the feasibility region.

```
[1]: #Import the numpy and pyplot libraries, we set the aliases np and plt so that it is
      ↪ easier to use
import numpy as np
import matplotlib.pyplot as plt
import ipywidgets as widgets
from ipywidgets import interactive
#We set the mode inline of matplotlib to get the result at the output of the cell code
%matplotlib inline

#Construct lines, in our coordinate system x represents our decision variable x1 and
      ↪ y represents our decision variable x2
x=np.linspace(0,30,2000) #2000 numbers from 0 to 30

def show_solution(z=6350, c1=300, c2=250):

    y1=40 - 2*x      # Operator time constraint
    y2=15-x/3       # Machining time constraint

    plt.figure(1)

    #1. Make plot
    plt.plot(x, y1, label=r'$2x_{1} + x_{2} \leq 40$', c='orange', ls='dashed')
    ↪ #Plot y1
    plt.plot(x, y2, label=r'$x_{1} + 3x_{2} \leq 45$', c='red')      #Plot y2
    plt.axvline(x=12, label=r'$x_{1} \leq 12$', c='green')        #plot y3

    #2. Adjust axis
    plt.xlim((0, 30))
    plt.ylim((0, 45))
    plt.xlabel(r'$x_{1}$')
    plt.ylabel(r'$x_{2}$')

    #3. Fill feasible region
    y4=np.minimum(y1, y2) #line representing the maximum between y3 and y2
    plt.fill_between(x, y4, 0, where=x<12, color='grey', alpha=0.5) #fill where y5 ys
    ↪ greater than y6
```

(continues on next page)

(continued from previous page)

```

#4 Annotate graph
plt.annotate('A', xy=(0, 0))
plt.annotate('B', xy=(0, 15))
plt.annotate('D',xy=(12, 0))
plt.annotate('E', xy=(12,11))

# 5. Plot objective function
obj_func= (z - c1*x)/c2 # we clear the x2 with the value of z
plt.plot(x, obj_func, label=r'$z = c_1*x_1 + c_2*x_2$', c='black')

# 6. Plot legend
plt.legend(bbox_to_anchor=(1.05,1), loc=2, borderaxespad=0.)

plt.show()

interactive_plot = interactive(show_solution, z=(3000,15000,10), c1=(10,500,10), c2=(100,1000,10))
output = interactive_plot.children[-1]
output.layout.height = '350px'
interactive_plot
    
```

## Analysis questions

1. Keep the objective function coefficient of the second variable to 250 and **decrease** the coefficient of the first variable. What happens to the slope of the objective function? Adjust the value of  $z$  until you find again the optimal value of the objective function. What happens to the optimal value of the objective function? Will it be always at the same vertex?
2. Now, repeat the experiment in the first question and **increase** the coefficient of the first variable. What happens to the slope of the objective function? Adjust the value of  $z$  until you find again the optimal value of the objective function. What happens to the optimal value of the objective function? Do you think it will always be at the same vertex?
3. Repeat the experiment in the first two questions but now changing the coefficient of the second variable and write down your conclusions.

## Changes in the constraint independent terms

The code cell below allows you to modify the independent terms of the constraints. The changes are going to change the feasibility region and again, if they are too extreme, the solution might change to another vertex.

```

[2]: #Import the numpy and pyplot libraries, we set the aliases np and plt so that it is
      ↪easier to use
import numpy as np
import matplotlib.pyplot as plt
import ipywidgets as widgets
from ipywidgets import interactive
#We set the mode inline of matplotlib to get the result at the output of the cell code
%matplotlib inline

#Construct lines, in our coordinate system x represents our decision variable x1 and
      ↪y represents our decision variable x2
x=np.linspace(0,50,2000) #2000 numbers from 0 to 30
    
```

(continues on next page)

(continued from previous page)

```

def show_solution(z=6350, b1=40, b2=45, b3=12):

    y_min=x*0          # non negativity constraint
    y_max = 250 + 0*x

    y1 = b1 - 2*x      # Operator time constraint
    y2=b2/3-x/3       # Machining time constraint

    plt.figure(1)

    #1. Make plot
    plt.plot(x, y1, label=r'$2x_{1} + x_{2} \leq b_1$', c='orange')          #Plot y1
    plt.plot(x, y2, label=r'$x_{1} + 3x_{2} \leq b_2$', c='red')          #Plot y2
    plt.axvline(x=b3, label=r'$x_{1} \leq b_3$', c='green')          #plot y3

    #2. Adjust axis
    plt.xlim((0, 50))
    plt.ylim((0, 45))
    plt.xlabel(r'$x_{1}$')
    plt.ylabel(r'$x_{2}$')

    #3. Fill feasible region
    y4=np.minimum(y1, y2) #line representing the maximum between y3 and y2
    plt.fill_between(x, y4, 0, where=x<b3, color='grey', alpha=0.5) #fill where y5 ys
    #greater than y6

    # 5. Plot objective function
    obj_func= (z - 300*x)/250 # we clear the x2 with the value of z
    plt.plot(x, obj_func, label=r'$z = 300*x_1 + 250*x_2$', c='black')

    # 6. Plot legend
    plt.legend(bbox_to_anchor=(1.05,1), loc=2, borderaxespad=0.)

    plt.show()

interactive_plot = interactive(show_solution, z=(3000,15000,10), b1=(10,100,1),
    b2=(10,100,1), b3=(0,20,0.1))
output = interactive_plot.children[-1]
output.layout.height = '350px'
interactive_plot

```

### Analysis questions

1. Keep the Right Hand Side (RHS) coefficients of the second and third constraint unchanged and **increase** the RHS coefficient of the first constraint. What happens to the feasibility region? Now adjust  $z$  so that you find again the optimal value of the objective function. What happens to the optimal value of the objective function? Will it be always at the same vertex?
2. Keep the Right Hand Side (RHS) coefficients of the second and third constraint unchanged and **decrease** the RHS coefficient of the first constraint. What happens now to the feasibility region? And what happens to the optimal value of the objective function? Will it be always at the same vertex as before?
3. Now repeat the experiment in the first two questions but now changing the RHS coefficients of the second and third constraints and write down your conclusions.

### Sensitivity analysis wrap-up

As you have seen, changing the coefficients of the objective function and the constraints can have a significant impact on the solution of the linear programming problem. In a nutshell, a sensitivity analysis explores the impact of changes of the coefficients of the objective function and the constraints on the solution of the linear programming problem. A solver providing a sensitivity analysis normally provides the following information:

- **Objective function coefficients:** The sensitivity analysis provides the range of values for the coefficients of the objective function for which the current optimal solution remains optimal. Normally, a solver will provide the **upper** and **lower** bounds for the coefficients of the objective function. If the coefficients are within these bounds, the current optimal value of decision variables and constraint slack values will remain optimal. If the coefficients are outside these bounds, the current optimal solution will not be optimal anymore. In the graphic representation, this will mean that the optimal solution will no longer be at the same vertex.
- **Right Hand Side (RHS) coefficients:** Similarly, a sensitivity analysis provides the range of values for the Right Hand Side (RHS) coefficients for which the current optimal solution remains optimal. Normally, a solver will provide the **upper** and **lower** bounds for the RHS coefficients. If the coefficients are within these bounds, the current optimal value of decision variables and constraint slack will value remain optimal. If the coefficients are outside these bounds, the current optimal solution will not be optimal anymore. Again, in the graphic representation, this will mean that the optimal solution will no longer be at the same vertex.

Sensitivity analysis is a powerful tool to understand how the changes in the coefficients of the objective function and the constraints affect the solution of the linear programming problem. It is important to understand that the sensitivity analysis is a local analysis, meaning that the conclusions are valid only for small changes in the coefficients, and unitary changes that we apply to one and only one coefficient at a time.

The following tables provide the results of a sensitivity analysis applied to the problem at hand in this interactive tutorial:

Decision variable	value (primal problem solution)	Reduced Costs (dual problem solution)	Objective function coefficient	Objective function lower bound	Objective function upper bound
x1	12	0	300	<b>83.3</b>	Infinity
x2	11	0	250	0	<b>900</b>

Con-straint	Right Side (RHS)	Slack (primal problem solution)	Shadow price (dual problem solution)	RHS lower bound	RHS upper bound
1	40	5	0	<b>35</b>	Infinity
2	45	0	83.3	<b>12</b>	<b>60</b>
3	12	0	216.67	<b>0</b>	<b>15</b>

Now, run the colab and try to set the coefficients of the objective function and the constraints to the values in the upper and lower bound values in the tables. You will see that the optimal solution remains the same while the coefficients are within the bounds. However, if you set the coefficients outside the bounds, the optimal solution will no longer be optimal.

If you set the first coefficient exactly to 83.3 you will notice that the objective function will be parallel to the constraint line. This means that any change below this point will shift the optimal towards the vertex to the right. A similar effect will happen if you set the second coefficient to 900.

On the other hand, if you shift the first constraint to 35, you will notice that now the three constraints intersect at the same point. If you make any changes to  $b_1$  below this point, the optimal solution will no longer be at the same vertex. The same thing happens if you try any RHS lower bound and upper bound in the table.

Do you want to learn more about sensitivity analysis? Check the following tutorial for a more detailed explanation of the algebraic concepts behind sensitivity analysis

- [Sensitivity analysis tutorial](#)

## 2.2 Exercises

In this section you have a collection of CLP problems sorted by difficulty:

- **Easy problems:** You can tackle these problems after a first introduction to CLP.
- **Normal problems:** These problems require that you are familiar with concepts like Duality and the Simplex method.
- **Hard problems:** This is the type of problems that definitively get you ready for the exam.

### 2.2.1 Problem Definition

A gold seeker needs to find at least 2 kg of bronze and 3 kg of copper to pay his gambling debts. There are two rivers in which the miner can find these precious metals. Each day the miner spends in river A, he finds 0.3 kg of bronze and 0.3 kg of copper while each day he spends in river B, he finds 0.2 kg of bronze and 0.4 kg of copper.

*Can you help this gold seeker decide how many days to spend in each day to pay his debts as soon as possible?*

### 2.2.2 The good carpenter

“The Fog Ltd” is a modern furniture company that makes tables and chairs. The selling price of a table is 30€ and the selling price of chairs is 10€. The carpenter at the company works up to 40 hours and needs 6 hours to make a table and 3 hours to make a chair. Customer demand requires that he makes at least 3 times as many chairs as tables. Tables take up four times as much storage space as chairs and there is room for at most four tables each week.

What is the optimal weekly production plan that optimises revenues?

Formulate this problem as a linear programming problem.

### 2.2.3 Ski Patrol

#### Problem Definition

Ski Patrol Ltd. manufactures two different models of snowboards branded Normal and Deluxe with a weekly production plan and inventory. They sell their material to local shops. The company obtains a profit of 300€ per Normal snowboard and 400€ per Deluxe snowboard. The manufacturing process consists of roughing and polishing. The Normal snowboard model requires 20 minutes of roughing and 2/3 minutes of polishing. The Deluxe snowboard model requires 40 minutes of roughing and 1 minute of polishing. The roughing equipment is available for 1000 minutes per week and the polishing equipment for 800 minutes per week. Ski Patrol has already signed a contract with a local shop called Downhill Racer to deliver 18 Normal snowboards per week.

Ski Patrol hires you to formulate a Linear Programming Problem to help them determine the best production mix to maximise benefits.

**Write a CLP problem to maximize profits**

### 2.2.4 Production Mix example

#### Problem Definition

Production Mix A company produces two kinds of products, P1 and P2.

The selling price of P1 is 300€ The selling price of P2 is 250€

- P1 requires 2 operator man-hours and 1 hour of machine operating time
- P2 requires 1 operator man-hours and 3 hours of machine time
- Operator man-hours cannot be higher than 40 hours per day
- Machine operating time cannot be higher than 45 hours per day
- Marketing states that P1 units per day cannot be higher than 12

**How many units of P1 and P2 should the company produce every day to maximise income?**

#### Model

We want to find the maximum solution to:

$$\max Z = 300x_1 + 250x_2$$

This is our objective function. The decision variables are:

$x_1$  : units of Product 1

$x_2$  : units of Product 2

The objective function is subject to the following constraints:

$$2x_1 + x_2 \leq 40$$

$$x_1 + 3x_2 \leq 45$$

$$x_1 \leq 12$$

We define the problem in its standard form as:

$$\max Z = 300x_1 + 250x_2 + 0s_1 + 0s_2 + 0s_3$$



And the constraints as:

$$2x_1 + x_2 + s_1 = 40$$

$$x_1 + 3x_2 + s_2 = 45$$

$$x_1 + s_3 = 12$$

### 2.2.5 Problem definition

Din Djarin is a retired bounty hunter that now runs a small factory specialised in making classic weapons in a colony recently reestablished in Mandalore. The factory manufactures two different types of weapons, swords and spears. Din uses special alloys, mixtures of metals and rare materials, which have made a name for the endurance of his weapons across the colony. The table below shows the weight in pounds of each material needed to make each type of weapon:

Metal	spear	sword
Beskar	1	0.5
Tungsten	1	1
Titanium	3	1

And the table below shows the total amount of each metal available, also in pounds:

Metal	Amount
Beskar	125
Tungsten	225
Titanium	300

The price of a spear is 40 credits and the price of a sword is 5 credits. Din has already sold 100 swords, so he needs to manufacture at least that many.

**Write a CLP problem to find the optimal number of spears and swords that maximise revenues**

## 2.2.6 Blending Cremanet

### Problem description



Cremanet is an energy drink that is inspired in the typical Valencian drink “Cremaet”. The main ingredients of Cremaet are:

- Coffee: Coffee has a strong flavor and a high caffeine content. The specific type of coffee grain used in Cremaet yields a cost of 4€ per liter, and it contains 170 mg of caffeine per liter.
- Rum mix: The rum mix is a blend of rum, sugar, and orange peel. The cost of the rum mix is 15€ per liter, and it contains 9% alcohol (which corresponds to 9 grams of ethanol per 100ml of rum mix).

Cremanet wants to position itself as a premium energy drink with a mild level of alcohol, according to the following specifications:

- The caffeine content of Cremanet should be at most 100 mg per liter.
- The alcohol content of Cremanet should be at most 5%.

The company has already received an order to produce 10 liters of cremanet. Also, the maximum availability of coffee is 20 liters, and the maximum availability of rum mix is 10 liters.

**Develop a linear programming model that allows Cremanet to determine the amount of coffee and rum mix that should be used in the production of Cremanet, such that the cost of the ingredients is minimized.**

## 2.2.7 Making Robots

### Problem Definition

The company Tetravaal located in Johannesburg manufactures two types of robots, Model  $P_1$  and Model  $P_2$ . The production plant is consisted of four different sections: metal machining, plastic moulding, electrical work and assembly. The metal machining section has a capacity of 7500 units of  $P_1$  or 6000 units of  $P_2$  per month.

Plastic moulding can process 5000 units of  $P_1$  or 9000 units of  $P_2$  per month.

Electrical work can process 6000 units of  $P_1$  or 7000 units of  $P_2$  per month.

In Assembly, there are two assembly lines that work in parallel, one per each robot model.

The first assembly line can process 4000 units of  $P_1$  per month

The second assembly line can process 5000 units of  $P_2$  per month

Knowing that the unitary profit of  $P_1$  is 500€ and that the unitary profit of  $P_2$  is 600€, and that both robots have a great demand and therefore all manufactured robots are sold, Michelle Bradley, CEO of Tetravaal, asks his engineering team:

Calculate the number of units of each robot that needs to be manufactured to maximise profit for the company.

**Write a CLP problem to calculate the number of units of each robot that needs to be manufactured to maximise profit for the company.**

## 2.2.8 Blending Craft Beer

### Problem Definition

Duff Beer is an american brewery that sells alcoholic beverages worldwide. Duff beer has a very large market share in the US. One of the new business lines of Duff Beer is in the craft beer sector. Duff beer buys 3 types of wort from local suppliers. The availability of wort (in liters) and the price for each of them is specified in Table 1.

**Table 1:** daily availability and cost of wort

Wort	Availability (liters)	Cost (€cents/liter)
Type 1	2500	120
Type 2	1200	095
Type 3	2000	150

The blend of the three types of wort must comply with the following quality requirements:

- For Beer Angels, no less than 60% of Type 3 and no more of 20% of Type 2. The price of one bottle of Beer Angel is 4.10€/liter.
- For Beer Beast, no less than 15% of Type 3 and no more than 60% of Type 2. The price of one bottle of Beer Beast is 2.80€/liter.

- For Beer Cactus, no more than 50% of type 2. The selling price is 2.45 €/liter.

Write a linear program to find the most profitable blend of wort to elaborate the three kinds of beers

## 2.2.9 Blending Problem

### Problem definition

The Kahuna company manufactures sausages using three kinds of meat. The relevant information about the ingredients is provided in the table below:

Ingredient	Cost (€/kg)	Availability (kg)
Pork	4.32	30
Wheat	2.46	20
Starch	1.86	17

The company makes two types of sausages: \* Economy ( $\geq 40\%$  Pork) \* Premium ( $\geq 60\%$  Pork)

One sausage is 50 grams (0.05 kg)

According to government regulations, the most starch we can use in our sausages is 25%

We have a contract with a butcher, and have already purchased 23 kg pork, that will go bad if it's not used.

We have a demand for 350 economy sausages and 500 premium sausages.

Write a linear program to figure out the most cost-effective recipe to manufacture our sausages.

## 2.2.10 Petroleum Blending

### Problem Definition

Harry Stamper is an engineer working on a petroleum company. The company produces three grades of motor oil – super, premium, and extra. The three grades are made from the same 3 ingredients, named component 1, component 2 and component 3. The company wants to determine the optimal mix of the 3 components in each grade of motor oil that will maximize profit.

The maximum availability of each component and their cost per barrel are as follows:

Component	Max barrels available / day	Cost / barrel
Component 1	4.500	12€
Component 2	2.700	10€
Component 3	3.500	14€

To ensure the appropriate blend, each grade must have a minimum amount of component 1 plus a combination of other components as follows:

Grade	Component 1	Component 2	Component 3	Selling price/barrel
Super	At least 50%	No more than 30%	•	23€
Premium	At least 40%	•	No more than 25%	20€
Extra	At least 60%	At least 10%	•	18€

The company wants to make at least 3000 barrels of each blend. *Formulate an LP to help Harry Stamper find the optimal blend that maximises profit.*

**Analysis questions** (Do this part once you have reviewed CLP analysis including sensitivity analysis questions). You can check the solution in the Notebook Petroleum Blending (Solved CBC)

- Which is the total profit obtained by the company with the Extra grade with the solution provided by the solver?
- Which is the maximum value that the profit per barrel of component 2 in the Extra grade can take without changing the base of the optimal solution?
- If the cost of the barrel of component 3 increased 1€, will the optimal composition change? Motivate your response
- Do we use every barrel available of every component in the optimal solution? Or do we have any surplus availability for any component? Motivate your response
- The provider of Component 2 may have a problem with the delivery and the availability may be reduced in 400 barrels. Will this shortage affect the optimal composition? Motivate your response\*
- Grace wants to use the results to renegotiate the terms and conditions with the suppliers of the different components. Which component is a better candidate to increase the availability? How would you use the model to negotiate new terms with the provider, assuming you do not want any change in the base of the optimal solution? Motivate your response

## 2.2.11 Fasting Diet

### Problem I

You have read an article about the benefits of fasting. This article publishes a series of findings around the fasting diet method. Specifically, the article states that:

- Decreasing the number of meals to three, two heavier meals and a light meal in between has significant benefits for the health
- The study reveals that the optimal amount of daily calories for an adult is higher than 1800
- and also lower than 2200 calories
- The study also reveals that the light meal should not represent more than 10% of the number of daily calories
- And that the last meal should not provide more than 700 calories

You think you can design a Linear Programming Problem where the decision variables represent the calories of each meal to find an optimal diet that fulfils all these requirements, while at the same time provide the maximum number of calories. For such a model, define:

- a. The objective function
- b. The constraints

c. The dual problem

[ ]:

## 2.2.12 Bonuses and merits

### Problem definition

A company pays a yearly bonus to employees based on the following indicators:

1. **Performance:** Performance is evaluated by the direct manager using a quantitative method. The indicator is labelled as  $CM_1$  and measured in a scale from 1 to 5.
2. **Involvement:** Involvement in social media, events, coaching and other voluntary activities. Involvement is measured by the human resource department using a quantitative method. The indicator is labelled as  $CM_2$  and measured in a scale from 1 to 5.
3. **Professional development:** Professional development evaluates the accomplishment of training programs and courses suggested by the proximity manager. The indicator is labelled as  $CM_3$ .

This year, the total score obtained among all employees is:

$$\sum(1) = 1625 \text{ Total points in performance}$$

$$\sum(2) = 1409 \text{ Total points in involvement}$$

$$\sum(3) = 1387 \text{ Total points in professional development}$$

The company wants to calculate bonuses using a linear expression weighting the three The board established the following rules:

- No employee should receive more than 5000€ in bonuses
- The value of performance is twice the value of involvement and professional development
- The total budget for bonuses this year is 1.5M€

Formulate an LPP that determines the optimal formula to calculate employee bonuses.

[ ]:

## 2.2.13 Resourcing: Product Mix of Aluminium Windows

### Problem Definition

Weyland Corp is a British firm that manufactures solar panel cells. This firm supplies three different types of cell called: fast, normal and ultra. There are three operations involved in the manufacturing process. The following table describes the hours/month required to manufacture each model:

**Table 1:** hours/month requirements

Operation	Fast (hours/month)	Normal (hours/month)	Ultra (hours/month)
1	1	3	2
2	2	0	3
3	1	4	0

And the following table describes the total hours available for each operation:

**Table 2:** Total hours/month available

Operation	Hours/Month
1	400
2	600
3	600

The profit per each model is specified in the following table: **Table 3:** Unit of profit per model

Model	Profit/Unit
Normal	30
Economic	20
Luxury	40

### Analysis questions

The following solution has been obtained using PuLP and the CBC solver.

The value for the overall profit is:

$$z = 9666.67$$

The following table shows the values of the decision variables and their corresponding reduced costs:

	Variables	Solution	Reduced cost
fast	cells_fast	300	-0
normal	cells_normal	33.3333	-0
ultra	cells_ultra	0	-8.33333

The following table shows the slack values of the constraints and their corresponding shadow prices:

	Constraint	Right Hand Side	Slack	Shadow Price
0	Operation_1	400	-0	6.66667
1	Operation_2	600	-0	11.6667
2	Operation_3	600	166.667	-0

Based on this solution, try to answer the following **analysis questions** ### Question 1 Peter Weyland believes that, for the reputation of the company, Weyland should manufacture at least 40 ultra cells per month. If this plan goes ahead, how would it affect the current profit?

#### Question 2

The engineer informed management that due to maintenance operations, the capacities of Operation 1, 2, and 3 have changed to 360, 600 and 500 respectively. How would this affect the profit?

### Question 3

Mr. Weyland insisted that the company needs to change the strategy towards producing ultra-fast cells to position Weyland Corp in this segment. The company reviewed the prices of the ultra-fast cells and changed them to 30€, 10€ and 50€ per fast, normal and ultra cells. With the new prices, would it be profitable to produce ultra-fast cells?

### Question 4

What is the impact in the new profit if we opted for this new prices?

### Question 5

The engineer is studying a new type of fast cells that would require 2 hours of operation 1, 0 hours of operation 2 and 1 hour of operation 3. The estimated market price is 20€. How would this affect the production plan?

### Question 6

Mr. Weyland thinks that the operation time for Operation 2 for the normal cells can be cut to only 3h. How would this improve the production plan?

## 2.2.14 The chilling adventures of Sabrina

### Problem definition

Spellman's Ltd is a company that manufactures chilling soft drinks. They want to manufacture two types of drinks A, and B. Both beverages use a semi-elaborate C, another expensive ingredient D and other ingredients that are not relevant for production planning. Sabrina is a young student of engineering and management doing an internship at Spellman's. She needs to formulate a Continuous Linear Program to configure the optimal daily production plan for the company.

The selling price of drink A is 3€/liter and the selling price of drink B is 2€/liter.

1 liter of drink A uses 3 grams of ingredient D. A liter of drink B uses 1 gram of ingredient D. There are only 3 grams of ingredient D available per day.

The factory only has one mixer to elaborate both drink types and the semi-elaborate. It takes 1 hour to process a liter of drink A, 1 hour to process 1 liter of drink B, and 1 hour to process 1cl of semi-elaborate C. The mixer is available 6 hours per day.

Drink A uses 2cl of semi-elaborate C and drink B uses 1cl of semi-elaborate C. The company has 3cl of semi-elaborate C plus the amount they decide to produce available per day.

1. Write a Continuous Linear Problem to help Sabrina design the optimal production plan that maximises revenues for the company.
2. Write the dual problem
3. Given the following solution:
  - 0 Liters of drink A
  - 3 Liters of drink B
  - 3 cl of semi-elaborate C

Verify the solution. Is the solution feasible? What are the values of the slack variables?

4. Use complementary slackness to find the dual solution corresponding to this vertex. Is the dual solution feasible? Is the solution optimal? Motivate your response



## 2.2.15 Those damn interns

### Problem II

You are the production manager of a company. You have used the following programming model to define the optimal production plan. The problem is expressed in the standard form. Objective function:

Max  $z = 3X_1 + 4X_2 - MA_1 + 0S_2 + 0S_3$  subject to:  $X_1 + 0X_2 + A_1 + 0S_2 + 0S_3 = 18$   $20X_1 + 40X_2 + 0A_1 + S_2 + 0S_3 = 1000$   
 $26.67X_1 + 20X_2 + 0A_1 + 0S_2 + S_3 = 800$  \*The slack variables are indexed in order of appearance.

The intern submitted the following data describing the solution obtained by a solver. This solver provides the solution of both the primal and the dual and a sensitivity analysis in a table, but your intern handed you the following handwritten report:

$X_1$  -> Solution value: 18 Reduced Cost: ? Status: basic Objective coefficient: ? Objective Lower Bound: -Inf Objective Upper Bound: 5.334

$X_2$  -> Solution value: 15.997 Reduced Cost: ? Status: ? Objective coefficient: 4 Objective Lower Bound: 2.2497 Objective Upper Bound: Inf

$A_1$  -> RHS= 18 Slack = 0 Sense = ? Shadow price= -2.334 Min RHS= 17.9964 Max RHS= 29.9963

$S_2$  -> RHS= 1000 Slack = 0.12 Sense = less or equal Shadow price= 0 Min RHS= 999. Max RHS= Inf

$S_3$  -> RHS= ? Slack = ? Sense = less or equal Shadow price= 0 Min RHS= 480.06 Max RHS= 800.06

- a. You analyse the solution and notice that there are some missing values noted as ? (You are going to fire this intern!). Write down the values missing in the solution. Motivate your response in each case
- b. Which of the two decision variables has a larger contribution to the optimal solution? Motivate your response.
- c. The first constraint determines the value of one of the decision variables ( $X_1$ ). What would happen to the objective function if the right hand side of this constraint changes to 19? and to 31? Motivate your response
- d. What would happen to the solution if the objective function coefficient of  $X_1$  changes to 5? And what if the coefficient of  $X_2$  changes to 3? Motivate your response

[ ]:

## 2.2.16 A software bug

### Problem Definition

You are using a legacy software to solve a production mix Continuous Linear Programming (CLP) problem in your company where the objective is to maximise profits. The following table provides the solution of the primal, dual and a sensitivity analysis for the three decision variables that represent the quantities to produce of each product

Variables	Solution	Reduced cost	Objective Coefficient	Lower bound	Upper bound
$x_1$	300.00	?	30.00	24.44	Inf
$x_2$	33.33	?	20.00	-0.00	90.00
$x_3$	0	-8.33	40.00	-Inf	48.33

Answer the following questions:

- a. Notice that there are some values missing (a bug in the software shows a ? sign instead of a numerical value, remember to use Python the next time around). Fill the missing values and explain your decision.

- b. According to the provided solution, which of the three products has the highest impact in your objective function? Motivate your response.
- c. What does the model tell you about product  $x_3$ ? Is it profitable to manufacture under the current conditions modeled in the problem? Provide quantitative values to motivate your response.
- d. Recall that in this type of problem, the objective coefficient represents the marginal profit per unit of product. What would happen if the objective function of variable  $x_3$  is increased over 50? Describe what would be the impact in the obtained solution The following table represents the value obtained for the decision variables related to the constraints:

Constraint	Right Hand Side	Shadow Price	Slack	Min RHS	Max RHS
$s_1$	400.00	6.67	0.00	300.00	525.00
$s_2$	600.00	11.67	0.00	0.00	800.00
$s_3$	600.00	0.00	166.67	433.33	Inf

Bearing in mind that the constraints represent the availability of 3 limited resources (operating time in minutes) and that the type of constraints is in every case “less or equal” answer the following questions:

- e. Which decision variables are basic? How many are there and, is this result expected? Motivate your response.
- f. Imagine that you had to cut down costs by reducing the availability of the limited resources in the model. Which one would you select? How much could you cut down without changing the production mix? Motivate your response
- g. Now, with the money saved, you could invest to increase the availability of other limited resources. Again, indicate which one would you select and by how much you would increase the availability of this resource without changing the production mix. Motivate your response
- h. How do the columns of the two tables relate to the primal and dual problem? Moreover, for each column, not considering upper and lower bounds, write down a brief description of each column and the relationship it has with the primal and dual problem

## 2.2.17 Invincible Nutritional Supplements

### Problem Definition

Invincible Ltd makes two types of nutritional supplements. The recipe of both supplements is based on three ingredients for which the company has limited availability. The table below shows the daily availability of the three ingredients and their costs:

**Table 1:** daily availability and cost of ingredients

Ingredient	Availability (grams)	Cost (€cents/gram)
1	1500	85
2	1300	90
3	2000	125

The selling prices are 3€/gram for supplement Awesome (product A) and 3.5€/gram for supplement Beast (product B).

The recipe must comply with the following requirements:

- Both products must contain at least 25% of ingredient 3.
- Product B must have at least 10% more of ingredient 2 than product A.
- The minimum quantity of any ingredient in any product is 10%

The factory has a production line that operates for 10 hours per day. The production speed of the line is 300 grams per hour for product A and 175 grams per hour for product B.

### Indices

We can define the following indices to express the problem in a compact form:

- $i$ : Ingredient  $i \in [1, 2, 3]$
- $j$ : Product  $j \in [A, B]$

### Decision variables

The decision variables are:

$x_{ij}$ : Amount of ingredient  $i$  (1, 2, 3) in product  $j$  (A, B) (ie  $i=[1,2,3]$ ,  $j=[A,B]$ ) in grams

### Objective Function

The objective function is:

$$\max z = \sum_{i=1}^3 \sum_{j=A}^B (d_j - c_i) * x_{ij}$$

where  $d_j$  is the selling price of a gram of product  $j$  and  $c_i$  is the cost of ingredient  $i$ .

$$\max z = (3-0.85)*x_{1A} + (3-0.9)*x_{2A} + (3-1.25)*x_{3A} + (3.5-0.85)*x_{1B} + (3.5-0.9)*x_{2B} + (3.5-1.25)*x_{3B}$$

$$\max z = 2.15 * x_{1A} + 2.1 * x_{2A} + 1.75 * x_{3A} + 2.65 * x_{1B} + 2.6 * x_{2B} + 2.25 * x_{3B}$$

### Constraints

Subject to the following constraints:

Availability constraints The availability constraints can be expressed in a compact form as:

$$\sum_j x_{ij} \leq a_i \quad \forall i$$

$$x_{1A} + x_{1B} \leq 1500$$

$$x_{2A} + x_{2B} \leq 1300$$

$$x_{3A} + x_{3B} \leq 2000$$

Both products must contain at least 25% of ingredient 3

$$x_{3A} \geq 0.25 * (x_{1A} + x_{2A} + x_{3A})$$

$$x_{3B} \geq 0.25 * (x_{1B} + x_{2B} + x_{3B})$$

Product B must have at least 10% more of ingredient 2 than product A:

$$x_{2B} \geq 1.1(x_{2A})$$

The minimum quantity of any ingredient in any product is 10%:

??

Production capacity constraint:

??

A solver provides the following tables with the solution of the primal and the dual:

Variables	Solution (GRB)	Reduced cost (GRB)	Objective Coefficient (GRB)	Objective Lower bound (GRB)	Objective Upper bound (GRB)
Grams of ingredient 1 in product A	1460.65	0	2.15	1.75	132.4
Grams of ingredient 1 in product B	39.35	0	2.65	-inf	3.58
Grams of ingredient 2 in product A	232.54	0	2.1	-inf	2.63
Grams of ingredient 2 in product B	255.79	0	2.6	-inf	3.09
Grams of ingredient 3 in product A	632.2	0	1.75	1.61	2.15
Grams of ingredient 3 in product B	98.38	0	2.25	-inf	3.05

idx	Constraint	Right Side	Hand	Shadow Price	Slack	Min RHS	Max RHS
0	Ingredient 1 availability	1500		0.4	0	271.89	1550.85
1	Ingredient 2 availability	1300		0	811.67	488.33	inf
2	Ingredient 3 availability	2000		0	1269.42	730.58	inf
3	Ingredient 3 $\geq 25\%$ in product_A	0		0	-50.85	-inf	50.85
4	Ingredient 3 $\geq 25\%$ in product_B	0		-1.04	0	-50.07	42.04
5	Product B $\geq 10\%$ of ingredient 2 than product A	0		-0.69	0	-330	42.04
6	Quantity of ingredient 1 in product A must be at least 10%	0		0	-1228.11	-inf	1228.11
7	Quantity of ingredient 1 in product B must be at least 10%	0		-1.04	0	-35.16	242.68
8	Quantity of ingredient 2 in product A must be at least 10%	0		-0.41	0	-300	21.82
9	Quantity of ingredient 2 in product B must be at least 10%	0		0	-216.44	-inf	216.44
10	Quantity of ingredient 3 in product A must be at least 10%	0		0	-399.66	-inf	399.66
11	Quantity of ingredient 3 in product B must be at least 10%	0		0	-59.03	-inf	59.03
12	Production capacity	10		512.58	0	9.67	15.69

- a** Write the expressions of the missing constraints. You may use indices to write your expressions in a compact form
- b** Write down the objective function of the dual problem and provide an economic interpretation of the dual variables
- c** What is the maximum price for ingredient 3 that the company can pay without changes in the optimal solution? Motivate your response
- d** The company can improve the daily capacity of the line in 1 hour for 50000€. How would this affect the profits and how long will it take for the company to make this investment profitable, considering only the increase in the daily profits
- e** Is the availability of ingredient 1 a binding constraint? How would you optimise the procurement of ingredients (daily amounts that are available) according to the provided solution?

## 2.2.18 Manufacturing Solar Panels Revisited

### Problem Definition

Consider the following solution to a daily production mix optimization problem where:

- The decision variables are the number of units to produce of three different types of products
- The objective function coefficients (Objective coefficient in table) are the unitary profits (difference between production costs and sell price per unit) in euros
- The three constraint represent the total availability of three sections in minutes

### Decision variables

Decision variable	Solution	Reduced cost	Objective Coefficient	Lower bound	Upper bound
product 1 units	300	0	30	24.44	inf
product 2 units	33.33	0	20	-0	90
product 3 units	0	-8.33	40	-inf	48.33

### Constraints

Constraint	Right Hand Side	Shadow Price	Slack	Min RHS	Max RHS
Section 1 availability	400	6.67	0	300	525
Section 2 availability	600	11.67	0	0	800
Section 3 availability	600	0	166.67	433.33	inf

Answer the following questions. Motivate your response based on the provided results:

- Is it profitable to produce units of the three types of products? If any of the products is not profitable, what changes are needed to make it profitable?
- In the actual economic context, the production costs are rising. Discuss how the rise of the production costs can affect your profit and identify the maximum unitary production cost increase that your daily production mix can support without changes in the base solution
- In order to increase your production capacity, you need to evaluate 2 different improvement proposals from an external consultant. The first one considers a 12% increase of the availability of section 1, and the second a 7.5% increase of the availability of section 2. Which one is more profitable for the company?

[ ]:

## 2.2.19 March & Sanchis Investments Ltd.

### Problem Definition

Suppose you are a financial advisor at March & Sanchis Investments Ltd. and your client has €1,000,000 to invest in four different types of assets: stocks, bonds, real estate, and crypto assets, with the Expected Returns and risk (an estimation of the risk per euro invested) expressed in the table below:

Asset Type	Expected Return (%)	Risk (%)
Stocks	10	1.25
Bonds	6	0.5
Real Estate	8	0.7
Crypto Assets	15	2.1

Your task is to help the client select the optimal portfolio of investments, maximizing the Expected Return. Assume that the investments are independent and take into account the following constraints:

- **Budget:** The total investment cannot exceed the budget of the client (€1,000,000)
- **Personal Preferences:** The client wants to invest at least €200,000 in real estate due to personal interests in owning property, and does not want to invest more than €400,000 in crypto assets as she would like to be cautious and limit exposure.
- **Risk:** The overall risk estimation cannot exceed 12% of the expected return

Model this problem as a continuous linear programming problem. What are the decision variables? What is the objective function? What are the constraints?

## 2.2.20 March & Sanchis Investments Ltd. Revisited

### Problem Definition

You are an investment analyst at March & Sanchis Investments Ltd. and you need are analysing the portfolio of a customer, based on the investment case described and solved here

A solver provides this solution:

### Decision Variables:

	Vari- ables	Solution (GRB)	Reduced cost (GRB)	Objective Coefficient (GRB)	Coeffi- cient (GRB)	Objective bound (GRB)	Lower	Objective bound (GRB)	Upper
1	x_1	116129.03	0.00	0.10	0.08			0.12	
2	x_2	0.00	-0.02	0.06	-Inf			0.08	
3	x_3	483870.97	0.00	0.08	0.05			0.10	
4	x_4	400000.00	0.00	0.15	0.12			inf	

### Constraints:

	Constraint	Right Side	Hand	Slack	Shadow Price	Min RHS	Max RHS
0	Maximum_budget	1000000.00		0.00	0.10	861538.46	2053846.15
1	Risk	0.00		0.00	6.45	-360.00	880.00
2	Minimum_investment_in_real_es- tate	200000.00		-283870.97	0.00	-Inf	483870.97
3	Maximum_invest- ment_in_crypto_assets	400000.00		0.00	0.03	48000.00	464285.71

Answer the following questions and motivate all of your responses based on the data provided in the solution

- How sensitive is the solution to changes in the expected return of the different types of assets? Describe how changes in the expected return of each asset could affect the solution.
- What is the minimum expected return of bonds that would render this asset profitable for your customer?
- Based on the solution provided, would you recommend your customer to consider increasing or decreasing the exposure to crypto assets?

### 2.2.21 Production Mix

#### Problem definition

Chappie Ltd. has 3 production lines (A, B, and C) to make 4 different types of metal sheets of width 0.4, 0.5, 0.6, and 0.7 mm. The company needs to meet the monthly demand for each kind of product (in Tons) expressed in the next table:

Sheet 0.4 mm	Sheet 0.5 mm	Sheet 0.6 mm	Sheet 0.7 mm
500	1200	1500	300

The production lines do not have the same overall efficiency for all types of sheets. Furthermore, due to design constraints, not all the lines are capable of manufacturing all types of sheets. The table below indicates the manufacturing costs in € of the different products in the different lines: a dash “-“ means that it is not possible to manufacture a product in a line:

Line	Sheet 0.4 mm	Sheet 0.5 mm	Sheet 0.6 mm	Sheet 0.7 mm
A	60	50	50	45
B	80	70	75	70
C	.	60	60	.

The different lines have the following capacities (in hours) in the planning period:

A	B	C
500	480	370

And the following table represents the manufacturing time (hours) per ton of product required for each type of metal sheet in the different lines:

Line	Sheet 0.4 mm	Sheet 0.5 mm	Sheet 0.6 mm	Sheet 0.7 mm
A	0.4	0.3	0.3	0.25
B	0.8	0.6	0.7	0.6
C	.	0.5	0.4	.

**Formulate a linear programming problem to obtain the monthly production plan (quantity of metal sheet in tons of each type of metal sheet to be manufactured in each production line)**

[ ]:

## 2.2.22 Sabrina Revisited

### Analysis of the problem model

Spellman's Ltd is a company that produces two types of chilling soft drinks: A, and B. Both beverages use a semi-elaborate C, another expensive ingredient D and other ingredients that are not relevant for production planning.

The selling price of drink A is 3€/liter and the selling price of drink B is 2€/liter.

1 liter of drink A uses 3 grams of ingredient D. A liter of drink B uses 1 gram of ingredient D. There are only 3 grams of ingredient D available per day.

The factory only has one mixer to elaborate both drink types and the semi-elaborate. It takes 1 hour to process a liter of drink A, 1 hour to process 1 liter of drink B, and 1 hour to process 1cl of semi-elaborate C. The mixer is available 6 hours per day.

Drink A uses 2cl of semi-elaborate C and drink B uses 1cl of semi-elaborate C. The company has 3cl of semi-elaborate C plus the amount they decide to produce every day.

The following problem model provides the optimal daily production:

**Decision variables:** -  $x_A$ : Production of drink A in liters -  $x_B$ : Production of drink B in liters -  $x_C$ : Production of semi-elaborate C in centiliters

$$x_A, x_C, x_B \in \mathbb{R}$$

### Objective function

$$\max z = 3 * x_A + 2 * x_B$$

z is the profit in euros.

### Constraints

- Availability of ingredient D in grams:

$$3 * x_A + x_B \leq 3$$

- Availability of mixer in hours:

$$x_A + x_B + x_C \leq 6$$

- Availability of semi-elaborate C in centiliters:

$$2 * x_A + x_B - x_C \leq 3$$

- Logical constraint

$$x_A, x_C, x_B \geq 0$$

A solver provides the following solution:

Optimal

Total profit is 6.00 €

The following table shows the decision variables:



j	Variables		Solu- tion	Reduced cost	Objective Coeffi- cient	Objective bound	Lower	Objective bound	Upper
A	Production (Liters)	A	0	-3	3	-inf		6	
B	Production (Liters)	B	3	0	2	1		inf	
C	Production (Liters)	C	3	0	0	-0		1.5	

The following table shows the constraints:

j	Constraint	Slack	Shadow Price	Right Hand Side	Min RHS	Max RHS
0	Availability of ingredient D (Liters)	0	2	3	0	4.5
1	Availability of mixer (hours)	0	0	6	3	inf
2	Availability of semi-elaborate (Liters)	3	0	3	0	inf

Answer the following analysis questions:

- Operational problems in the mixer, like mechanical problems, could reduce the available mixer. Based on the provided solution, how is the company able to deal with this type of problem? Motivate your response.
- The managers of the company want you to negotiate with a new supplier that could increase your daily supply of ingredient D. What would be your terms for negotiating with the new supplier, ensuring that the basic variables remain the same?
- The managers of the company would like to review the prices of both products. Analyse how changes in the prices of both products could affect the production plan and provide useful feedback to guide the revision of the prices

## Duality

- Write down the dual problem.

## Problem Extension

- Spellman's Ltd wants to evaluate the possibility of subcontracting part of the production of semi-elaborate C to an external provider, at a cost of 0.35€ per cl. With the new set-up, the daily availability of semi-elaborate C will be the sum of the semi-elaborate C produced at Spellman's plus the amount purchased from the new provider. What changes are needed in the problem to evaluate the subcontracting of semi-elaborate C? (1.5 points)

## 2.2.23 Lost in Space

### Problem Description

Will Robinson and his crew are lost in space and have crashed their spaceship in an alien planet. They need to collect metals to repair the engine of their spaceship. More specifically, the piece to repair is made of a metal alloy of chrome, aluminium, and titanium. To build the new piece, Will needs to collect 1 kilogram of chrome, 800 grams of aluminium and 1.5 kilograms of titanium. Will and his crew can gather these metals from 3 different mining locations they have identified around the camp. Will has estimated the average kg/hour that can be extracted in each location:

Mining Location	Chrome (kg/hour)	Aluminium (kg/hour)	Titanium (kg/hour)
Location 1	0.1	0.3	0.05
Location 2	0.05	0.2	0.1
Location 3	0.02	0.15	0.2

Write down a Continuous Linear Programming model to minimize the total time used to collect the materials needed to build the part needed to fix the engine.

## 2.3 Solved Exercises

In this section you have the solution to the different exercises.

### 2.3.1 Problem Definition

A gold seeker needs to find at least 2 kg of bronze and 3 kg of copper to pay his gambling debts. There are two rivers in which the miner can find these precious metals. Each day the miner spends in river A, he finds 0.3 kg of bronze and 0.3 kg of copper while each day he spends in river B, he finds 0.2 kg of bronze and 0.4 kg of copper.

*Can you help this gold seeker decide how many days to spend in each day to pay his debts as soon as possible?*

#### Problem Model

##### Decision variables

$x_A$  Days spent in river A

$x_B$  Days spent in river B

##### Objective function

$$\min z = x_A + x_B$$

##### Constraints

$$0.3 * x_A + 0.2 * x_B \geq 2 \text{ Bronze constraint}$$

$$0.3 * x_A + 0.4 * x_B \geq 3 \text{ Copper constraint}$$

### 2.3.2 The good carpenter

#### Problem definition

“The Fog Ltd” is a modern furniture company that makes tables and chairs. The selling price of a table is 30€ and the selling price of chairs is 10€. The carpenter at the company works up to 40 hours and needs 6 hours to make a table and 3 hours to make a chair. Customer demand requires that he makes at least 3 times as many chairs as tables. Tables take up four times as much storage space as chairs and there is room for at most four tables each week.

What is the optimal weekly production plan that optimises revenues?

## Solution

### Decision variables

The decision variables in the problem is the number of chairs and tables per week to manufacture:

- $x_T$ : Number of tables per week
- $x_C$ : Number of chairs per week

Let us assume that they are both continuous, i.e:  $x_T, x_C \in \mathbb{R}$

### Objective function

The objective is to maximise the weekly revenues:

$$\max z = 30 * x_T + 10 * x_C$$

$z$  is the result variable (profit in euros per week).

### Constraints

The objective function is subject to the following constraints:

- **Working time:** The working time cannot exceed 40 hours:

$$6 * x_T + 3 * x_C \leq 40$$

- **Demand:** The number of chairs must be at least 3 times the number of tables:

$$x_C \geq 3 * x_T$$

- **Storage space:** The product stock cannot exceed the storage space. There is only room for 4 tables and a chair takes 1/4 of the space of a table

$$0.25 * x_C + x_T \leq 4$$

## 2.3.3 Ski Patrol

### Problem Definition

Ski Patrol Ltd. manufactures two different models of snowboards branded Normal and Deluxe with a weekly production plan and inventory. They sell their material to local shops. The company obtains a profit of 300€ per Normal snowboard and 400€ per Deluxe snowboard. The manufacturing process consists of roughing and polishing. The Normal snowboard model requires 20 minutes of roughing and 2/3 minutes of polishing. The Deluxe snowboard model requires 40 minutes of roughing and 1 minute of polishing. The roughing equipment is available for 1000 minutes per week and the polishing equipment for 800 minutes per week. Ski Patrol has already signed a contract with a local shop called Downhill Racer to deliver 18 Normal snowboards per week.

Ski Patrol hires you to formulate a Linear Programming Problem to help them determine the best production mix to maximise benefits.

### Write a CLP to maximise profits

#### Decision Variables

$x_1$ : Units of Normal snowboards per week

$x_2$ : Units of Premium snowboards per week

#### Objective Function

$$\max z = 300 \cdot x_1 + 400 \cdot x_2$$

### Constraints

$$20 \cdot x_1 + 40 \cdot x_2 \leq 1000 \text{ Roughing process time (minutes)}$$

$$2/3 \cdot x_1 + x_2 \leq 800 \text{ Polishing process time (minutes)}$$

$$x_1 \geq 18 \text{ Contract (Demand) Units}$$

$$x_2 \geq 0 \text{ (Physical constraint)}$$

## 2.3.4 Production Mix example

### Problem Definition

Production Mix A company produces two kinds of products, P1 and P2.

The selling price of P1 is 300€ The selling price of P2 is 250€

- P1 requires 2 operator man-hours and 1 hour of machine operating time
- P2 requires 1 operator man-hours and 3 hours of machine time
- Operator man-hours cannot be higher than 40 hours per day
- Machine operating time cannot be higher than 45 hours per day
- Marketing states that P1 units per day cannot be higher than 12

**How many units of P1 and P2 should the company produce every day to maximise income?**

### Model

We want to find the maximum solution to:

$$\max Z = 300x_1 + 250x_2$$

This is our objective function. The decision variables are:

$x_1$  : units of Product 1

$x_2$  : units of Product 2

The objective function is subject to the following constraints:

$$2x_1 + x_2 \leq 40$$

$$x_1 + 3x_2 \leq 45$$

$$x_1 \leq 12$$

We define the problem in its standard form as:

$$\max Z = 300x_1 + 250x_2 + 0s_1 + 0s_2 + 0s_3$$

And the constraints as:

$$2x_1 + x_2 + s_1 = 40$$

$$x_1 + 3x_2 + s_2 = 45$$

$$x_1 + s_3 = 12$$

### 2.3.5 Problem Definition

The company Tetravaal located in Johannesburg manufactures two types of robots, Model  $P_1$  and Model  $P_2$ . The production plant is consisted of four different sections: metal machining, plastic moulding, electrical work and assembly. The metal machining section has a capacity of 7500 units of  $P_1$  or 6000 units of  $P_2$  per month.

Plastic moulding can process 5000 units of  $P_1$  or 9000 units of  $P_2$  per month.

Electrical work can process 6000 units of  $P_1$  or 7000 units of  $P_2$  per month.

In Assembly, there are two assembly lines that work in parallel, one per each robot model.

The first assembly line can process 4000 units of  $P_1$  per month

The second assembly line can process 5000 units of  $P_2$  per month

Knowing that the unitary profit of  $P_1$  is 500€ and that the unitary profit of  $P_2$  is 600€, and that both robots have a great demand and therefore all manufactured robots are sold, Michelle Bradley, CEO of Tetravaal, asks his engineering team:

Calculate the number of units of each robot that needs to be manufactured to maximise profit for the company.

#### Model

We want to maximise the company profits:

$$\max z = 500x_1 + 600x_2$$

where  $z$  represents the profits (€). The decision variables are:

$x_1$  : units of  $P_1$  per month  $x_2$  : units of  $P_2$  per month

The objective function is subject to the following constraints:

$$x_1/7500 + x_2/6000 \leq 1 \text{ Metal machining constraint}$$

$$x_1/5000 + x_2/9000 \leq 1 \text{ Plastic moulding constraint}$$

$$x_1/6000 + x_2/7000 \leq 1 \text{ Electrical work constraint}$$

$$x_1 \leq 4000 \text{ First assembly line constraint}$$

$$x_2 \leq 5000 \text{ Second assembly line constraint}$$

### 2.3.6 Making Weapons

#### Problem Definition

Din Djarin is a retired bounty hunter that now runs a small factory specialised in making classic weapons in a colony recently re-established in Mandalore. The factory manufactures two different types of weapons, swords and spears. Din uses special alloys, mixtures of metals and rare materials, which have made a name for the endurance of his weapons across the colony. The table below shows the weight in pounds of each material needed to make each type of weapon:

Metal	spear	sword
Beskar	1	0.5
Tungsten	1	1
Titanium	3	1

And the table below shows the total amount of each metal available, also in pounds:

Metal	Amount
Beskar	125
Tungsten	225
Titanium	300

The price of a spade is 40 credits and the price of a sword is 5 credits. Din has already sold 100 swords, so he needs to manufacture at least that many.

**Find the optimal number of spades and swords that maximise revenues**

### Model

The objective is to maximise the revenues obtained with the sales of both types of weapons. The objective function can be modeled as:

$$\max z = 40x_1 + 5x_2$$

where  $z$  represents the objective variable (profits) and the decision variables are:

- $x_1$ : units of spears
- $x_2$ : units of swords

Let us for now consider that they are real and non-negative.

The objective function is subject to the following constraints:

Metal availability constraints:

$$x_1 + 0.5x_2 \leq 125$$

$$x_1 + x_2 \leq 225$$

$$3x_1 + x_2 \leq 300$$

And the demand on swords:

$$x_2 \geq 100$$

## 2.3.7 Blending Craft Beer

### Problem Definition

Duff Beer is an american brewery that sells alcoholic beverages worldwide. Duff beer has a very large market share in the US. One of the new business lines of Duff Beer is in the craft beer sector. Duff beer buys 3 types of wort from local suppliers. The availability of wort (in liters) and the price for each of them is specified in Table 1.

**Table 1:** daily availability and cost of wort

Wort	Availability (liters)	Cost (€cents/liter)
Type 1	2500	120
Type 2	1200	095
Type 3	2000	150

The blend of the three types of wort must comply with the following quality requirements:

- For Beer Angels, no less than 60% of Type 3 and no more of 20% of Type 2. The price of one bottle of Beer Angel is 4.10€/s/liter.
- For Beer Beast, no less than 15% of Type 3 and no more than 60% of Type 2. The price of one bottle of Beer Beast is 2.80€/s/liter.
- For Beer Cactus, no more than 50% of type 2. The selling price is 2.45€/liter.

**Write a linear program to find the most profitable blend of wort to elaborate the three kinds of beers**

### Problem Model

#### Indices

We can define the following indices to express the problem in a compact form: -  $i$ : wort type  $i \in [1, 2, 3]$  -  $j$ : Beer brand  $j \in [A, B, C]$  where A notes Angels, B notes Beast, and C notes Cactus

#### Decision variables

The decision variables are:

$x_{ij}$ : Amount of wort of type  $i$  (1, 2, 3) in Beer  $j$ : A (Angels), B (Beast), C (Cactus), (ie  $i \in [1, 2, 3]$ ,  $j \in [A, B, C]$ )

#### Objective Function

The objective function is:

$$\max z = \sum_{i=1}^3 \sum_{j=A}^C (d_j - c_i) * x_{ij}$$

where  $d_j$  is the selling price of a bottle of beer  $j$  and  $c_i$  is the cost per liter of wort type  $i$ .

$$\max z = 410(x_{1A} + x_{2A} + x_{3A}) + 280(x_{1B} + x_{2B} + x_{3B}) + 245(x_{1C} + x_{2C} + x_{3C}) - 120(x_{1A} + x_{1B} + x_{1C}) - 95(x_{2A} + x_{2B} + x_{2C}) - 150(x_{3A} + x_{3B} + x_{3C})$$

$$\max z = 290x_{1A} + 315x_{2A} + 260x_{3A} + 160x_{1B} + 185x_{2B} + 130x_{3B} + 125x_{1C} + 150x_{2C} + 95x_{3C}$$

#### Constraints

Subject to the following constraints:

**Availability constraints** The availability constraints can be expressed in a compact form as:

$$\sum_j x_{ij} \leq a_i \forall i$$

Where  $a_i$  is the availability of wort type  $i$ :

$$x_{1A} + x_{1B} + x_{1C} \leq 2500$$

$$x_{2A} + x_{2B} + x_{2C} \leq 1200$$

$$x_{3A} + x_{3B} + x_{3C} \leq 2000$$

**Quality constraints** The proportion of a given type of wort  $i'$  in a type of beer  $j'$  can be expressed as:

$$\frac{x_{i'j'}}{\sum_i (x_{ij'})} \forall i', j'$$

If we compare this to the minimum proportion  $R_{min_{i'j'}}$  we obtain:

$$\frac{x_{i'j'}}{\sum_i (x_{ij'})} \geq R_{min_{i'j'}} \forall i', j'$$

Or

$$x_{i'j'} \geq R_{min_{i'j'}} * \sum_i (x_{ij'}) \forall i', j'$$

If we take the denominator to the RHS. We can obtain a similar expression for the maximum proportion  $R_{max_{i'j'}}$ :

$$x_{i'j'} \geq R_{max_{i'j'}} * \sum_i (x_{ij'}) \forall i', j'$$

From these expressions, we can derive the constraints for every type of beer:

**Angels quality requirement constraints**

$$x_{3A} \geq 0.6(x_{1A} + x_{2A} + x_{3A})$$

$$x_{2A} \leq 0.2(x_{1A} + x_{2A} + x_{3A})$$

**Beast quality requirement constraints**

$$x_{3B} \geq 0.15(x_{1B} + x_{2B} + x_{3B})$$

$$x_{2B} \leq 0.6(x_{1B} + x_{2B} + x_{3B})$$

**Cactus quality requirement constraints**

$$x_{2C} \leq 0.5(x_{1C} + x_{2C} + x_{3C})$$

### 2.3.8 Blending Problem

**Problem definition**

The Kahuna company manufactures sausages using three kinds of meat. The relevant information about the ingredients is provided in the table below:

Ingredient	Cost (€/kg)	Availability (kg)
Pork	4.32	30
Wheat	2.46	20
Starch	1.86	17

The company makes two types of sausages: \* Economy ( $\geq 40\%$  Pork) \* Premium ( $\geq 60\%$  Pork)

One sausage is 50 grams (0.05 kg)

According to government regulations, the most starch we can use in our sausages is 25%

We have a contract with a butcher, and have already purchased 23 kg pork, that will go bad if it's not used.

We have a demand for 350 economy sausages and 500 premium sausages.

**Write a linear program to figure out how to most cost effectively blend our sausages.**

Let's model our problem

$pe$  = Pork in the economy sausages (kg)

$we$  = Wheat in the economy sausages (kg)

$se$  = Starch in the economy sausages (kg)

$pp$  = Pork in the premium sausages (kg)

$wp$  = Wheat in the premium sausages (kg)

$sp$  = Starch in the premium sausages (kg)

We want to minimise costs such that:



$$\text{Cost} = 4.32(pe + pp) + 2.46(we + wp) + 1.86(se + sp)$$

With the following constraints:

$$pe + we + se = 350 * 0.05$$

$$pp + wp + sp = 500 * 0.05$$

$$pe \geq 0.4(pe + we + se)$$

$$pp \geq 0.6(pp + wp + sp)$$

$$se \leq 0.25(pe + we + se)$$

$$sp \leq 0.25(pp + wp + sp)$$

$$pe + pp \leq 30$$

$$we + wp \leq 20$$

$$se + sp \leq 17$$

$$pe + pp \geq 23$$

### 2.3.9 Petroleum Blending (Solved)

#### Problem Definition

Harry Stamper is an engineer working on a petroleum company. The company produces three grades of motor oil – super, premium, and extra. The three grades are made from the same 3 ingredients, named component 1, component 2 and component 3. The company wants to determine the optimal mix of the 3 components in each grade of motor oil that will maximize profit.

The maximum availability of each component and their cost per barrel are as follows:

Component	Max barrels available / day	Cost / barrel
Component 1	4.500	12€
Component 2	2.700	10€
Component 3	3.500	14€

To ensure the appropriate blend, each grade must have a minimum amount of component 1 plus a combination of other components as follows:

Grade	Component 1	Component 2	Component 3	Selling price/barrel
Super	At least 50%	No more than 30%	.	23€
Premium	At least 40%	.	No more than 25%	20€
Extra	At least 60%	At least 10%	.	18€

The company wants to make at least 3000 barrels of each blend. *Formulate an LP to help Harry Stamper find the optimal blend that maximises profit.*

### Problem Model

The decision variables are:

$x_{ij}$  = amount in barrels of component  $i$  in grade  $j$

Objective function

$$\max z = 23 * (x_{11} + x_{21} + x_{31}) + 20 * (x_{12} + x_{22} + x_{32}) + 18 * (x_{13} + x_{23} + x_{33}) - 12 * (x_{11} + x_{12} + x_{13}) - 10 * (x_{21} + x_{22} + x_{23}) - 14 * (x_{31} + x_{32} + x_{33})$$

Constraints

$$x_{11} \geq 0.5 * (x_{11} + x_{21} + x_{31}) \text{ Component 1 requirement in Super}$$

$$x_{21} \leq 0.3 * (x_{11} + x_{21} + x_{31}) \text{ Component 2 requirement in Super}$$

$$x_{12} \geq 0.4 * (x_{12} + x_{22} + x_{32}) \text{ Component 1 requirement in Premium}$$

$$x_{32} \leq 0.25 * (x_{12} + x_{22} + x_{32}) \text{ Component 3 requirement in Premium}$$

$$x_{13} \geq 0.6 * (x_{13} + x_{23} + x_{33}) \text{ Component 1 requirement in Extra}$$

$$x_{23} \geq 0.1 * (x_{13} + x_{23} + x_{33}) \text{ Component 2 requirement in Extra}$$

$$x_{11} + x_{12} + x_{13} \leq 4500 \text{ Component 1 availability}$$

$$x_{21} + x_{22} + x_{23} \leq 2700 \text{ Component 1 availability}$$

$$x_{31} + x_{32} + x_{33} \leq 3500 \text{ Component 1 availability}$$

$$x_{11} + x_{21} + x_{31} \geq 3000 \text{ Super minimum production}$$

$$x_{12} + x_{22} + x_{32} \geq 3000 \text{ Premium minimum production}$$

$$x_{13} + x_{23} + x_{33} \geq 3000 \text{ Extra minimum production}$$

### Analysis questions

This is the solution of the problem obtained with Gurobi solver in Petroleum Blending (Solved CBC):

### Decision Variables

	Variables	Solu- tion (GRB)	Reduced cost (GRB)	Objective Coefficient (GRB)	Objective Lower bound (GRB)	Objective Upper bound (GRB)
(Super, Component 1)	Quantity_in_bar- rels_('Super','Compo- nent_1')	1500.000	0.000	11.000	8.000	Inf
(Super, Component 2)	Quantity_in_bar- rels_('Super','Compo- nent_2')	450.000	0.000	13.000	13.000	13.000
(Super, Component 3)	Quantity_in_bar- rels_('Super','Compo- nent_3')	1050.000	0.000	9.000	9.000	9.000
(Premium, Component 1)	Quantity_in_bar- rels_('Pre- mium','_Compo- nent_1')	1200.000	0.000	8.000	-Inf	11.000
(Premium, Component 2)	Quantity_in_bar- rels_('Pre- mium','_Compo- nent_2')	1050.000	0.000	10.000	-Inf	10.000
(Premium, Component 3)	Quantity_in_bar- rels_('Pre- mium','_Compo- nent_3')	750.000	0.000	6.000	6.000	10.800
(Extra, Component 1)	Quantity_in_bar- rels_('Extra','_Compo- nent_1')	1800.000	0.000	6.000	-Inf	17.333
(Extra, Component 2)	Quantity_in_bar- rels_('Extra','_Compo- nent_2')	1200.000	0.000	8.000	8.000	25.000
(Extra, Component 3)	Quantity_in_bar- rels_('Extra','_Compo- nent_3')	0.000	-0.000	4.000	-Inf	4.000

### Constraints

idx	Constraint	Right Side	Hand	Shadow Price	Slack	Min RHS	Max RHS
0	Component_1_requirement_in_super	0.00		-18.00	0.00	-850.00	0.00
1	Component_2_requirement_in_super	0.00		0.00	450.00	-450.00	Inf
2	Component_1_requirement_in_pre- mium	0.00		-18.00	0.00	-450.00	0.00
3	Component_3_requirement_in_pre- mium	0.00		0.00	0.00	-450.00	450.00
4	Component_1_requirement_in_Extra	0.00		-18.00	0.00	-450.00	0.00
5	Component_2_requirement_in_Extra	0.00		0.00	-900.00	-Inf	900.00
6	Component_1_availability	4500.00		20.00	0.00	4500.00	6200.00
7	Component_2_availability	2700.00		4.00	0.00	2250.00	3150.00
8	Component_3_availability	3500.00		0.00	1700.00	1800.00	Inf
9	Super_minimum_production	3000.00		0.00	-0.00	-Inf	3000.00
10	Premium_minimum_production	3000.00		-1.20	0.00	-0.00	3000.00

- Which is the total profit obtained by the company with the Extra grade with the solution provided by the solver?

Just multiply the value of every basic (non-zero) decision variable times the objective function coefficient.

- Which is the maximum value that the profit per barrel of component 2 in the Extra grade can take without changing the base of the optimal solution?

The profit per barrel of component two in the extra grade is 8.00 (look at the second last row of the first table). The maximum value that it can take is the upper bound of the objective function in that row, which is 25. The profit per barrel of component 2 could change from 8.00 to 25 without changing the base. Since there are no changes in the constraint, there are no changes in the feasible region and the optimal solution will be the same point and same quantities. Therefore, if we were able to raise the profit by 1 eur per barrel of component 2 in extra grade, the total profit (objective function) will increase 12000 (because this is the quantity of this component in the extra grade in the optimal solution).

- If the cost of the barrel of component 3 increased 1€, will the optimal composition change? Motivate your response

In this case an increase of the cost of barrel of component 3 means that the profit per component 3 in the three grades is going to decrease by 1 eur. Since the lower bound of the profit per component 3 in the “Super” grade is equal to the objective function coefficient, there will be a change in the optimal composition.

- Do we use every barrel available of every component in the optimal solution? Or do we have any surplus availability for any component? Motivate your response

If we look at the components availability, we can clearly see that we have a slack of 1700 barrels of component 3.

- The provider of Component 2 may have a problem with the delivery and the availability may be reduced in 400 barrels. Will this shortage affect the optimal composition? Motivate your response\*

If the component 2 is short in 4000 barrels (from 2700 to 2300) the availability will still be higher than the lower bound of the component 2 available constraint which is 2250, so this shortage in principle will not affect the optimal solution.

- Grace wants to use the results to renegotiate the terms and conditions with the suppliers of the different components. Which component is a better candidate to increase the availability? How would you use the model to negotiate new terms with the provider, assuming you do not want any change in the base of the optimal solution? Motivate your response

Now, this is a quite an interesting question. If we were to increase the availability, we can first discard component 3, because we have the aforementioned slack, which means that we are not using all the available material, so the candidates are component 1 and 2. Now, comparing both components, component 1 has a much larger shadow price, which means that for every change of one unit on the right hand side, we will have an increase of 20 eur in the profit. So this is definitively the best candidate. We could try to negotiate the terms and conditions and try to tell the provider that we want more units of this component. We could use up to 6200 units (upper right hand side) without changes in the optimal composition. Now, since we are buying more units, we may as well ask for a lower price. If we look at the profits of component 1 in the first table, we could increase the profits of Premium, component 1 by 3 euros (from 8 to 11 which is the upper bound) so we could try to renegotiate the price from 12 down to 9.

### 2.3.10 Fasting Diet

#### Problem I

You have read an article about the benefits of fasting. This article publishes a series of findings around the fasting diet method. Specifically, the article states that:

- Decreasing the number of meals to three, two heavier meals and a light meal in between has significant benefits for the health
- The study reveals that the optimal amount of daily calories for an adult is higher than 1800
- and also lower than 2200 calories

- The study also reveals that the light meal should not represent more than 10% of the number of daily calories
- And that the last meal should not provide more than 700 calories

You think you can design a Linear Programming Problem where the decision variables represent the calories of each meal to find an optimal diet that fulfils all these requirements, while at the same time provide the maximum number of calories. For such a model, define:

**a. The objective function**

Maximise the calories intake in the three meals:

$$\max z = x_1 + x_2 + x_3$$

where: -  $x_1$ : Calories of first meal

- $x_2$ : Calories of second meal (light meal)
- $x_3$ : Calories of third meal

**b. The constraints** Now, given the results of the study:

Optimal daily amount

$$x_1 + x_2 + x_3 \geq 1800$$

$$x_1 + x_2 + x_3 \leq 2200$$

Calories of light meal (less than 10% of total calories):

$$x_2 \leq 0.1(x_1 + x_2 + x_3)$$

Calories of third meal (less than 700):

$$x_3 \leq 700$$

**c. The dual problem**

We can write the problem as:

$$\max z = x_1 + x_2 + x_3$$

s.t.

$$x_1 + x_2 + x_3 \geq 1800$$

$$x_1 + x_2 + x_3 \leq 2200$$

$$-0.1 \cdot x_1 + 0.9 \cdot x_2 - 0.1 \cdot x_3 \leq 0$$

$$0 \cdot x_1 + 0 \cdot x_2 + 1 \cdot x_3 \leq 700$$

Hence, the dual becomes:

$$\min z = 1800 \cdot u_1 + 2200 \cdot u_2 + 0 \cdot u_3 + 700 \cdot u_4$$

s.t

$$u_1 + u_2 - 0.1 \cdot u_3 + 0 \cdot u_4 \geq 1$$

$$u_1 + u_2 + 0.9 \cdot u_3 + 0 \cdot u_4 \geq 1$$

$$u_1 + u_2 - 0.1 \cdot u_3 + u_3 \geq 1$$

Since the primal is a maximisation problem,  $u_1 \leq 0$  because the first constraint is of type  $\geq$ . On the other hand,  $u_2, u_3, u_4 \leq 0$  because the rest of constraints are of type  $\leq$ .

### 2.3.11 Bonuses and merits

#### Problem definition

A company pays a yearly bonus to employees based on the following indicators:

1. **Performance:** Performance is evaluated by the direct manager using a quantitative method. The indicator is labelled as  $CM_1$  and measured in a scale from 1 to 5.
2. **Involvement:** Involvement in social media, events, coaching and other voluntary activities. Involvement is measured by the human resource department using a quantitative method. The indicator is labelled as  $CM_2$  and measured in a scale from 1 to 5.
3. **Professional development:** Professional development evaluates the accomplishment of training programs and courses suggested by the proximity manager. The indicator is labelled as  $CM_3$ .

This year, the total score obtained among all employees is:

$$\sum(CM_1) = 1625 \text{ Total points in performance}$$

$$\sum(CM_2) = 1409 \text{ Total points in involvement}$$

$$\sum(CM_3) = 1387 \text{ Total points in professional development}$$

The company wants to calculate bonuses using a linear expression weighting the three indicators. The board established the following rules:

- No employee should receive more than 5000€ in bonuses
- The value of performance is twice the value of involvement and professional development
- The total budget for bonuses this year is 1.5M€

Formulate an LPP that determines the optimal formula to calculate employee bonuses.

Let us first define the decision variables, as:

- $x_1$ : Coefficient in the bonuses linear expression for the performance KPI
- $x_2$ : Coefficient in the bonuses linear expression for the involvement KPI
- $x_3$ : Coefficient in the bonuses linear expression for the professional development KPI

In this problem, the objective is to find the optimal values for these coefficients so that we spent the budget for bonuses in an optimal way. Let us define the objective function as the sum of the coefficients:

$$\max z = x_1 + x_2 + x_3$$

This objective function does not have an economic meaning for the company, it is introduced to formulate the problem. The type of objective must be of type maximise (otherwise the solution will always be zero), and we do not use any coefficients so that the solution is not biased towards any indicator.

The constraints will be:

No employee should receive more than 5000€ in bonuses, therefore, with the maximum grading of 5 for all indicators, the bonus must be less or equal than 5000:

$$5 * x_1 + 5 * x_2 + 5 * x_3 \leq 5000$$

The bonus for performance is twice the bonus for the other two indicators:

$$x_1 = 2 * x_2$$

$$x_1 = 2 * x_3$$

The total amount spent cannot exceed the available budget. Since we know the total scores by all employees:

$$1625 * x_1 + 1409 * x_2 + 1387 * x_3 \leq 1.5 * 10^6$$

### 2.3.12 Resourcing: Solar panel cells

#### Problem Definition

Weyland Corp is a British firm that manufactures solar panel cells. This firm supplies three different types of cell called: fast, normal and ultra. There are three operations involved in the manufacturing process. The following table describes the hours/month required to manufacture each model:

**Table 1:** hours/month requirements

Operation	Fast (hours/cell)	Normal (hours/cell)	Ultra (hours/cell)
1	1	3	2
2	2	0	3
3	1	4	0

And the following table describes the total hours available for each operation:

**Table 2:** Total hours/month available

Operation	Hours/Month
1	400
2	600
3	600

The profit per each model is specified in the following table: **Table 3:** Unit of profit per model

Model	Profit/Unit
Fast	30
Normal	20
Ultra	40

#### Problem Model

##### Decision variables

The decision variables are:

$x_1$  :Units of Fast cells / month

$x_2$  :Units of Normal cells / month

$x_3$  :Units of Ultra cells / month

### Objective Function

The objective function is:

$$\max z = 30x_1 + 20x_2 + 40x_3$$

### Constraints

Subject to the following constraints:

$$x_1 + 3x_2 + 2x_3 \leq 400$$

$$2x_1 + 0x_2 + 3x_3 \leq 600$$

$$x_1 + 4x_2 + 0x_3 \leq 600$$

### Solution

The following solution has been obtained using PuLP and the Gurobi solver.

Total profit is 9666.67 €

The following table shows the values of the decision variables, the reduced costs and the sensitivity analysis of the objective coefficients:

	Variables	Solu- tion	Reduced cost	Objective Coeffi- cient	Objective bound	Lower	Objective bound	Upper
fast	cells_fast	300	0	30	24.44		inf	
nor- mal	cells_nor- mal	33.33	0	20	-0		90	
ultra	cells_ultra	0	-8.33	40	-inf		48.33	

The following table shows the slack and shadow prices of the constraints, together with a sensitivity analysis of the RHS:

	Constraint	Right Hand Side	Shadow Price	Slack	Min RHS	Max RHS
0	Operation_1	400	6.67	0	300	525
1	Operation_2	600	11.67	0	0	800
2	Operation_3	600	0	166.67	433.33	inf

### Question 1

Peter Weyland believes that, for the reputation of the company, Weyland should manufacture at least 40 ultra cells per month. If this plan goes ahead, how would it affect the current profit?

In the current optimal solution, ultra cells are not part of the solution and therefore using resources to manufacture this type of cells will decrease the profits. The reduced costs for ultra cells are -8.33, meaning that the profits are going to decrease 8.33 for every extra cell that is manufactured. Therefore, 40 extra units of ultra would lower the profits: **333.33**



**Question 2**

The engineer informed management that due to maintenance operations, the capacities of Operation 1, 2, and 3 have changed to 360, 600 and 500 respectively. How would this affect the profit?

We need to analyse the cases one by one. Regarding operation 1, the slack is 0 and the shadow price is 6.67, meaning that having 1 unit less will lower the profits 6.67€. The minimum RHS is 300, which is lower than 360 and therefore there is no change in the basic solution. With this, A decrease of 40 units will cause a decrease in the budget of 266.67€.

There is no impact on the capacity of Operation 2, the RHS is still 600.

Operation 3 has a slack of 166.67, which is higher than the 100 decrease due to maintenance operations, so there will be no impact on the profits.

**Question 3**

Mr. Weyland insisted that the company needs to change the strategy towards producing ultra fast cells to position Weyland Corp in this segment. The company reviewed the prices of the ultra fast cells and changed them to 30€, 10€ and 50€ per fast, normal and ultra cells. With the new prices, would it be profitable to produce ultra fast cells?

We need to verify the bounds for the objective coefficients for the three variables, case by case.

For fast cells, there is no change in the price.

For normal cells, the new price of 10€ is higher than the lower bound of the solution, therefore, there will be no change caused by these decrease.

For ultra cells, the new price is over the maximum lower price, so this change will cause a change in the basic solution:

	Vari-ables	Solu-tion	Reduced cost	Objective Coeffi-cient	Objective bound	Lower	Objective bound	Upper
ul-tra	cells_ul-tra	0	-8.33	40	-inf		<b>48.33</b>	

Using the tableau we can verify that the entering variable is the ultra cell and conclude that it is profitable to produce ultra fast cells with the new prices.

**Question 4**

What is the impact in the new profit if we opted for this new prices?

We need to define the new problem and solve it. Using PuLP and CBC, the solution obtained is: - Total profit is **10000.00** € per month

Therefore, the profit would change in **333.33** € per month

**Question 5**

The engineer is studying a new type of fast cells that would require 2 hours of operation 1, 0 hours of operation 2 and 1 hour of operation 3. The estimated market price is 20€. How would this affect the production plan?

We need to define a new problem model to introduce the new standard cell type and solve it. The following solution has been obtained with PuLP and Gurobi.

The total profit is **10000.00** € per month

The following tables show the values obtained:

	Decision Variable	Solution value	Unit profit	Reduced cost	Lower bound	Upper bound
0	cells_fast	300	30	0	23.3333	inf
1	cells_normal	0	20	-10	-inf	30
2	cells_standard	50	20	0	13.3333	60
3	cells_ultra	0	40	-10	-inf	50

The following table shows the problem constraints:

	Con- straint	Left-hand side	Sense	Right-hand side	Slack	Shadow price	Lower bound	Upper bound
0	Opera- tion_1	400	<=	400	0	10	300	600
1	Opera- tion_2	600	<=	600	0	10	0	800
2	Opera- tion_3	400	<=	600	200	0	400	inf

### Question 6

Mr. Weyland thinks that the operation time for Operation 2 for the normal cells can be cut to only 3 hours per cell. How would this improve the production plan?

This will mean a change in a LHS coefficient of a constraint, we would need to build a new model with this new coefficient and solve it again with a solver.

## 2.3.13 The chilling adventures of Sabrina

### Problem definition

Spellman's Ltd is a company that manufactures chilling soft drinks. They want to manufacture two types of drinks A, and B. Both beverages use a semi-elaborate C, another expensive ingredient D and other ingredients that are not relevant for production planning. Sabrina is a young student of engineering and management doing an internship at Spellman's. She needs to formulate a Continuous Linear Program to configure the optimal daily production plan for the company.

The selling price of drink A is 3€/liter and the selling price of drink B is 2€/liter.

1 liter of drink A uses 3 grams of ingredient D. A liter of drink B uses 1 gram of ingredient D. There are only 3 grams of ingredient D available per day.

The factory only has one mixer to elaborate both drink types and the semi-elaborate. It takes 1 hour to process a liter of drink A, 1 hour to process 1 liter of drink B, and 1 hour to process 1cl of semi-elaborate C. The mixer is available 6 hours per day.

Drink A uses 2cl of semi-elaborate C and drink B uses 1cl of semi-elaborate C. The company has 3cl of semi-elaborate C plus the amount they decide to produce available per day.

1. Write a Continuous Linear Problem to help Sabrina design the optimal production plan that maximises revenues for the company.

**Decision variables:** -  $x_A$ : Production of drink A in liters -  $x_B$ : Production of drink B in liters -  $x_C$ : Production of semi-elaborate C in centiliters

$$x_A, x_C, x_B \in \mathbb{R}$$

**Objective function**

$$\max z = 3 * x_A + 2 * x_B$$

z is the profit in euros.

**Constraints s.t.**

- Availability of ingredient D in grams:

$$3 * x_A + x_B \leq 3$$

- Availability of mixer in hours:

$$x_A + x_B + x_C \leq 6$$

- Availability of semi-elaborate C in centiliters:

$$2 * x_A + x_B - x_C \leq 3$$

- Logical constraint

$$x_A, x_C, x_B \geq 0$$

**2. Write the dual problem** The dual is:

$$\min z = 3 * u_1 + 6 * u_2 + 3 * u_3$$

s.t.

$$3 * u_1 + u_2 + 2 * u_3 \geq 3$$

$$u_1 + u_2 + u_3 \geq 2$$

$$u_2 - u_3 \geq 0$$

**3. Given the following solution:**

- 0 Liters of drink A
- 3 Liters of drink B
- 3 cl of semi-elaborate C

Verify the solution. Is the solution feasible? What are the values of the slack variables?

We plug in the values of the solution in our constraints: - Availability of ingredient D in grams:

$$3 * x_A + x_B \leq 3$$

$$3 * 0 + 3 \leq 3$$

$$3 + s_1 = 3$$

$$s_1 = 0$$

- Availability of mixer in hours:

$$x_A + x_B + x_C \leq 6$$

$$0 + 3 + 3 \leq 6$$

$$6 + s_2 = 6$$

$$s_2 = 0$$

- Availability of semi-elaborate C in centiliters:

$$2 * x_A + x_B - x_C \leq 3$$

$$2 * 0 + 3 - 3 \leq 3$$

$$0 + s_3 = 3$$

$$s_3 = 3$$

- Logical constraint

$$s_1, s_2, s_3 \geq 0$$

The solution is feasible because it meets all constraints and slack variables are non-negative.

4. Use complementary slackness to find the dual solution corresponding to this vertex. Is the dual solution feasible? Is the solution optimal? Motivate your response.

By complementary slackness, since  $s_3 = 3$ , we know that  $u_3 = 0$ , and also, since  $x_C$  is greater than zero, we know that the third constraint of the dual is binding. By plugging this information into the third constraint we obtain:

$$u_2 - u_3 = 0$$

$$u_2 = 0$$

We can plug this value into the second constraint of the dual, which is also binding since  $x_B$  is non-zero, to obtain:

$$u_1 + u_2 + u_3 = 2$$

$$u_1 = 2$$

All values are non-negative, so the solution is feasible and since both primal and dual are feasible, the solution is optimal.

Gurobi provides the following solution:

Optimal

Total profit is 6.00 €

The following table shows the decision variables:

j	Variables	Solution (GRB)	Reduced cost (GRB)	Objective Coefficient (GRB)	Objective bound (GRB)	Lower	Objective bound (GRB)	Upper
A	units_A	0	-3	3	-inf		6	
B	units_B	3	0	2	1		inf	
C	units_C	3	0	0	-0		1.5	

The following table shows the constraints:

j	Constraint	Slack	Shadow Price	Right Hand Side	Min RHS	Max RHS
0	Availability_of_ingredient_A	0	2	3	0	4.5
1	Availability_of_mixer_hours	0	0	6	3	inf
2	Availability_of_semi_elaborate	3	0	3	0	inf

### 2.3.14 Those damn interns

#### Problem Definition

You are the production manager of a company. You have used the following programming model to define the optimal production plan. The problem is expressed in the standard form. Objective function:

$$\max z = 3 * x_1 + 4 * x_2 - M * a_1 + 0 * s_2 + 0 * s_3$$

s.t

$$x_1 + 0 * x_2 + a_1 + 0 * s_2 + 0 * s_3 = 18$$

$$20 * x_1 + 40 * x_2 + 0 * a_1 + s_2 + 0 * s_3 = 1000$$

$$26.67 * x_1 + 20 * x_2 + 0 * a_1 + 0 * s_2 + s_3 = 800$$

where M is a very large number and artificial and slack variables are indexed in order of appearance.

The intern submitted the following data describing the solution obtained by a solver. This solver provides the solution of both the primal and the dual and a sensitivity analysis in a table, but your intern handed you the following handwritten report:

X1 -> Solution value: 18 Reduced Cost: ? Status: basic Objective coefficient: ? Objective Lower Bound: -Inf Objective Upper Bound: 5.334

X2 -> Solution value: 15.997 Reduced Cost: ? Status: ? Objective coefficient: 4 Objective Lower Bound: 2.2497 Objective Upper Bound: Inf

A1-> RHS= 18 Slack = 0 Sense = ? Shadow price= -2.334 Min RHS= 17.9964 Max RHS= 29.9963

S2-> RHS= 1000 Slack = 0.12 Sense = less or equal Shadow price= 0 Min RHS= 999. Max RHS= Inf

S3-> RHS= ? Slack = ? Sense = less or equal Shadow price= 0 Min RHS= 480.06 Max RHS= 800.06

**a.** You analyse the solution and notice that there are some missing values noted as ? (You are going to fire this intern!). Write down the values missing in the solution. Motivate your response in each case.

For decision variable  $x_1$  Since the value is non-zero (it is a basic decision variable), by complementary slackness, the reduced cost is 0. The objective function indicates that the objective coefficient is 3.

For decision variable  $x_2$  Also, by complementary slackness, the reduced cost is 0 and since the value is greater than zero, the decision variable is basic.

For slack,  $a_1$  If we note the coefficient in the objective function, (-M) we know that this is an artificial variable that must be always 0 and the sense is 'equal'.

For  $s_3$ , The Right Hand Side for the third constraint is 800 looking at the problem.

We can calculate the slack by substituting the values of X1 and X2 in the expression:

$$26.67X_1+20X_2+0A_1+0S_2+S_3=800 \quad 26.67 \cdot 18 + 20 \cdot 15.997 + S_3 = 800 \quad S_3 = 0$$

We could have also argued that since the solution is a Feasible Basic Solution, only 3 decision variables (in this case X1, X2, and S2) can be basic (as many as constraints) and therefore S3 must be non-basic (0).

**b.** Which of the two decision variables has a larger contribution to the optimal solution? Motivate your response (1 point).

If we multiply the value times the objective function:

$$3 \cdot x_1 = 3 \cdot 18 = 54$$

$$4 \cdot x_2 = 4 \cdot 15.997 = 63.998$$

Clearly  $x_2$  has a larger contribution to the objective function

c. The first constraint determines the value of one of the decision variables ( $x_1$ ). What would happen to the objective function if the right hand side of this constraint changes to 19? and to 31? Motivate your response

The maximum RHS for the first constraint is 29.993. If the RHS changes to 19, the basic decision variables would remain the same, although the values in the optimal solution might change. If the value changes to 31 then the basic decision variables would certainly change.

d. What would happen to the solution if the objective function coefficient of X1 changes to 5? And what if the coefficient of X2 changes to 3? Motivate your response

The Upper bound for the objective coefficient of  $x_1$  is 5.334 and therefore, the solution would not change if the objective 5 increased to 5.

The lower bound for the objective coefficient of  $x_2$  is 2.2497 and therefore, the solution would not change if the coefficient decreases down to 3.

### 2.3.15 A software bug

#### Problem Definition

You are using a legacy software to solve a production mix Continuous Linear Programming (CLP) problem in your company where the objective is to maximise profits. The following table provides the solution of the primal, dual and a sensitivity analysis for the three decision variables that represent the quantities to produce of each product

Variables	Solution	Reduced cost	Objective Coefficient	Lower bound	Upper bound
$x_1$	300.00	?	30.00	24.44	Inf
$x_2$	33.33	?	20.00	-0.00	90.00
$x_3$	0	-8.33	40.00	-Inf	48.33

Answer the following questions:

a. Notice that there are some values missing (a bug in the software shows a ? sign instead of a numerical value, remember to use Python the next time around). Fill the missing values and explain your decision

The reduced costs of  $x_1$  and  $x_2$  must be zero, since the two variables are basic, the reduced cost is zero.

b. According to the provided solution, which of the three products has the highest impact in your objective function? Motivate your response

The product of the objective coefficient times the solution determines the impact in the objective function. In this case, the variable that has the highest impact in the objective function is  $x_1$

c. What does the model tell you about product  $x_3$ ? Is it profitable to manufacture under the current conditions modeled in the problem? Provide quantitative values to motivate your response

In this solution, it is not profitable to manufacture  $x_3$ . In fact, manufacturing 1 unit would have a negative impact of 8.33 in the objective function.

d. Recall that in this type of problem, the objective coefficient represents the marginal profit per unit of product. What would happen if the objective function of variable  $x_3$  is increased over 50? Describe what would be the impact in the obtained solution

The upper bound of the objective coefficient is 48.33, meaning that if the objective coefficient goes over this value, we would experience a change in the FSB. Most likely, variable The objective coefficient would need to increase up to 48.33 to make this product profitable  $x_3$  would enter the solution.

The following table represents the value obtained for the decision variables related to the constraints:

Constraint	Right Hand Side	Shadow Price	Slack	Min RHS	Max RHS
$s_1$	400.00	6.67	0.00	300.00	525.00
$s_2$	600.00	11.67	0.00	0.00	800.00
$s_3$	600.00	0.00	166.67	433.33	Inf

Bearing in mind that the constraints represent the availability of 3 limited resources (operating time in minutes) and that the type of constraints is in every case “less or equal” answer the following questions:

**e.** Which decision variables are basic? How many are there and, is this result expected? Motivate your response

The basic decision variables are  $x_1$ ,  $x_2$ , and  $s_3$ , as many basic decision variables as constraints, since this is a Feasible Basic Solution (FBS).

**f.** Imagine that you had to cut down costs by reducing the availability of the limited resources in the model. Which one would you select? How much could you cut down without changing the production mix? Motivate your response

If the Right Hand Sides represent availability of resources, we could reduce the availability of the third constraint in 166.7 units (slack) without any change in the base.

**g.** Now, with the money saved, you could invest to increase the availability of other limited resources. Again, indicate which one would you select and by how much you would increase the availability of this resource without changing the production mix. Motivate your response

The best option is to invest in the resource of the second constraint, since it has the largest shadow price. The RHS can increase up to 800 units without a change in the base.

**h.** How do the columns of the two tables relate to the primal and dual problem? Moreover, for each column, not considering upper and lower bounds, write down a brief description of each column and the relationship it has with the primal and dual problem

In the first table, the column solution contains the values of the decision variables of the primal and the column reduced costs represents the slack variables of the dual. In the second table, the column slack represents the values of the slack variables in the primal solution and the column shadow prices, represent the values of the decision variables of the dual.

## 2.3.16 Invincible Nutritional Supplements

### Problem Definition

Invincible Ltd makes two types of nutritional supplements. The recipe of both supplements is based on three ingredients for which the company has limited availability. The table below shows the daily availability of the three ingredients and their costs:

**Table 1:** daily availability and cost of ingredients

Ingredient	Availability (grams)	Cost (€cents/gram)
1	1500	85
2	1300	90
3	2000	125

The selling prices are 3€/gram for supplement Awesome (product A) and 3.5€/gram for supplement Beast (product B).

The recipe must comply with the following requirements:

- Both products must contain at least 25% of ingredient 3.
- Product B must have at least 10% more of ingredient 2 than product A.

- The minimum quantity of any ingredient in any product is 10%

The factory has a production line that operates for 10 hours per day. The production speed of the line is 300 grams per hour for product A and 175 grams per hour for product B.

### Indices

We can define the following indices to express the problem in a compact form: - i: Ingredient  $i \in [1, 2, 3]$

- j: Product  $j \in [A, B]$

### Decision variables

The decision variables are:

$x_{ij}$ : Amount of ingredient i (1, 2, 3) in product j: A, B (ie  $i=[1,2,3]$ ,  $j=[A,B]$ ) in grams

### Objective Function

The objective function is:

$$\max z = \sum_{i=1}^3 \sum_{j=A}^B (d_j - c_i) * x_{ij}$$

where  $d_j$  is the selling price of a gram of product j and  $c_i$  is the cost of ingredient i.

$$\max z = (3-0.85)*x_{1A} + (3-0.9)*x_{2A} + (3-1.25)*x_{3A} + (3.5-0.85)*x_{1B} + (3.5-0.9)*x_{2B} + (3.5-1.25)*x_{3B}$$

$$\max z = 2.15 * x_{1A} + 2.1 * x_{2A} + 1.75 * x_{3A} + 2.65 * x_{1B} + 2.6 * x_{2B} + 2.25 * x_{3B}$$

### Constraints

Subject to the following constraints:

Availability constraints The availability constraints can be expressed in a compact form as:

$$\sum_j x_{ij} \leq a_i \quad \forall i$$

$$x_{1A} + x_{1B} \leq 1500$$

$$x_{2A} + x_{2B} \leq 1300$$

$$x_{3A} + x_{3B} \leq 2000$$

Both products must contain at least 25% of ingredient 3

$$x_{3A} \geq 0.25 * (x_{1A} + x_{2A} + x_{3A})$$

$$x_{3B} \geq 0.25 * (x_{1B} + x_{2B} + x_{3B})$$

Product B must have at least 10% more of ingredient 2 than product A:

$$x_{2B} \geq 1.1(x_{2A})$$

The minimum quantity of any ingredient in any product is 10%:

??

Production capacity constraint:

??

A solver provides the following tables with the solution of the primal and the dual:



Variables	Solution (GRB)	Reduced cost (GRB)	Objective Coefficient (GRB)	Objective Lower bound (GRB)	Objective Upper bound (GRB)
Grams of ingredient 1 in product A	1460.65	0	2.15	1.75	132.4
Grams of ingredient 1 in product B	39.35	0	2.65	-inf	3.58
Grams of ingredient 2 in product A	232.54	0	2.1	-inf	2.63
Grams of ingredient 2 in product B	255.79	0	2.6	-inf	3.09
Grams of ingredient 3 in product A	632.2	0	1.75	1.61	2.15
Grams of ingredient 3 in product B	98.38	0	2.25	-inf	3.05

idx	Constraint	Right Side	Hand	Shadow Price	Slack	Min RHS	Max RHS
0	Ingredient 1 availability	1500		0.4	0	271.89	1550.85
1	Ingredient 2 availability	1300		0	811.67	488.33	inf
2	Ingredient 3 availability	2000		0	1269.42	730.58	inf
3	Ingredient 3 >=25% in product_A	0		0	-50.85	-inf	50.85
4	Ingredient 3 >=25% in product_B	0		-1.04	0	-50.07	42.04
5	Product B >=10% of ingredient 2 than product A	0		-0.69	0	-330	42.04
6	Quantity of ingredient 1 in product A must be at least 10%	0		0	-1228.11	-inf	1228.11
7	Quantity of ingredient 1 in product B must be at least 10%	0		-1.04	0	-35.16	242.68
8	Quantity of ingredient 2 in product A must be at least 10%	0		-0.41	0	-300	21.82
9	Quantity of ingredient 2 in product B must be at least 10%	0		0	-216.44	-inf	216.44
10	Quantity of ingredient 3 in product A must be at least 10%	0		0	-399.66	-inf	399.66
11	Quantity of ingredient 3 in product B must be at least 10%	0		0	-59.03	-inf	59.03
12	Production capacity	10		512.58	0	9.67	15.69

**a** Write the expressions of the missing constraints. You may use indices to write your expressions in a compact form

The minimum quantity of any ingredient in any product is 10%, this means that the quantity of ingredient  $i$  in product  $j$  must be at least 10% of the total amount of ingredients used for product  $j$ , which is the sum of the three ingredients:

$$x_{ij} \geq 0.1 * (x_{1j} + x_{2j} + x_{3j}) \quad \forall i, j$$

As for the production capacity constraint, let us express the production speed for product  $j$   $v_j$  as the ratio between the quantity of product  $j$  produced and the production time of product  $j$ ,  $t_j$  in hours:

$$v_j = \frac{\text{grams of product } j}{\text{production hours } j} = \frac{x_{1j} + x_{2j} + x_{3j}}{t_j}$$

Now, the problem states that the production speeds are  $v_A = 300$  grams/hour and  $v_B = 175$  grams/hour. The problem also states that the production time available is 10 hours, that is:

$$t_A + t_B \leq 10$$

Plugging in the expressions for the speeds and the values given yields:

$$\frac{x_{1A} + x_{2A} + x_{3A}}{300} + \frac{x_{1B} + x_{2B} + x_{3B}}{175} \leq 10$$

**b** Write down the objective function of the dual problem and provide an economic interpretation of the dual variables

The objective coefficients of the dual are the same as the RHS of the primal and are given in the RHS column given by the solver:

$$\min z^* = 1500 * u_1 + 1300 * u_2 + 2000 * u_3 + 10 * u_{12}$$

- $u_1$ : Shadow price of the availability of constraint 1 (non-negative). It indicates the increase in the objective function per every additional gram of ingredient 1.
- $u_2$ : Shadow price of the availability of constraint 1 (non-negative). It indicates the increase in the objective function per every additional gram of ingredient 2.
- $u_3$ : Shadow price of the availability of constraint 1 (non-negative). It indicates the increase in the objective function per every additional gram of ingredient 3.
- $u_4$ : Shadow price of the quality requirement that the quantity of ingredient 3 must be higher than 25% in product A (non-positive). It indicates the decrease in the objective function if the required amount of ingredient 3 increases
- $u_5$ : Shadow price of the quality requirement that the quantity of ingredient 3 must be higher than 25% in product B (non-positive). It indicates the decrease in the objective function if the required amount of ingredient 3 increases
- $u_6 - u_{11}$ : Shadow price of the quality requirement that the quantity of any ingredient in any product must be higher than 10% (non-positive). It indicates the decrease in the objective function if the minimum amount required increases
- $u_{12}$ : Shadow price of the production line capacity constraint (non-negative) It indicates the increase in the objective function for any additional hour of production capacity in the line

**c** What is the maximum price for ingredient 3 that the company can pay without changes in the optimal solution? Motivate your response

The coefficient of  $x_{3A}$  can decrease at most 14€ cents (from 1.75 to 1.61 euros) without changing the optimal solution. Since this coefficient is the price of product A minus the cost of ingredient 3, this is the maximum increase in the price of ingredient 3 without changes in the optimal solution. Thus, the maximum price is 139€cents.

**d** The company can improve the daily capacity of the line in 1 hour for 50000€. How would this affect the profits and how long will it take for the company to make this investment profitable, considering only the increase in the daily profits

The daily profits would increase in 512.58 euros. The company would need:

$$\frac{50000}{512.8} = 97.5$$

operation days to return this investment with the additional profits obtained

**e** Is the availability of ingredient 1 a binding constraint? How would you optimise the procurement of ingredients (daily amounts that are available) according to the provided solution? (2 points)

The constraint is binding because the slack is zero. The company can reduce the availability of ingredient 2 in 811.67 grams and the availability of ingredient 3 in 1269.42 grams

## 2.3.17 Manufacturing Solar Panels Revisited

### Problem Definition

Consider the following solution to a daily production mix optimization problem where:

- The decision variables are the number of units to produce of three different types of products
- The objective function coefficients (Objective coefficient in table) are the unitary profits (difference between production costs and sell price per unit) in euros
- The three constraint represent the total availability of three sections in minutes

### Decision variables

Decision variable	Solution	Reduced cost	Objective Coefficient	Lower bound	Upper bound
product 1 units	300	0	30	24.44	inf
product 2 units	33.33	0	20	-0	90
product 3 units	0	-8.33	40	-inf	48.33

### Constraints

Constraint	Right Hand Side	Shadow Price	Slack	Min RHS	Max RHS
Section 1 availability	400	6.67	0	300	525
Section 2 availability	600	11.67	0	0	800
Section 3 availability	600	0	166.67	433.33	inf

Answer the following questions. Motivate your response based on the provided results:

- Is it profitable to produce units of the three types of products? If any of the products is not profitable, what changes are needed to make it profitable?
- In the actual economic context, the production costs are rising. Discuss how the rise of the production costs can affect your profit and identify the maximum unitary production cost increase that your daily production mix can support without changes in the base solution
- In order to increase your production capacity, you need to evaluate 2 different improvement proposals from an external consultant. The first one considers a 12% increase of the availability of section 1, and the second a 7.5% increase of the availability of section 2. Which one is more profitable for the company?

### Solution

- Is it profitable to produce units of the three types of products? If any of the products is not profitable, what changes are needed to make it profitable?

Since the reduced cost of product 3 is negative, it is not profitable to produce product 3. To make it profitable, the objective coefficient of product 3 should be increased to at least 48.33.

- In the actual economic context, the production costs are rising. Discuss how the rise of the production costs can affect your profit and identify the maximum unitary production cost increase that your daily production mix can support without changes in the base solution

If the production costs increase, we can increase the selling costs ensuring that the resulting unitary profits is within the lower and upper bounds of the objective function coefficients. Note that in this case, product 2 provides the smaller margin to modify the selling price, since the difference between the upper and lower bounds is 90.

We might also consider keeping the selling price constant, and analyze the difference between the objective coefficient and its lower bound to see how much the production costs can increase. In this case, product 1 provides the smaller margin to increase the production costs, since the difference between the objective coefficient and its lower bound is  $30 - 24.44 = 5.56$ .

- In order to increase your production capacity, you need to evaluate 2 different improvement proposals from an external consultant. The first one considers a 12% increase of the availability of section 1, and the second a 7.5% increase of the availability of section 2. Which one is more profitable for the company? (1 point)

Let us first consider the first proposal. The new availability of section 1 is  $400 * 1.12 = 448$ . The new capacity is within the Minimum and Maximum values for the RHS, so this change is not going to change the base solution. The objective function is going to increase proportionally to the shadow price of section 1, which is 6.67. Therefore, the profit is going to increase by  $6.67 * 400 * 0.12 = 6.67 * 48 = 320.16€$ .

In the second proposal, on the other hand, the new availability of section 2 is  $600 * 1.075 = 645$ . The new capacity is within the Minimum and Maximum values for the RHS, so this change is not going to change the base solution either. The objective function is going to increase proportionally to the shadow price of section 2, which is 11.67. Therefore, the profit is going to increase by  $11.67 * 45 = 525,15€$ .

Therefore, the second proposal is more profitable for the company.

You are an investment analyst at March & Sanchis Investments Ltd. and you need are analysing the portfolio of a customer, based on the investment case described and solved here

A solver provides this solution:

**Decision Variables:**

	Vari-ables	Solution (GRB)	Reduced cost (GRB)	Objective Coeffi- cient (GRB)	Objective bound (GRB)	Lower	Objective bound (GRB)	Upper
1	x_1	116129.03	0.00	0.10	0.08		0.12	
2	x_2	0.00	-0.02	0.06	-Inf		0.08	
3	x_3	483870.97	0.00	0.08	0.05		0.10	
4	x_4	400000.00	0.00	0.15	0.12		inf	

**Constraints:**

	Constraint	Right Side	Hand	Slack	Shadow Price	Min RHS	Max RHS
0	Maximum_budget	1000000.00		0.00	0.10	861538.46	2053846.15
1	Risk	0.00		0.00	6.45	-360.00	880.00
2	Minimum_investment_in_real_es- tate	200000.00		-283870.97	0.00	-Inf	483870.97
3	Maximum_invest- ment_in_crypto_assets	400000.00		0.00	0.03	48000.00	464285.71

Answer the following questions and motivate all of your responses based on the data provided in the solution

- How sensitive is the solution to changes in the expected return of the different types of assets? Describe how changes in the expected return of each asset could affect the solution.
- What is the minimum expected return of bonds that would render this asset profitable for your customer?
- Based on the solution provided, would you recommend your customer to consider increasing or decreasing the exposure to crypto assets?

### 2.3.18 Solution

For the sake of clarity, and as a reminder, the decision variables are: -  $x_1$ : amount to invest in stocks -  $x_2$ : amount to invest in bonds -  $x_3$ : amount to invest in real estate -  $x_4$ : amount to invest in crypto assets

- How sensitive is the solution to changes in the expected return of the different types of assets? Describe how changes in the expected return of each asset could affect the solution.

Changes in the market can cause the expected return of the different types of assets to change. The lower and upper bounds of the Objective coefficient provide the range of values of the expected return of each asset that would not change the basic solution (meaning that the resulting profit would change, but not the optimal amount to invest in each asset). For example, if we pay attention to the specific values, the expected return of stocks can increase by 0.02 (from 0.10 to 0.12) or decrease by 0.02 (from 0.10 to 0.08) without changes in the optimal solution. This represents a change of +/- 20% in the expected return of bonds, which is a significant change.

Overall, we get the following ranges: - Stocks: -20% decrease, and +20% increase - Bonds: unlimited decrease, and +33.33% increase - Real estate: -37.5% decrease, and +25% increase - Crypto assets: -20% decrease, and unlimited increase

The optimal solution is more sensitive to changes in the expected return of stocks and real state, and less sensitive to changes in the expected return of bonds and crypto assets. In general, the ranges obtained in percentage are wide, and we can conclude that the solution is not particularly sensitive to changes in the expected return of the different types of assets.

- What is the minimum expected return of bonds that would render this asset profitable for your customer?

Currently, we can note that it is not profitable to invest in bonds, as the reduced cost is negative (-0.02), and the solution is zero. We should increase the expected return of bonds until the reduced cost is positive, which would mean that the solution is no longer zero. The minimum expected return of bonds that would render this asset profitable for the customer is therefore 0.08.

- Based on the solution provided, would you recommend your customer to consider increasing or decreasing the exposure to crypto assets?

The shadow price of crypto assets is positive, meaning that it would be profitable for our customer to increase the exposure to crypto assets. The shadow price is 0.03, which means that for every additional euro invested in crypto assets, the profit would increase by 0.03 euros. We could increase the exposure up to the maximum right hand side of the constraint, which is 464285.71 euros without changes in the base solution.

### 2.3.19 Production Mix

#### Problem definition

Chappie Ltd. has 3 production lines (A, B, and C) to make 4 different types of metal sheets of width 0.4, 0.5, 0.6, and 0.7 mm. The company needs to meet the monthly demand for each kind of product (in Tons) expressed in the next table:

Sheet 0.4 mm	Sheet 0.5 mm	Sheet 0.6 mm	Sheet 0.7 mm
500	1200	1500	300

The production lines do not have the same overall efficiency for all types of sheets. Furthermore, due to design constraints, not all the lines are capable of manufacturing all types of sheets. The table below indicates the manufacturing costs in € of the different products in the different lines: a dash “-” means that it is not possible to manufacture a product in a line:

Line	Sheet 0.4 mm	Sheet 0.5 mm	Sheet 0.6 mm	Sheet 0.7 mm
A	60	50	50	45
B	80	70	75	70
C	•	60	60	•

The different lines have the following capacities (in hours) in the planning period:

A	B	C
500	480	370

And the following table represents the manufacturing time (hours) per ton of product required for each type of metal sheet in the different lines:

Line	Sheet 0.4 mm	Sheet 0.5 mm	Sheet 0.6 mm	Sheet 0.7 mm
A	0.4	0.3	0.3	0.25
B	0.8	0.6	0.7	0.6
C	•	0.5	0.4	•

**Formulate a linear programming problem to obtain the monthly production plan (quantity of metal sheet in tons of each type of metal sheet to be manufactured in each production line)**

**indexes**

i: Manufacturing lines (A,B,C)

j: Product types Sheet (0.4, 0.5, 0.6, 0.7) mm

**Decision Variables**

$x_{ij}$  = Quantity of product  $j$  to produce in line  $i$

\*\*Objective function Minimize cost

$$\min z = \sum_i \sum_j c_{ij} * x_{ij}$$

where  $c_{ij}$  is the cost of manufacturing product  $j$  in line  $i$  as expressed in the second table.

**Constraints Capacity**

$$\sum_j a_{ij} \cdot x_{ij} \leq b_i, \forall i$$

Where  $a_{ij}$  represent the manufacturing time required to manufacture product  $j$  in line  $i$  as expressed in the last table and  $b_{\{i\}}$  represents the capacity of line  $i$  for the planning period

**Demand**

$$\sum_i x_{ij} \geq d_j, \forall j$$

## 2.3.20 Sabrina Revisited

### Analysis of the problem model

Spellman's Ltd is a company that produces two types of chilling soft drinks: A, and B. Both beverages use a semi-elaborate C, another expensive ingredient D and other ingredients that are not relevant for production planning.

The selling price of drink A is 3€/liter and the selling price of drink B is 2€/liter.

1 liter of drink A uses 3 grams of ingredient D. A liter of drink B uses 1 gram of ingredient D. There are only 3 grams of ingredient D available per day.

The factory only has one mixer to elaborate both drink types and the semi-elaborate. It takes 1 hour to process a liter of drink A, 1 hour to process 1 liter of drink B, and 1 hour to process 1cl of semi-elaborate C. The mixer is available 6 hours per day.

Drink A uses 2cl of semi-elaborate C and drink B uses 1cl of semi-elaborate C. The company has 3cl of semi-elaborate C plus the amount they decide to produce every day.

The following problem model provides the optimal daily production:

**Decision variables:** -  $x_A$ : Production of drink A in liters -  $x_B$ : Production of drink B in liters -  $x_C$ : Production of semi-elaborate C in centiliters

$$x_A, x_C, x_B \in \mathbb{R}$$

### Objective function

$$\max z = 3 * x_A + 2 * x_B$$

z is the profit in euros.

### Constraints

- Availability of ingredient D in grams:

$$3 * x_A + x_B \leq 3$$

- Availability of mixer in hours:

$$x_A + x_B + x_C \leq 6$$

- Availability of semi-elaborate C in centiliters:

$$2 * x_A + x_B - x_C \leq 3$$

- Logical constraint

$$x_A, x_C, x_B \geq 0$$

A solver provides the following solution:

Optimal

Total profit is 6.00 €

The following table shows the decision variables:

j	Variables		Solu- tion	Reduced cost	Objective Coeffi- cient	Objective bound	Lower	Objective bound	Upper
A	Production (Liters)	A	0	-3	3	-inf		6	
B	Production (Liters)	B	3	0	2	1		inf	
C	Production (Liters)	C	3	0	0	-0		1.5	

The following table shows the constraints:

j	Constraint	Slack	Shadow Price	Right Hand Side	Min RHS	Max RHS
0	Availability of ingredient D (Liters)	0	2	3	0	4.5
1	Availability of mixer (hours)	0	0	6	3	inf
2	Availability of semi-elaborate (Liters)	3	0	3	0	inf

Answer the following analysis questions:

- Operational problems in the mixer, like mechanical problems, could reduce the available mixer. Based on the provided solution, how is the company able to deal with this type of problem? Motivate your response

The mixer is fully used (slack of 0 in the mixer constraint). A reduction of the availability of the mixer would lead to a reduction in the production of the drinks. The minimum Right-Hand-Side (RHS) indicates that the current optimal solution will remain in the same basic solution as long as the availability is greater than 3 hours. This means that the production of both products A and B will remain as long as the mixer is available for at least 3 hours. If the mixer is available for less than 3 hours, the optimal solution will no longer remain in the same basic solution. Therefore, the company can deal with this type of problem by ensuring that the mixer is available for at least 3 hours per day.

- The managers of the company want you to negotiate with a new supplier that could increase your daily supply of ingredient D. What would be your terms for negotiating with the new supplier, ensuring that the basic variables remain the same?

The shadow price of the ingredient D constraint is 2. This means that the company is willing to pay up to 2€ for an additional gram of ingredient D. The maximum RHS of the ingredient D is 4.5 grams. Therefore, the company should negotiate with the new supplier to provide the additional ingredient D at a price lower than 2€ per gram, increasing the daily supply from 3 grams to 4.5 grams. This will ensure that the basic variables remain the same, and the company can increase its daily supply of ingredient D without affecting the optimal solution.

- The managers of the company would like to review the prices of both products. Analyse how changes in the prices of both products could affect the production plan and provide useful feedback to guide the revision of the prices

First, with the current prices, the Production of product A is not profitable for the company (0 Liters). Since the reduced cost is -3 euros, the company should increase the price of product A to at least 6 euros per liter to make it profitable.

Taking a look at product B, the production of product B is profitable for the company (3 Liters). The price could be increased without affecting the production plan. However, the company should be careful decreasing the price of product B, as the lower bound for the price per liter (objective function coefficient) is 1 euro.



## Duality

1. Write down the dual problem.

The dual problem is the following:

**Decision variables:** -  $u_1$ : Shadow price of the ingredient D constraint -  $u_2$ : Shadow price of the mixer constraint -  $u_3$ : Shadow price of the semi-elaborate C constraint

**Objective function**  $\min z = 3 * u_1 + 6 * u_2 + 3 * u_3$

**Constraints**  $3 * u_1 + u_2 + 2 * u_3 \geq 3$

$$u_1 + u_2 + u_3 \geq 2$$

$$u_2 - u_3 \geq 0$$

$$u_1, u_2, u_3 \geq 0$$

## Problem Extension

1. Spellman's Ltd wants to evaluate the possibility of subcontracting part of the production of semi-elaborate C to an external provider, at a cost of 0.35€ per cl. With the new set-up, the daily availability of semi-elaborate C will be the sum of the semi-elaborate C produced at Spellman's plus the amount purchased from the new provider. What changes are needed in the problem to evaluate the subcontracting of semi-elaborate C? (1.5 points)

**Decision variables:** We need to introduce a new decision variable:

- $x_{C_{sub}}$ : Amount of semi-elaborate C purchased from the new provider

**Objective function** The objective function will also change to include the cost of purchasing the semi-elaborate C:

$$\max z = 3 * x_A + 2 * x_B - 0.35 * x_{C_{sub}}$$

**Constraints** The new constraint to evaluate the availability of semi-elaborate C will be:

$$2 * x_A + x_B - x_C - x_{C_{sub}} \leq 3$$

The availability of the mixer constraint will also change to include the production of the semi-elaborate C purchased from the new provider:

$$x_A + x_B + x_C + x_{C_{sub}} \leq 6$$

And obviously, the logical constraint will also change to include the new decision variable:

$$x_A, x_C, x_B, x_{C_{sub}} \geq 0$$

## 2.3.21 Lost in Space

### Problem Description

Will Robinson and his crew are lost in space and have crashed their spaceship in an alien planet. They need to collect metals to repair the engine of their spaceship. More specifically, the piece to repair is made of a metal alloy of chrome, aluminium, and titanium. To build the new piece, Will needs to collect 1 kilogram of chrome, 800 grams of aluminium and 1.5 kilograms of titanium. Will and his crew can gather these metals from 3 different mining locations they have identified around the camp. Will has estimated the average kg/hour that can be extracted in each location:

Mining Location	Chrome (kg/hour)	Aluminium (kg/hour)	Titanium (kg/hour)
Location 1	0.1	0.3	0.05
Location 2	0.05	0.2	0.1
Location 3	0.02	0.15	0.2

Write down a Continuous Linear Programming model to minimize the total time used to collect the materials needed to build the part needed to fix the engine.

**Decision Variables:**

Let  $x_1$ ,  $x_2$ , and  $x_3$  be the number of hours spent in each location (1, 2, and 3) respectively.  $x_1$ ,  $x_2$ , and  $x_3$  are continuous variables (non-negative real numbers).

**Objective Function:**

Minimize the total time spent collecting the materials:

$$\min z = x_1 + x_2 + x_3$$

**Constraints:** Ensure that the amount of each metal collected is enough to build the part needed to fix the engine:

- Chrome (kg):

$$0.1x_1 + 0.05x_2 + 0.02x_3 \geq 1$$

- Aluminium (Kg):

$$0.3x_1 + 0.2x_2 + 0.15x_3 \geq 0.8$$

- Titanium (Kg):

$$0.05x_1 + 0.1x_2 + 0.2x_3 \geq 1.5$$

Likewise, we could define the matrix  $A$  containing the coefficients of the constraints and the vector  $b$  containing the right-hand side of the constraints, and the vector  $c$  containing the coefficients of the objective function as follows:

$$A = \begin{bmatrix} 0.1 & 0.05 & 0.02 \\ 0.3 & 0.2 & 0.15 \\ 0.05 & 0.1 & 0.2 \end{bmatrix}$$

$$b = \begin{bmatrix} 1 \\ 0.8 \\ 1.5 \end{bmatrix}$$

$$c = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

The problem model can be written as:

$$\min z = c^T x$$

subject to:

$$Ax \geq b$$

$$\text{Where } x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

## 2.4 Python libraries

This section contains Python library tutorials:

### 2.4.1 Introduction to Linear Programming with PuLP

Try me



#### Requirements

##### Install in your environment

###### Pip Installation

The simplest way to install PuLP in your environment is using `pip`. If you have installed Python and `pip` in your environment, just open a terminal and try:

```
pip install pulp
```

###### Conda Installation

If you use Conda, open a Conda Terminal and try:

```
conda install -c conda-forge pulp
```

###### Google Colabs installation

Run the following code cell to try this notebook in Google Colabs:

```
[ ]: !pip install pulp
```

###### Binder installation

Run the following code cell to try this notebook in Binder:

```
[ ]: !pip install pulp
!pip install pandas
!pip install numpy
```

### The PuLP library

In this tutorial, we will learn to model and solve Linear Programming Problems using the Python open source Linear Programming library **PuLP**.

To guide this example, we will use a simple LPP formulated in class:

$$\text{maximise } z = 300x + 250y$$

Subject to:

$$2x + y \leq 40$$

$$x + 3y \leq 45$$

$$x \leq 12$$

In Pulp this problem can be solved with the following code cell:

```
[3]: # Let's start importing the library PuLP to solve linear programs
import pulp

# Create an instance of the problem class using LpProblem
model = pulp.LpProblem("Production_Mix_example", pulp.LpMaximize) #this will create_
↳an instance of an LP Maximise problem

# Create the variables of the problem
x = pulp.LpVariable('x', lowBound=0, cat='Continuous')
y = pulp.LpVariable('y', lowBound=0, cat='Continuous')

# Add the objective function to the model
model += 300 * x + 250 * y, "Profit"

# And the constraints
model += 2 * x + y <= 40, "Man Power"
model += x + 3 * y <= 45, "Machine Operating Time"
model += x <=12, "Marketing"

# solve the problem
model.solve()
pulp.LpStatus[model.status]

# Print our decision variable values
print("Production of Product A = {}".format(x.varValue))
print("Production of Product B = {}".format(y.varValue))
```

## Code Explanation

The next sections explain step by step the main functions involved in the code above and how they work together to solve a CLP problem.

### Problem Class LpProblem

PuLP uses *classes* providing different methods to model and solve LPPs. The class that will contain our model is the **LpProblem** class. To create a new LpProblem we use the pulp LpProblem function:

- **LpProblem(name='None', sense=1)**: Creates a new Linear Programming Problem. The parameter name (default 'None') assigns a name to the problem. The parameter sense (either pulp.LpMinimize or pulp.LpMaximize) sets the type of objective function. The default is minimize.

So, the line code:

```
model = pulp.LpProblem("Production_Mix_example", pulp.LpMaximize)
```

Created a problem called "Production\_Mix\_example" and set the objective function to maximise.

### Variable class LpVariable

The definition of a LPP program with PuLP is very similar to the standard procedure used to model a problem. First, we need to define the unknown variables in our problem. For this purpose we use the class **LpVariable**. The function LpVariable allows us to create a variable:

- **LpVariable(name, lowBound=None, upBound=None, cat='Continuous', e=None)**: Creates an instance of variable with the following properties:
  - **Name**: The name of the variable to be used in the solution.
  - **lowBound**: The lower bound of the variable, the default is unrestricted (-Inf).
  - **upBound**: The upper bound of the variable. The default is unrestricted (Inf).
  - **cat**: Either 'Continuous' for continuous variables, 'Binary' for binary variables or 'Integer' for Integer variables. We will see in detail binary and integer variables in the course unit for Mixed Integer Programming, but now you know that you will be able to model and solve this type of problems with PuLP. The default is 'Continuous'.
  - **e**: This parameter is outside the scope of this course and can be neglected for now.

We can define the variables of our problem using the LpVariable function:

```
# Create the variables of the problem
x = pulp.LpVariable('x', lowBound=0, cat='Continuous')
y = pulp.LpVariable('y', lowBound=0, cat='Continuous')
```

Note that we have created two variables, x and y, with lower bound 0 and type continuous. We could have also use upBound to set the upper bound of the variables. For instance, if we wanted to set the upper bound of x to 12, we could have used this parameter.

### Adding expressions

In PuLP, both objective function and constraints are *expressions* (algebraic expressions containing variables) that have to be added to the instance problem using the standard operand '+='. For instance, to add the objective function in this example, we could write:

```
model += 300 * x + 250 * y, "Profit"
```

With this line of code, we have added a new expression with name "Profit" that multiplies the technological coefficients to the variables  $x$  and  $y$  (as defined in the code snippet in the previous section).

Note that we have used the operand '+=' to add the expression to the `model` variable. We can use the same procedure to add the constraints:

```
# And the constraints
model += 2 * x + y <= 40, "Man Power"
model += x + 3 * y <= 45, "Machine Operating Time"
model += x <= 12, "Marketing"
```

Note that we added names to the constraints. This is not mandatory, but it is a good practice to do so. Note also that we needed one line of code per constraint. This is not a problem for this simple example, but it can be a problem for more complex problems. We will see how to solve this issue in the next section.

### Solving the problem

Once we have defined the problem, we can solve it using the **solve** method of the `LpProblem` class:

```
# solve the problem
model.solve()
```

Note that we have not specified any solver. PuLP will use the default solver (CBC) to solve the problem. We will see in next chapters how to use other solvers.

### Checking the status of the solution

Once the problem has been solved, we can check the status of the solution using the **LpStatus** method of the `LpProblem` class:

```
pulp.LpStatus[model.status]
```

The status of the solution can be:

- **Optimal:** An optimal solution has been found.
- **Not Solved:** The problem has not been solved yet.
- **Infeasible:** There are no feasible solutions (e.g. if you set the constraints  $x \leq 1$  and  $x \geq 2$ ).
- **Unbounded:** The constraints are not bounded, maximising the solution will tend towards infinity (e.g. if the only constraint was  $x \geq 3$ ).
- **Undefined:** The optimal solution may exist but may not have been found.
- **Error:** An error occurred (e.g. if you try to retrieve the infeasible or unbounded status from a model which is not infeasible or unbounded).
- **None:** No solution status exists (e.g. if you create a model but do not solve it).

## Retrieving the solution

Once the problem has been solved, we can retrieve the solution using the `varValue` method of the `LpVariable` class:

```
print(x.varValue)
```

## Making PuLP programs more scalable

In this section, we will see how to make PuLP programs more scalable. We will use some Python basic concepts related to iterables. You can check these tutorials if you need a refresher:

- [Basic Python Iterators](#)
- [Intermediate Python Iterators](#)

### Using dictionary variables

Note however that if we use the script above, the problem just does not scale up because **we need a line of code for every unknown**. What if we have hundreds of unknowns? Luckily for us, PuLP provides a convenient method to write more efficient codes for our program, the `LpVariable.dicts` method, which basically allows us to create a set of variables with the same category, upper bounds and lower bounds at once:

- `LpVariable.dicts(name, index, lowBound=None, upBound=None, cat='Continuous')`: Creates a dictionary containing variables of type `cat` (default 'Continuous'), indexed with the keys contained in the *iterable* `index` and bounded by `lowBound` (default -Inf) and `upBound` (default Inf).

For instance, we can define our variables using the `LpVariable.dicts` method like this:

```
# First we define a tuple with the variable names x and y
variable_names = (1,2)
# Then we create a variable from a dictionary, using the variable names as keys
X = pulp.LpVariable.dicts("vars", variable_names, lowBound=0, cat='Continuous')
```

Note that we used a **tuple** with the variable names to create both variables in a single line of code. If we had 20 variables, we could have used a tuple with the 20 variable names and created all the variables in a single line of code. This is a much more scalable way to create variables in PuLP!

### Using lpSum

We can also use the `lpSum` function to create linear expressions from a vector of variables, just as we can use summations to express the objective function and constraints in a more compact mathematical form. For instance, we can write the objective function like this:

$$C = [c_1, c_2] = [300, 250]$$

$$X = [x_1, x_2]$$

$$\max z = 300x_1 + 250x_2 = \sum_{j=1}^2 c_j * x_j$$

PuLP provides a convenience function named `lpSum` to achieve the same result efficiently. `lpSum` takes an array of expressions and returns the summation of the elements in the array. Let us see its action:

```
# We define the objective function coefficients
C = [300, 250]
# Then we add the objective function to the model like
model += (
    pulp.lpSum([
        C[i] * X[variable_names[i]]
```

(continues on next page)

(continued from previous page)

```

        for i in range(len(X))]
    ), "Profit"

```

Notice that we have used **list comprehension** to create the array where every element is the product of a coefficient times the corresponding decision variable, and passed this array to the `lpSum` function using an index array. We have used **range** and **length** to go through all decision variables.

### Using matrices and arrays to define problem constraints

We can also use matrices and arrays to define the constraints of the problem. For instance, we can write the constraints of the problem like this:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 1 & 3 \\ 1 & 0 \end{bmatrix}$$

$$b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 40 \\ 45 \\ 12 \end{bmatrix}$$

$$\sum_{j=1}^2 a_{ij} * x_j \leq b_i \forall i$$

In Python, we can achieve the same *compactness* defining the coefficients of the constraints in of a matrix, the LHS coefficients in a vector, and use a for loop to add the constraints to the model:

```

# need We also define the name for the constraints
constraint_names = ['Man Power', 'Machine Operating Time', 'Marketing']

# Define the RHS constraint coefficients in a matrix
A=[[2, 1], #Coefficients of the first constraint
   [1, 3], #Coefficients of the second constraint
   [1, 0]] #Coefficients of the third constraint

# And vector b
b = [40, 45, 12] #limits of the three constraints

# need We also define the name for the constraints
constraint_names = ['Man Power', 'Machine Operating Time', 'Marketing']
# Now we add the constraints using
# model += expression, name
# eg model += 2*X + y <= 40
# We add all constraints in a loop, using a vector and the function lpSum to generate
↳the linear expression:
for i in range(len(A)):
    model += pulp.lpSum([
        A[i][j] * X[variable_names[j]]
        for j in range(len(variable_names))]) <= b[i] , constraint_names[i]

```

Here we just put all data in iterables, and used a for loop to add the constraints to the model. This is a much more scalable way to add constraints to a model!

Let's put everything together and solve the problem:

```

[4]: import pulp
# First we define a tuple with the variable names x and y
variable_names = (1,2)

```

(continues on next page)



(continued from previous page)

```

# Then we create a variable from a dictionary, using the variable names as keys
X = pulp.LpVariable.dicts("x",
                          (i for i in variable_names),
                          lowBound=0,
                          cat='Continuous')

# We define the objective function coefficients
coefficients = [300, 250]

# We create the model
model = pulp.LpProblem("Profit maximising problem", pulp.LpMaximize)

# We use the function lpSum to generate the linear expression from the coefficients_
↳and variables
model += (
    pulp.lpSum([
        coefficients[i] * X[variable_names[i]]
        for i in range(len(X))]
    ), "Profit"

# Define the RHS constraint coefficients in a matrix
A=[[2, 1], #Coefficients of the first constraint
   [1, 3], #Coefficients of the second constraint
   [1, 0]] #Coefficients of the third constraint

# And vector b
b = [40, 45, 12] #limits of the three constraints

# need We also define the name for the constraints
constraint_names = ['Man Power', 'Machine Operating Time', 'Marketing']
# Now we add the constraints using
# model += expression, name
# eg model += 2*X + y <= 40
# We add all constraints in a loop, using a vector and the function lpSum to generate_
↳the linear expression:
for i in range(len(A)):
    model += pulp.lpSum([
        A[i][j] * X[variable_names[j]]
        for j in range(len(variable_names))] <= b[i] , constraint_names[i]

# Solve our problem
model.solve()
pulp.LpStatus[model.status]

# Print our decision variable values
for variable in model.variables():
    print ("{} = {}".format(variable.name, variable.varValue))

```

```

x_1 = 12.0
x_2 = 11.0

```

```

C:\Users\ffraile\PycharmProjects\operations-research-notebooks\venv\lib\site-packages\
↳pulp\pulp.py:1352: UserWarning: Spaces are not permitted in the name. Converted to
↳ '_'
warnings.warn("Spaces are not permitted in the name. Converted to '_'")

```

### Displaying the solution using Pandas

Pandas is a great library to manipulate tables. You can find a basic tutorial [here](#).

We can use Pandas to display the solution in a nice table. First we need to import Pandas:

```
import pandas as pd
```

We are also going to use the library `display` to show styled output using Markdown notation.

Now, let us display the solution in a nice table using Pandas. We are going to first display the solution value using markdown and then we will use Pandas to create a table with the results.

```
[12]: # We are going to use panda to display the results as tables using Panda
import pandas as pd
#And we will use numpy to perform array operations
import numpy as np
#We will use display and Markdown to format the output of code cells as Markdown
from IPython.display import display, Markdown

# Solution
max_z = pulp.value(model.objective)

#We use display and Markdown to show the value using markdown
display(Markdown("The value of the objective function is **%.2f**"%max_z))

# Print our decision variable values
display(Markdown("The following tables show the values obtained: "))
# First we create a dataframe from the dictionary of the solution. We want to use the_
↪variable indexes to present the results and
# place the different values provided by the solver in the data frame.
var_df = pd.DataFrame.from_dict(X, orient="index",
                               columns = ["Variables"])
# First we add the solution. We apply a lambda function to get only two decimals:
var_df["Solution"] = var_df["Variables"].apply(lambda item: "{:.2f}".
↪format(float(item.varValue)))
# We do the same for the reduced cost:
var_df["Reduced cost"] = var_df["Variables"].apply(lambda item: "{:.2f}".
↪format(float(item.dj)))

# We use the display function to represent the results:
display(var_df)

# we define a dictionary with the constraints:
const_dict = dict(model.constraints)
#We create a list of records from the dictionary and exclude the Expression to have a_
↪more compact solution.
con_df = pd.DataFrame.from_records(list(const_dict.items()), exclude=["Expression"],_
↪columns=["Constraint", "Expression"])

#Now we add columns for the solution, the slack and shadow price

con_df["Right Hand Side"] = con_df["Constraint"].apply(lambda item: "{:.2f}".format(-
↪const_dict[item].constant))
con_df["Slack"] = con_df["Constraint"].apply(lambda item: "{:.2f}".format(const_
```

(continues on next page)

(continued from previous page)

```

↪dict[item].slack))
con_df["Shadow Price"] = con_df["Constraint"].apply(lambda item: "{:.2f}".
↪format(const_dict[item].pi))

# And we display the results
display(con_df)

```

The value of the objective function is **6350.00**

The following tables show the values obtained:

	Variables	Solution	Reduced cost
x	vars_x	12.00	0.00
y	vars_y	11.00	0.00

	Constraint	Right Hand Side	Slack	Shadow Price
0	Man_Power	40.00	5.00	-0.00
1	Machine_Operating_Time	45.00	-0.00	83.33
2	Marketing	12.00	-0.00	216.67

### Analysis questions

1. Search for the exercise “The Good Carpenter” and try to model it.
2. Ask an AI assistant to model the problem and to provide a Python script to solve it. You can use this sample prompt:
 

Provide the problem model and a Python script using PuLP to solve the problem definition below assuming that decision variables are continuous: [Copy the problem definition from the exercise “The Good Carpenter” here]
3. Try the solution provided by the AI assistant. Does it work?
4. Is the solution provided by the AI assistant scalable? If the answer is not, ask it to provide a solution that is more scalable and explain its solution, using the following prompt template:
 

The Python script provided does not scale up well if the number of decision variables or the number of constraints increase. Can you provide a more scalable solution using PuLP? Please describe the changes you made and motivate why

Did the AI assistant use the same methods explained in this tutorial?

5. Ask the assistant to use the problem you provided as an example to demonstrate how indices, summations, and “for all” expressions can make mathematical models more compact and scalable. Also, ask the assistant if these mathematical expressions are conceptually similar to the PuLP functions used in the scalable solution.

### Solved exercises

The following notebooks include exercises solved with PuLP, using different solvers, do not forget to check them out!

- Making Chappie solved with CBC
- Blending problem solved with CBC
- Blending Craft Beer solved with CBC
- Blending Craft Beer solved with GRB
- Chappie II solved with CBC
- Manufacturing solar cell panels solved with CBC
- Manufacturing solar cell panels solved with GRB

- Petroleum Blending Solved with CBC

### 2.4.2 Introduction to Linear Programming with Scipy

Try me



#### Requirements

##### Install in your environment

Google Colabs installation

 Scipy is already installed in Google Colabs, no installation required!

##### Scipy Installation

The simplest way to install SciPy in your environment is using `pip`. If you have installed Python and `pip` in your environment, just open a terminal and try:

```
pip install scipy
```

##### Conda Installation

If you use Conda, open a Conda Terminal and try:

```
conda install scipy
```

##### Binder installation

Run the following code cell to try this notebook in Binder:

```
[ ]: !pip install pandas
      !pip install numpy
      !pip install scipy
```

## Linear Optimisation with SciPy

In this tutorial, we will learn to model and solve Linear Programming Problems using the Python open source scientific library `SciPy`. `SciPy` is an awesome library extensively used for scientific and technical computing. It is built on top of `NumPy` and provides a wide range of functionality including optimization, signal processing, interpolation, and more. It also contains modules for linear algebra, optimization, and integration. `SciPy` is widely used in the scientific and engineering communities and is a powerful tool for data analysis and visualization.

In this tutorial, we will learn how to use `SciPy` to model and solve CLP problems. Just as with the previous tutorial, to guide this example, we will use a simple CLP formulated in class:

$$\text{maximise } z = 300x + 250y$$

Subject to:

$$2x + y \leq 40$$

$$x + 3y \leq 45$$

$$x \leq 12$$

```
[10]: # Let's start importing the linprog function of the optimize package of SciPy
from scipy.optimize import linprog
# We are going to use panda to display the results as tables using Panda
import pandas as pd
#And we will use numpy to perform array operations
import numpy as np
#We will use display and Markdown to format the output of code cells as Markdown
from IPython.display import display, Markdown
```

### Problem function `linprog`

Transforming the problem model

The function `linprog` of the `SciPy` package can be used to solve continuous linear programming problems expressed in the form:

$$\min z = c^T * x$$

s.t.

$$A_{ub} * x \leq b_{ub}$$

$$A_{uc} * x = b_{uc}$$

$$l \leq x \leq u$$

Where  $x$  is a (column) vector with the decision variables,  $c$  is a (column) vector with the objective function coefficients,  $z$  is the objective variable (scalar),  $A_{ub}$  is a matrix with the LHS coefficients of the constraints of type *less or equal*, and  $b_{ub}$  is a vector that contains the corresponding RHS coefficients of the same constraints, and finally,  $A_{uc}$  is a matrix with the LHS coefficients of type *equal*, and  $b_{uc}$  is a vector that contains the corresponding RHS coefficients.

This means that we need to convert our problem to comply with the format expected by `linprog`. In our example, we can convert the objective function as:

$$\min z^* = -z = -300 * x - 250y = [-300, -250]^T * [x, y]$$

That is, since our objective function is of type *maximize*, we use the equivalent minimization problem and the solution will be the negative of our original objective variable. As for the constraint, we need to express all the constraint (except the bounds) as of type *less\_or\_equal*. In our case, we do not need to apply any transformation, and our matrix  $A_{ub}$  becomes:

$$A_{ub} = \begin{bmatrix} 2 & 1 \\ 1 & 3 \\ 1 & 0 \end{bmatrix}$$

Note that the each rows correspond to a constraint, therefore, the vector  $b_{ub}$  is:

$$b_{ub} = [40, 45, 12]^T$$

Now,  $l$  is going to contain the minimum values of the decision variables, or **lower bound**. Since both variables need to be non-negative:

$$l = [0, 0]$$

and finally,  $u$  is going to contain the maximum values or **upper bound**, since  $x$  cannot be higher than 12, the upper bounds are:

$$u = [12, \infty]$$

Note that the upper bound for  $x$  is already expressed in the third constraint. We could remove this constraint and just use the upper bound, but this make it more difficult to obtain the shadow prices from the solution, so for now, we will use this formulation.

linprog function

The documentation of linprog can be found [here](#), but in short, it takes the following arguments:

**linprog** (**c**, **A\_ub=None**, **b\_ub=None**, **A\_eq=None**, **b\_eq=None**, **bounds=None**, **method='highs'**)

- **c**: 1-D array containing the coefficients of the linear objective function to be minimized.
- **A\_ub**: 2-D array (optional). The inequality constraint matrix. Each row of **A\_ub** specifies the coefficients of a linear inequality constraint on  $x$ .
- **b\_ub**: 1-D array (optional). The inequality constraint vector. Each element represents an upper bound on the corresponding value of **A\_ub** @  $x$ .
- **A\_eq**: 2-D array (optional). The equality constraint matrix. Each row of **A\_eq** specifies the coefficients of a linear equality constraint on  $x$ .
- **b\_eq**: 1-D array (optional). The equality constraint vector. Each element of **A\_eq** @  $x$  must equal the corresponding element of **b\_eq**.
- **bounds**: sequence: (optional). A sequence of (min, max) pairs for each element in  $x$ , defining the minimum and maximum values of that decision variable. Use **None** to indicate that there is no bound. By default, bounds are (0, **None**) (all decision variables are non-negative). If a single tuple (min, max) is provided, then min and max will serve as bounds for all decision variables.
- **method**: string (optional). This is the method-specific documentation for 'highs', which chooses automatically between 'highs-ds' and 'highs-ipm'. 'interior-point' (default), 'revised simplex', and 'simplex' (legacy) are also available.

```
[11]: # objective coefficient vector
c = np.array([-300, -250])
# Inequality constraints LHS matrix
A = np.array([[2, 1], [1, 3], [1, 0]])
# Inequality constraints RHS
b = np.array([40, 45, 12])
```

(continues on next page)

(continued from previous page)

```
#Bounds for x
x_bounds = (0, None) # Here we may as well use 12 as upper bound, but then, we need
↳to read the marginal of the upper bound to learn the shadow price!
# Bounds for y
y_bounds = (0, None)

res = linprog(c, A_ub=A, b_ub=b, bounds=[x_bounds, y_bounds])
print(f"result of the objective function is: {-res.fun}")
print(f"result of the decision variables is: {res.x}")

print(res.message)
```

```
result of the objective function is: 6350.0
result of the decision variables is: [12. 11.]
Optimization terminated successfully. (HiGHS Status 7: Optimal)
```

Now, let us display the solution in a nice table using Pandas. We are going to first display the solution value using markdown and then we will use Pandas to create a table with the results.

```
[12]: var_df = pd.DataFrame(index=['x', 'y'])
var_df['solution'] = res.x
var_df['coefficients'] = -c
var_df['reduced costs'] = res.lower.marginals

display(var_df)
```

	solution	coefficients	reduced costs
x	12.0	300	0.0
y	11.0	250	0.0

And now another table with the constraints:

```
[13]: con_df = pd.DataFrame(index=['Man hours', 'Machine time', 'Marketing'])
rhs = b
con_df['RHS'] = rhs
con_df['slacks'] = res.slack
con_df['shadow_prices'] = -res.ineqlin.marginals
display(con_df)
```

	RHS	slacks	shadow_prices
Man hours	40	5.0	0.000000
Machine time	45	0.0	83.333333
Marketing	12	0.0	216.666667

### Solved exercises

The following notebooks include exercises solved with Scipy's Linprog:

- Making Chappie Solved with LinProc





## COMBINATORIAL OPTIMIZATION, MIXED INTEGER PROGRAMMING AND NETWORK THEORY

This chapter covers basic theoretical and practical aspects of Mixed integer Programming (MIP) both theoretical and practical topics related to Continuous Linear Programming (CLP) and Network Theory using Python. Like, Continuous Linear Programming, MIP uses linear equations to model optimization problems. However, some of the decision variables can only take Integer values. Feasible solutions are no longer found at vertices of the feasibility region, and this fundamental fact drastically increases the complexity of MIP problems. Network Theory provides a framework rather useful to solve MIP problems, and therefore it is also introduced in this course. CLP allows us to model, solve and analyse the solution of optimization problems using linear equations. “The world is not linear” but still, CLP is a really powerful tool to solve a plethora of real world optimisation problems.

### 3.1 Tutorials

This section contains some basic tutorials:

#### 3.1.1 Combinatorial Optimization

##### Introduction

Combinatorial optimization gathers different types of optimization problems where the decision maker needs to select an alternative from a finite set of possible alternatives with different outcomes, given a set of requirements (i.e. constraints). Some examples of problems in combinatorial optimization are:

- **Grouping:** Find a subset of members within a set that optimizes an objective function (e.g. minimises costs or maximises profits) given a set of constraints. For instance, select the best starting 11 in a football team given their performance statistics.
- **Ordering:** Find the optimal ordering of the members of a set that maximise or minimise an objective function with some constraints. For instance the optimal sequence of production orders that minimises production time.
- **Assignment:** Assign members of a set to members of another set to optimise an objective function given a set of constraints. For instance, assign tasks to employees at a minimum cost

Combinatorial optimization is strongly connected with set theory and the concept of **search**, or (iteratively) explore the set to find the optimal solution. Heuristics will explore the set of alternatives looking for the optimal solution.

The following types of problems are outstanding instances of combinatorial optimization problems:

- **Integer Programming:** Any Linear Programming Problems with Integer or binary variables. When this happens, the feasibility region and objective function are no longer convex, and we cannot apply the same principles and methods as with CLP. When we combine Integer, Binary and Continuous variables, we are dealing with *Mixed*

*Integer Programming (MIP)*. MIP covers a wide range of OR problems, which we will be covering extensively in this unit.

- **Travelling salesman problems:** In the classic formulation of the Travelling Salesman Problem (TSP), a salesman needs to visit a set of cities and wants to find the optimal *route* that would take him through all cities and back to his/her original departure city. Thus, given a list of cities and the distances between each pair of cities, the TSP aims to determine what is the shortest possible route that visits each city and returns to the origin city. The TSP is thus an ordering problem. There are many OR problems, specially in logistics and manufacturing that can be modeled as TSP.
- **Knapsack problems:** In knapsack problems, given a set of items, each with a weight and a value, we need to determine the number of each item to include in a collection so that the total value is maximised while the total weight is less than or equal than the maximum weight the knapsack can hold.
- **Minimum Spanning tree:** Minimum Spanning Trees (MST) problems have a wide number of applications in fields like logistics or telecommunications when we deal with *networks* that interconnect different objects or *edges* through *vertices*. In this context, MST are grouping problems where we want to find a subset of *edges* in a *graph* that interconnects certain *vertices* at a minimum cost. We will address MST problems, among others, in *Network Theory*.

As you can already imagine by now, there is a really wide range of applications for combinatorial optimization. Some examples are **Planning and Scheduling** of activities, **product and process design**, portfolio **selection**, networking and routing (Internet traffic, airlines routes, logistics).

### Types of techniques

Given the importance of the field, scientist have developed many techniques to solve combinatorial optimization problems over the years. They can be classified into the following categories:

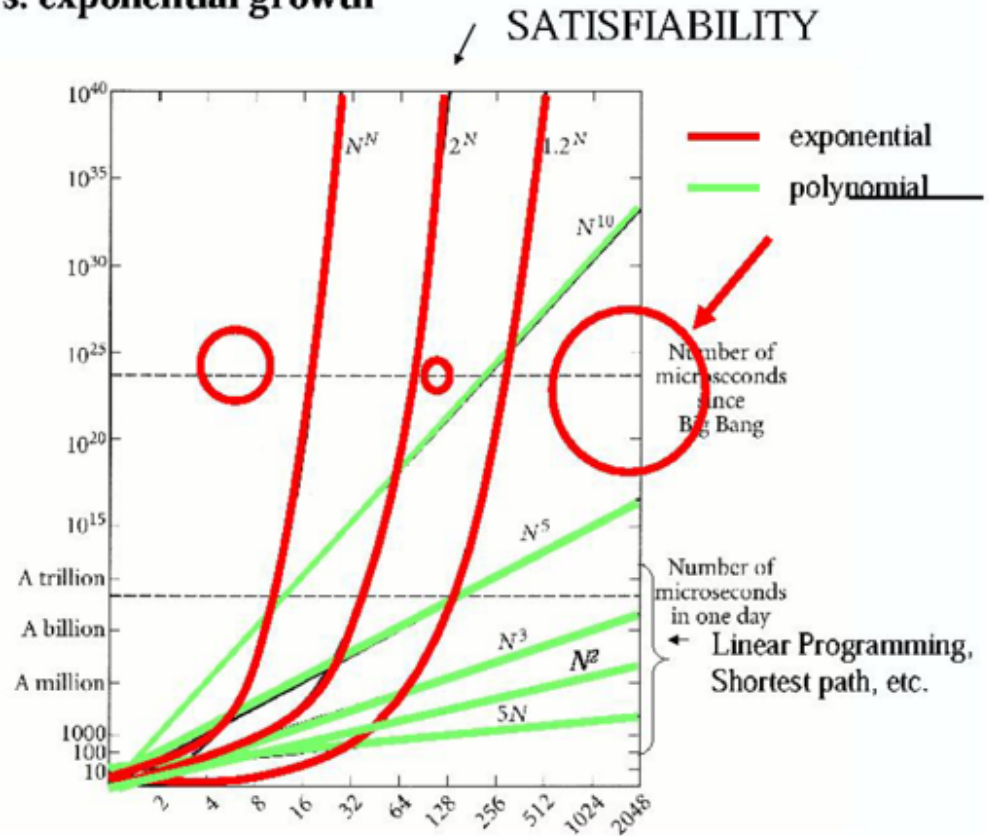
- **Numeric Techniques:** Numeric techniques gather algorithms that provide an optimal solution by exploring all possible options. The simplest example is brute force, although there are many other algorithms that fall into this category. Whenever you use numeric functions like Python's Numpy to compute a minimum, maximum, or ordering of a set, internally the library is applying a numeric technique to compute the result. As we will soon see, for problem instances where the set is large, it might not be computationally efficient to rely solely on numeric techniques, and that is why we have other methods. Numeric techniques are out of the scope of this unit.
- **Problem-solving heuristics:** These techniques find a (sub-)optimal solution in a given number of steps. We will focus on some of these techniques in this unit. Some examples are:
  - **State space search:** These algorithms test successive combinations (or states) of the decision variables iteratively, in such a way that at every step the solution is closer to the optimal solution, until the optimum solution is found. One example of search heuristic is the Branch and Bound algorithm that will be covered in this unit.
  - **Greedy algorithms:** Greedy algorithms first sort the members of the set according to their impact in the objective function, and add them to the solution iteratively.
  - **Genetic algorithms:** Explore the possible solutions combining the features of an initial set of possible solutions, selecting the fittest candidates to combine their features in the next iteration.
- **Graph Theory:** Graph theory is not really a type of technique, but rather, a separate field of mathematics dedicated to the study of relations between objects through mathematical structures known as graphs. Many combinatorial optimization problems can be easily modeled through these structures, and there are many efficient algorithms that effectively compute graph-like structures to find MSTs or **shortest paths** which provide the optimal solution to our problem.

### Efficiency of Techniques

But why do we need this plethora of techniques in the first place? The main reason is that it is not feasible to use numeric techniques in relatively large instances of problems, because the number of possible combinations **grows exponentially** with the number of members in the sets. The image below, taken from *Dynamic logic* by Harel et al., 2000 highlights this fact:

### Polynomial vs. exponential growth

(Harel 2000)



As shown in the graph, for relatively medium instances of combinatorial optimization problems like Integer Programming, the time required to explore all possible combinations might be larger than time itself! We need techniques that need fewer operations to find an optimal solution, or that at least can provide a suboptimal solution in a given interval of time.

Not all techniques provide results in the same number of operations, that is, not all techniques are equally **efficient**. An informal definition of an efficient algorithm is an algorithm in which the time required to solve a problem instance does not increase too fast as the problem size increases. Below, there is a formal definition of efficiency:

An algorithm is efficient (good, polynomial ) if the time necessary to solve any instance of input size  $L$  is bounded above by a polynomial function of  $L$ . (Thus it does not increase as fast as an exponential function such as  $2^L$ .)

The **Big O Notation** is very handy to compare the efficiency of algorithms. For instance, an algorithm that solves a problem of size  $L$  in a time proportional to the square of  $L$  is a polynomial algorithm in  $\mathcal{O}(n^2)$ , where as an algorithm that solves the same problem in an exponential time proportional to  $2^L$  is noted as  $\mathcal{O}(2^L)$ . The latter is less efficient because the time required to solve the problem increases faster as  $L$  becomes larger. An important consideration that needs to be made at this point is that efficiency does not explicitly mention the number of operations, but rather the computing time. This has different implications. The first is that algorithms are formulated in mathematical language, and we need a computing

model to translate operations into computational time. In complexity theory, which is the field of mathematics dedicated to the analysis of the complexity of problems, the **Turing Machine**, which is a mathematical model of a “perfect” machine is used to define time as a function of the number of operations in the definitions above of efficiency above.

Based on this definition of a computational machine model, problems can be classified according to their complexity, in a classification or taxonomy of problems.

### Classification of problems in complexity theory

Based on the definitions above, a problem is as complex as the most efficient algorithm devised to solve it. An algorithm is considered efficient if it can solve an instance of a problem of input size  $L$  in a polynomial function of  $L$ . But what about other problems? Are there different categories of complex problems? Complexity theory is a field of mathematics that focuses precisely on the analysis of the complexity of problems, and its findings help scientist guide the search for new algorithms and mathematical techniques. The following sections describe the different problem classes, which are categories defined for problems according to their complexity.

#### Class Polynomial - Class P (Easy problems)

A problem is in the class P (or polynomially solvable) if it admits a polynomial algorithm. This category gathers all problems for which we have found an algorithm which is able to find an efficient algorithm, able to solve it in a number of operations which is a polynomial function of the size of the problem.

#### Class Non-deterministic Polynomial time - Class NP (Possibly easy problems)

A problem is in the class non-deterministic polynomial (NP) if the problem can be verified in polynomial time, meaning that we can find a bound for the optimal solution in polynomial time. Think for instance of a maximization problem, if we can find an upper bound for the objective function in polynomial time, we might not know which is the optimal solution, but we can already use this information in our problem, as we know that the optimal value will not exceed the upper bound.

From this, we can define the **associated decision problem** as a problem that allows us to verify if a given number is a valid bound for our problem. If the result of the associated decision problem is True (meaning, yes, the provided number is a valid bound), then we can use this information to provide an approximation to the optimal solution.

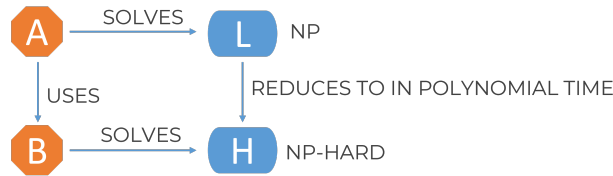
Problems for which there is an algorithm that solves the associated decision problem in polynomial time are classified as NP problems.

#### Class Non-deterministic Polynomial time Hard - Class NP-Hard (Possibly Hard Problems)

#### Reduction

To understand the NP-Hard category, we first need to introduce the concept of **reduction**. A reduction is basically an algorithm that allows us to convert an instance of a problem  $A$  into instances of another problem  $B$ . That is, we can use problem  $B$  to simulate and possibly find ##### Formal definition Formally, a problem  $H$  is NP-Hard if there is a **Polynomial** algorithm to reduce *every* problem  $L$  in class NP to it.

Let us use the following figure to illustrate this:



If a problem  $L$  can be reduced to  $H$  in polynomial time, and we can find a solution for  $H$  using an algorithm  $B$  in one unit of time, then we can solve  $L$  in polynomial time with an algorithm  $A$  which basically reduces  $L$  to  $H$  and then uses  $B$  to solve the problem.

The Traveling Salesman Problem (TSP) is an example of an NP-Hard problem.

Informal definition

Based on the definition above, we can argue that NP-Hard problems are at least as hard as the hardest NP problem, since we can reduce every problem in NP to it in polynomial time, it follows that algorithms for NP-Hard problems are at least as efficient as algorithms for NP problems.

Class Non-Deterministic Polynomial time class Complete - NP-Complete (Hard Problems)

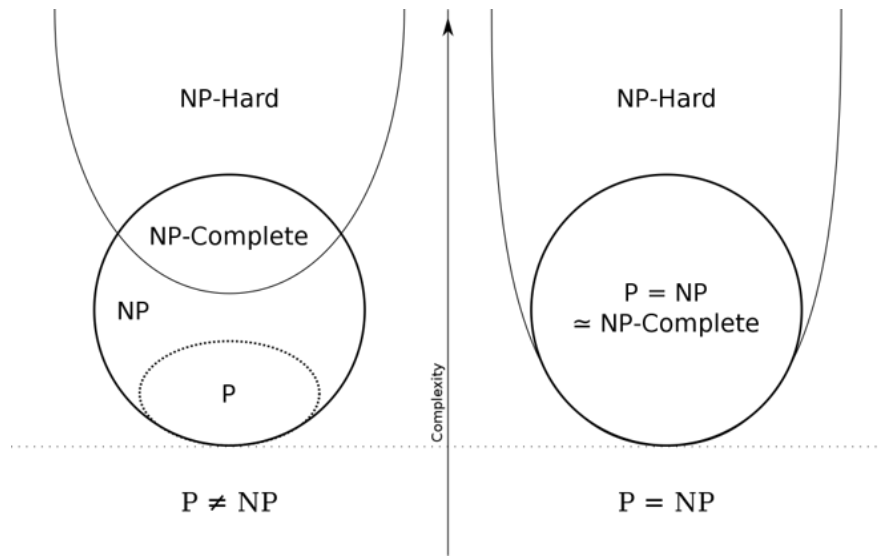
Formally, a problem is NP-Complete if:

- A given solution can be verified in polynomial time (that is, it is an NP Problem).
- It can be used to simulate any other problem for which we can verify that a solution in polynomial time (that is, it is an NP-Hard problem).

It follows that, if a problem is both NP and NP-Hard, it can be used to validate the solution of the *complete* NP class

Note however that although solutions for NP-Problems can be verified in polynomial time, this does not imply that an optimal solution can be found in polynomial time. That is, we can use **heuristics** to verify a bound and use this bound as an approximation of the optimal solution, and we can do this efficiently in polynomial time, but we cannot find an optimal solution in polynomial time, which would in turn imply that the problem is in class P.

In fact, determining that every NP-complete problem is also a class P problem is an open question and one of the [Millennium Prize Problems](#). Since, every NP-complete problem is *complete*, in the sense that it can be used to simulate any NP problem, if a NP-complete problem can be solved in polynomial time, then every NP problem can!



(Source wikipedia by Behnam Esfahbod BY-SA 3.0 Link)

### 3.1.2 Nasty Factorial! Assessing the complexity of an ordering problem

Try me



#### Problem definition

Ordering is not as simple as it seems. Ordering an array of length  $N$  implies finding the right sequence of elements out of  $N!$  possible sequences. The exclamation mark is the factorial operator, and this makes the size of the problem increase really fast with  $N$ . To illustrate how fast, just try this script, it will represent the number of possible orderings of a set of  $N$  elements for  $N$  from 1 to 10:

```
[3]: import numpy as np
from scipy.special import factorial
from matplotlib import pyplot as plt
from datetime import datetime
import ipywidgets as widgets
from ipywidgets import interactive
#We set the mode inline of matplotlib to get the result at the output of the cell code
%matplotlib inline

# Plot the number of possible orderings of a set of N elements (set_size = N)
def plot_factorial(set_size):
    x = np.arange(1, set_size + 1)
    y = factorial(x)
    plt.plot(x,y)
    plt.xlabel("N")
    plt.ylabel("N!")
    plt.title("Number of possible orderings of a set of N elements")
    plt.show()
```

(continues on next page)

(continued from previous page)

```

interactive_plot = interactive(plot_factorial, set_size=(1, 100, 1))
output = interactive_plot.children[-1]
output.layout.height = '350px'
interactive_plot

```

```

[3]: interactive(children=(IntSlider(value=5, description='set_size', max=10, min=1), ...
↳Output (layout=Layout (height='...

```

## Complexity

### Assessing the complexity of the problem

Informally, *complexity* relates to how complex it is to solve a problem. Conceptually, this can be estimated by the number of operations that the computer has to do find a solution. We can assume that in a computer, the time used to do an operation is more or less constant, so we are going to estimate the complexity by measuring the time used to solve the problem. To do this, we are going to use the `datetime` library of Python. This library allows us to take a reference of the time at the beginning of the execution of a piece of code and another reference at the end of the execution. The difference between the two references is the time used to execute the code.

Let us see this in action using the library Numpy. First, using the function `np.random.shuffle()` we are going to generate a random ordering of an array of numbers. Then, we will use the function `np.sort()` to order the array, and using the `datetime` function as described above, we are going to measure the time used to sort the array with `np.sort()`:

```

[4]: N = 10
# Let's generate an array of 10 numbers
x = np.arange(0,N)
# Let's change the order of the numbers randomly
np.random.shuffle(x)
#print(x)
#now, Let's take a reference of the time before sorting
init_time = datetime.now()
# Let's sort, note that the right ordering sequence is 1 out of n!
y = np.sort(x)
#Let's take another reference of the time after sorting and measure the
#difference in seconds:
end_time = datetime.now()
duration = end_time - init_time
print(f'all righty, i ordered the array in {duration.microseconds} microseconds")
# print(y)

all righty, i ordered the array in 0 microseconds

```

Note that, in the result saved in the notebook, it only took a fraction of a second to order an array of 10 numbers! Ok, but what happens if we increase the size of the problem? Let's use the same function to generate arrays of different sizes and measure how the time used to order them increases with the size of the problem:

```

[8]: # Let's generate an array of different sizes
n = [10, 100, 1000, 100000]
#Init the computing time
computing_time = np.zeros(len(n))

# Do the steps in the scripts above for every size in the array n
for i in range(len(n)):
    arr_length=n[i]
    x_arr = np.arange(0,arr_length)

```

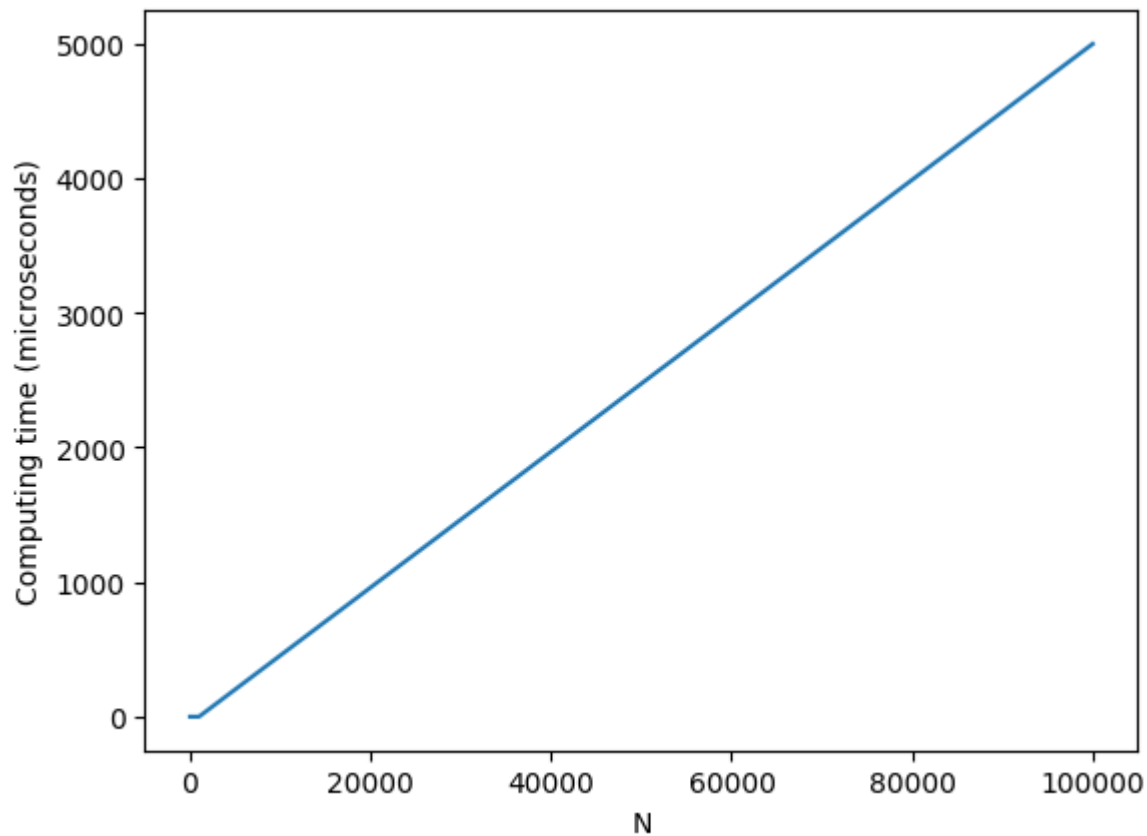
(continues on next page)

(continued from previous page)

```
np.random.shuffle(x_arr)
init_time = datetime.now()
np.sort(x_arr)
end_time = datetime.now()
duration = end_time - init_time
computing_time[i] = duration.microseconds

#Plot the results
plt.plot(n, computing_time)
plt.xlabel("N")
plt.ylabel("Computing time (microseconds)")
```

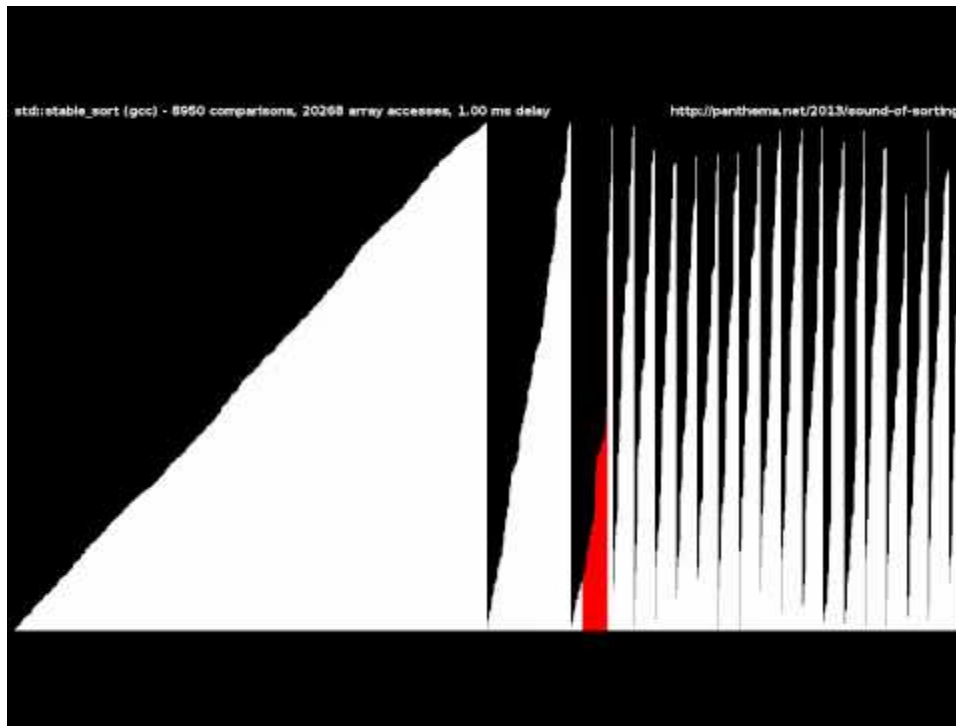
[8]: Text(0, 0.5, 'Computing time (microseconds)')



This simple script shows that the time needed by the sorting algorithm used by Numpy can be approximated to a **linear function** of the size of the set  $N$ . This is a very efficient algorithm!

It is not only efficient, also mesmerizing, in the following video you can see it in action:





We can agree that this problem is not complex, because we have an algorithm that solves it easily. But what would happen if we did not have this algorithm at hand? Imagine that we have to try for instance all the possible orderings of an array of numbers to find the right one. This is called **brute force**. How many operations would we have to do? Let's try to estimate it.

### Assessing the complexity without an algorithm

Let us assume, that we could generate an ordering and check it, really fast, in one microsecond.

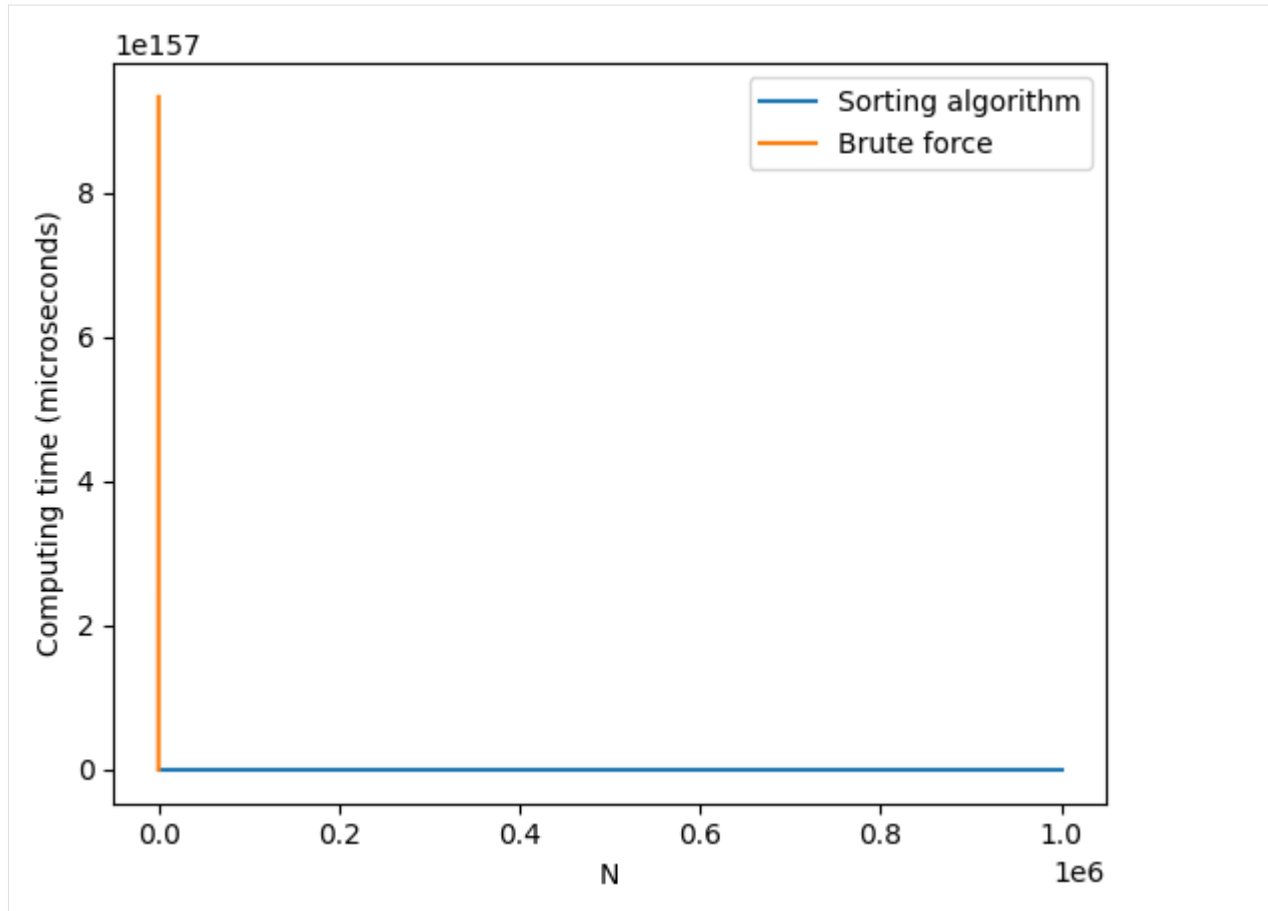
How many orderings would we have to check? Well, we have  $N!$  possible orderings, so we would have to do  $N!$  operations. We can use the following script to estimate the computing time needed to solve the problem using brute force, and compare it with the computing time needed by the sorting algorithm of Numpy:

```
[6]: # Let's calculate the factorial of N
      N_factorial = factorial(n)

      # Since it is in microseconds, we can compare it with the computing time in
      # microseconds obtained before:

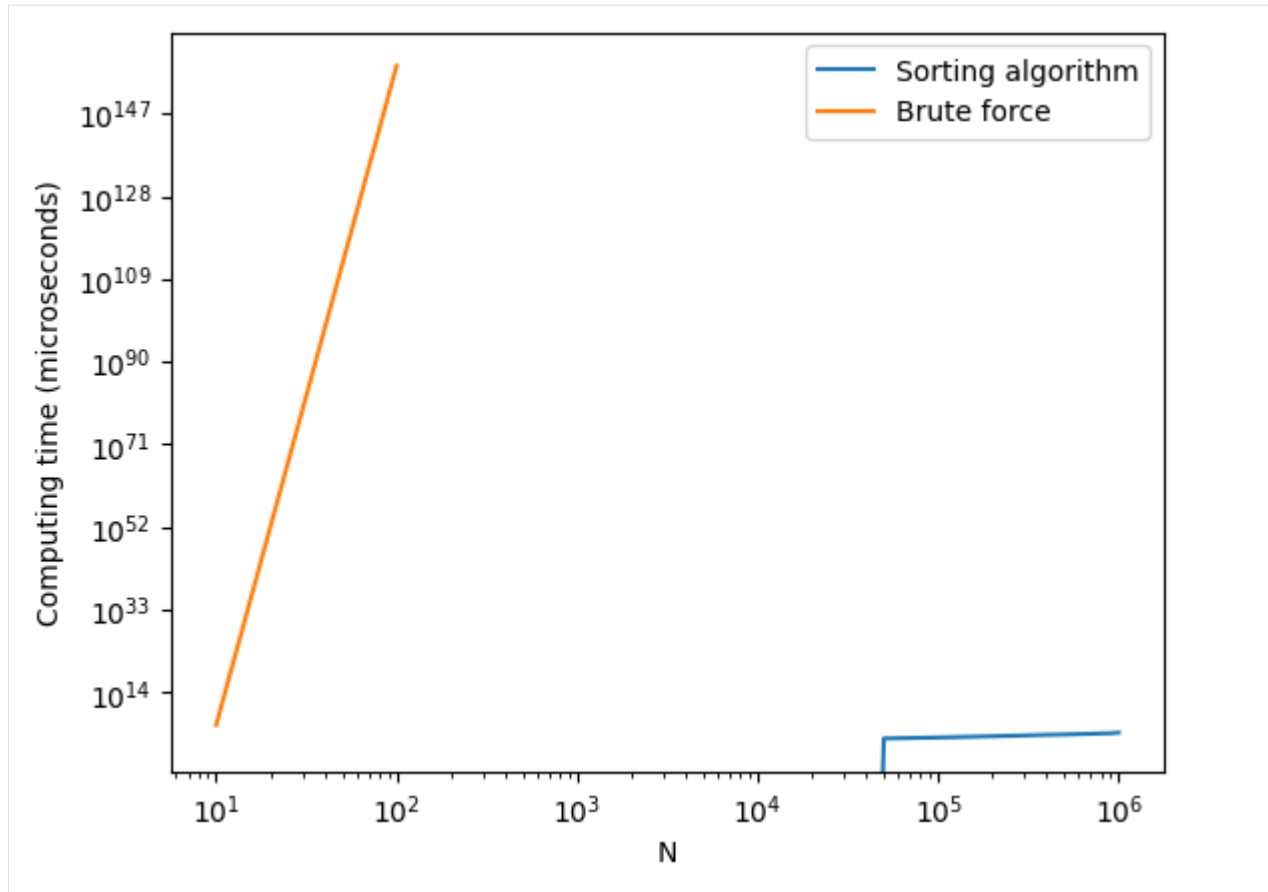
      plt.plot(n, computing_time, label="Sorting algorithm")
      plt.plot(n, N_factorial, label="Brute force")
      plt.xlabel("N")
      plt.ylabel("Computing time (microseconds)")
      plt.legend()
```

```
[6]: <matplotlib.legend.Legend at 0x1bfb0853220>
```



Wow, what happened here? The time needed to solve the problem using brute force is so big that it is not even visible in the plot! This is because the factorial function grows so fast that it is not possible to represent it in a number in Python. By comparison, the time needed by the sorting algorithm is negligible. We could try to plot in a log scale to see the difference:

```
[7]: plt.plot(n, computing_time, label="Sorting algorithm")
plt.plot(n, N_factorial, label="Brute force")
plt.xlabel("N")
plt.ylabel("Computing time (microseconds)")
plt.legend()
plt.yscale("log")
plt.xscale("log")
```



Wow, it is still impossible to visualize both functions in the same plot! Now in log scale, the brute force solution is a straight line, and the sorting algorithm is more or less constant. This means that the time needed by the brute force algorithm grows **exponentially** with the size of the problem. Since the time needed by the sorting algorithm increases **linearly** with the size of the array to sort, we can conclude that the algorithm is very efficient. In fact, every algorithm that has a complexity that grows slower than any polynomial is considered efficient, and it is called a **polynomial algorithm**. In this case, the sorting algorithm is a **linear algorithm**, and sorting an array of is a **polynomial** (or class P) problem.

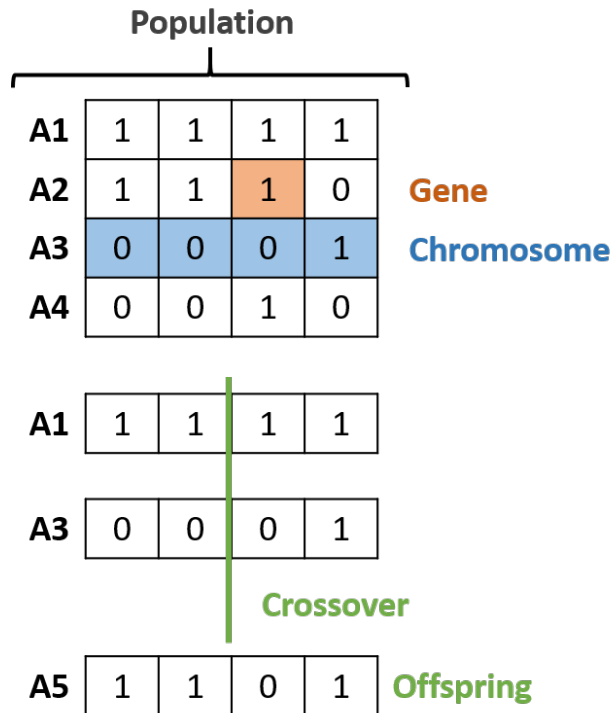
### Analysis questions

Try to answer the following **analysis questions**. You can use AI assistant to help you, but try to answer them by yourself first and analyze critically the answers given by the AI assistant.

1. How many microseconds have you lived? Try to find an estimate of the number of microseconds you have lived, and express it as a factorial number. Based on the results of the previous section, what is the largest array you could sort using brute force in your whole life (assuming that you can check an ordering in one microsecond)?
2. Repeat the previous exercise, but now think of the number of microseconds that have passed since the big bang. What is the largest array you could sort using brute force since the big bang (also assuming that you can check an ordering in one microsecond)?
3. The experiments above might not be accurate, because we provided too few values of N. Modify this line of code `n = [10, 100, 1000, 100000]` to include more values of N, and repeat the analysis. What do you observe?
4. Note that the time needed by the sorting algorithm might not be exactly constant. Why would you modify the script to deal with this issue?

### 3.1.3 Genetic algorithms

Genetic algorithms explore the possible solutions using a method inspired in natural selection. The fundamental components of natural selection are the ability to encode information related to the characteristics of individuals into chromosomes and genes, allowing individuals to mix this information in pairs (by *breeding*) so that the next generation inherits features of both parents, also introducing some mutations so that these new individuals may also have unique features. Then, selection will select only the fittest individuals to produce the next generation. Genetic Algorithms can be used to solve a wide range of problems, including Mixed Integer Programming.



Moreover, genetic algorithms work as follows:

- **Initialisation:** A random function generates an initial population. The *chromosome* is a set of features or genes that represents a solution of a problem. Each individual in the initial population has a different chromosome.
- **Fitness Function:** The objective function is evaluated for each individual to determine which are the fittest individuals in the population. It determines the *fitness score* of each individual. The highest fitness score represents the fittest score of the generation
- **Selection Function:** The individuals with the highest fitness score are selected and paired according to their fitness score
- **Crossover Function:** For every pair, the offspring is created assigning genes from both parents
- **Mutation Function:** A random function changes some genes in the offspring with low probability
- **Termination:** If the algorithm converges at a given generation, new offspring will have no differential features and will not improve the fittest score, so we can terminate the search

### 3.1.4 Solving Sudokus with a Genetic Algorithm

Try me



#### Introduction

Sudoku is a very popular Japanese game invented in the 70s. Basically, the objective is to fill in a grid of 9x9 cells with digits, so that in each column, in each row and in each 3x3 grid there is exactly one digit from 1-9. The following image shows an example of a Sudoku puzzle.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Tim Stellmach - public domain, wikipedia

We can model this problem as a combinatorial optimization problem and apply different combinatorial algorithms and heuristics to solve it. Although it clearly might not be the best alternative, we are going to use this very well known problem to describe how genetic algorithms work and what are the main trade-offs that arise when solving optimization problems with genetic algorithms.

#### Requirements

You need to install Numpy in your Python Kernel runtime. Install Numpy if you are running on Binder or you do not have Numpy installed in your system:

```
[ ]: !pip install numpy
```

## Evolutionary Algorithms in a nutshell

Evolutionary algorithms are a family of algorithms that are inspired by the process of natural selection. The main idea is to represent the solution of the problem as a mathematical object called a **genome**, and to combine different genomes to generate new ones. The genomes that are more fit to the problem are more likely to be selected to generate new genomes, and the process is repeated until a solution is found, or a maximum number of iterations is reached. These are the main steps of an evolutionary algorithm:

- **Initialization:** Generate an initial set of genomes. We will call this set the **population**.
- **Evaluation:** Evaluate the fitness of each genome in the population.
- **Selection:** Select the genomes that are more fit to the problem.
- **Crossover:** Combine the selected genomes to generate new genomes.
- **Mutation:** Mutate the new genomes.
- **Termination:** Check if the termination criteria is met. If not, go to step 2.

Evolutionary algorithms are very flexible, and well-fitted to solve many combinatorial optimization problems. They are also very easy to implement, and they do not require a deep knowledge of the problem to be solved to achieve good results. However, they are not very efficient, and they are not guaranteed to find the optimal solution.

You can find a slightly more extended explanation of Genetic algorithms in this [tutorial](#)

### Approach 1: Naive approach

One of the main advantages of genetic algorithms is that we do not need to have a priori an in-depth knowledge of the problem, we can use simple rules or heuristics to generate the genome of each iteration, and let the fitness function select the fittest alternative. The simplest way to model the Sudoku as a combinatorial optimization problem would be to regard the solution as a permutation of the missing digits. For instance, if we look at the solution of the example above:

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Tim Stellmach - public domain, wikipedia

We can actually assign to every unknown cell an index from 1 to the number of unknowns, from left to right and top to bottom, such that the solution is the ordered set:

$s = (s_1, s_2, \dots, s_N) = (4, 6, 8, 9, 1, 2, 7, 2, 3, 4, 8, 1, 3, 4, 2, 4, 7, 5, 9, 7, 1, 4, 2, 2, 6, 5, 7, 9, 1, 3, 9, 4, 8, 5, 9, 1, 5, 3, 7, 4, 2, 8, 7, 6, 3, 3, 4, 5, 2, 6, 1)$

At the beginning of the problem, we do not know the exact value that we need to assign to each cell, that is, we do not know that  $s_1 = 4$  or that  $s_5 = 5$ , but we know that it will be a value that is missing, and it is very easy to calculate the values that are missing, just by counting the digits that are present, because we know that each digit appears nine time in the solution. So, with this simple procedure, we can determine at the beginning of the problem which are the digits that are missing:

$c = \{1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 6, 6, 6, 6, 7, 7, 7, 7, 7, 7, 8, 8, 8, 8, 9, 9, 9, 9, 9\}$

And we know that the solution  $s$  is going to be a **permutation** of these elements!

### Assessment of the search space

We need to find the right permutation though, and there are  $N!$  number of permutations, where  $N$  is as noted above the total number of missing digits in our problem. In this case  $N = 51$  so there are:

$$N! = 51! = 155111875328738228022424301646930321106325972001698611200000000000$$

possible combinations, which are many! Luckily for us, there are **repetitions**, since some digits are repeated. Considering that there are 6 1's, 7 2's, 6 3's, 8 4's, 5 5's, 4 6's, 6 7's, 4 8's and 5 9's, we can calculate the number of permutations with repetitions as:

$$\frac{51!}{6! \cdot 7! \cdot 6! \cdot 8! \cdot 5! \cdot 4! \cdot 6! \cdot 4! \cdot 5!} = 2465536997482928914408092530982635167547392 = 2.47 * 10^{42}$$

Which is still a really large number of possible combinations! The question is, are they too many for a genetic algorithm to find a solution? We will find out soon enough, but first, let's discuss how our genetic algorithm is going to work. Step by step:

### Initialisation

The first step is to generate an initial **genome** (all genes of a set of possible solutions). A gene is going to represent the value of a digit in a position of the Sudoku, and all the genes or chromosome of a solution, all the values of all digits missing in the problem. Basically, to generate the genome of the first population, we can generate just as many permutations of  $c$  as the size of the initial population. Say the initial population size is  $P$ , so initialising the genome is basically to obtain  $P$  permutations of  $c$ . Let us define a function  $\tau(c)$  that randomly re-arranges the items of an array  $c$  and therefore returns a random permutation out of all possible permutations. so, our initial genome  $G_0$  is going to be just a set of permutations of  $c$ :

$$G_0 = [g_{01}, g_{02}, \dots, g_{0P}]$$

$$g_{0k} = \tau(c) \quad \forall k \in [1, P]$$

As a reference, the numpy function **shuffle** provides such a re-arrangement.

### Fitness function

Now, the second step is to evaluate the fitness function. We can define a fitness function that counts the number of duplicated digits in every row, in every column, and in every 3x3 block. That is, if a digit appears two times in a row, the fitness function will add 1, and if a digit appears twice in a column, it will add 1, same if it appears twice in a block. So let us define this fitness function as  $f(g_{ik})$  and obviously the result is going to be an integer number. Furthermore, the *fitnessarray*:

$$f_0 = [f(g_{01}), f(g_{02}), \dots, f(g_{0P})]$$

will contain the fitness of the different candidate solutions in the initial population, and by extension:

$$f_i = [f(g_{i1}), f(g_{i2}), \dots, f(g_{iP})]$$

will contain the fitness of the different candidate solutions in the population at generation  $i$ .

### Selection function

To select the fittest, we just need to select the candidates that contain fewer mistakes, those where the fitness solution is lower. We just need to sort the elements of the fitness array in ascending order and select some of them for the next generation. Let us note the selection function as  $\sigma(f_i)$  and as  $S$  the number of individuals that will produce offspring in the next generation. The set of survivals in a generation  $i$  is:

$$s_i = \sigma(f_i) = [s_{i1}, s_{i2}, \dots, s_{iS}]$$

whereas described, the elements of  $s$  will be the  $S$  elements of  $G_i$  with the lowest fitness function value.

### Crossover and mutation

We could randomly select two parents from the survivals and combine values in the solution, but we need to ensure that the result is still a permutation of  $c$ , so for the sake of simplicity, let us not implement any crossover function and focus on the generation of mutations of surviving individuals. To generate a mutation, what we can do is copy some values randomly from the parent in the exact same position as they are in the parent, and shuffle the remaining values. For instance, imagine that our problem was very easy and only missing 3 values. We could select randomly 1 out of the 3 values and copy it in the exact same location and generate a permutation of the other two elements. If the parent had all the digits of a valid solution, so will the children, and if we are lucky, the permutation will yield a valid solution. Let us note as  $R$  the number of digits (or genes) of the parent that we copy, and  $N - R$  the number of genes that mutate. Note that, if we select randomly the  $(R)$  genes out of the  $N$  genes, this is a combination, and therefore, the number of possible combinations of  $R$  genes out of the parent is:

$$C_R^N = \frac{N!}{R!(N - R)!}$$

And now, we are going to shuffle the  $(N - R)$  remaining genes, so there are  $(N - R)$  different permutations of these genes, that combined with the previous result, yield the following total possible different offspring from a single parent:

$$\frac{N!}{R!}$$

For instance, the example above, with  $N = 51$  gaps or blank positions, if we set  $R = 49$ , there are:

$$C_{49}^{51} = \frac{N!}{R!(N - R)!} = \frac{51!}{49!(51 - 49)!} = \frac{51 * 50 * 49!}{49! * 2!} = \frac{51 * 50}{2!} = 1275$$

possible ways to combine the genes, and if we multiply by  $2!$ , we get 2500 different combinations of a parent. Now, this is only for 1 parent, if the selection function lets  $S$  survivals, we need to multiply by this factor as well to get the total size of the possible offspring, so for instance, if we set it to 10, the possible different members in a generation is 25000.

We really need to fine tune these parameters to make sure that the algorithm is fast, because we need to make sure that we explore a sufficient proportion of all the possible offspring in the iterations. For instance if  $R$  is too small, the number of possible children will be too large compared to the population size, and it will be very difficult for the algorithm to find the right solution. Additionally,  $(N - R)$  must be at least 2, so that we can at least switch two digits to generate another solution. On the other hand, if the population size is too large compared to the possible offspring our algorithm is inefficient because the population size is too large compared to the possible different offspring, and we will be evaluating several times the same combination of genes in every generation. As a rule of thumb, You need to make sure that the total number of different offspring in a generation is larger than the population size but in the same order of magnitude.



## Termination

Finally, we will terminate when we reach a maximum number of iterations, or when we have found the optimal solution. We also need to ensure that we can get rid of *bad batches*, meaning that something can go wrong in the mutations and our algorithm can converge to a solution that can not be fixed with a permutation of N-R numbers of all candidate solution, that is the fitness function can have *local minimum* and our solution can converge to a local minimum, which is not optimal, so if we detect for instance that the fitness function does not change in a number of iterations, we just go back to step 1 and initialise the entire population again. We also need to set another parameter to control the reset, which will represent the maximum number of iterations without improvement in the fitness function.

## Implementation

Ok, the following Python script implements this approach. To try it, you just need to provide the values of your Sudoku as a Python Array where the digits in [1-9] define the clues or filled digits in your Sudoku, and the 0s represent the gaps.

### Hyper-parameters

The parameters that control the behaviour of the algorithm, or hyperparameters are as described above: - Number of genes that are passed to children in each generation (R). Default: N-R = 2 - Population size, the population size of each generation (P), default: 500 - Number of iterations, default: 1000: - Number of survivals in each generation, default 10 - Number of iterations without improvement, default 10

```
[3]: from random import sample
import numpy as np
from datetime import datetime
from matplotlib import pyplot as plt

# digits from 1 to 9
digits = np.arange(1,10)

problem = np.array([[8, 0, 6, 2, 0, 9, 0, 1, 5],
                   [9, 5, 0, 3, 0, 8, 7, 0, 0],
                   [4, 0, 0, 1, 0, 0, 3, 0, 9],
                   [7, 9, 3, 4, 8, 5, 0, 0, 0],
                   [0, 0, 0, 7, 0, 0, 0, 0, 4],
                   [0, 8, 4, 6, 0, 3, 9, 5, 7],
                   [0, 0, 0, 0, 3, 4, 2, 0, 0],
                   [3, 0, 9, 0, 0, 0, 1, 0, 8],
                   [6, 0, 7, 0, 2, 0, 0, 0, 3]])

# clues_idx is a tuple that contains the coordinates of the clues.
# It can be used for indexing non-zero values in the problem
clues_idx = np.nonzero(problem)

# Conversely, gaps_idx is a tuple that contains the coordinates of the zeros.
# It is used for indexing the values we need to fill in.

gaps_idx = np.nonzero(problem == 0)

# We know that a valid solution is a permutation of the digits from 1 to 9 repeated 9
# times. Let us find the digits that
# are missing. digit_count counts, starting with zero the number of digits present in
# the problem:
digit_count = np.bincount(problem.flatten())
```

(continues on next page)

(continued from previous page)

```

# the size of the problem (N):
problem_size = digit_count[0]

# HYPER-PARAMETERS
# Hyper-parameter R, default: N-R = 2
in_chromosome_size = problem_size - 2

# Hyper-parameter 2, population size, default: 500
population_size = 500

# Hyperparameter 3, number of iterations, default: 1000:
num_iter = 1000

# Hyperparameter 4, number of survivals in each generation: default 10
num_survivals = 10
# Hyperparameter 5, Number of iterations without improvement: default: 10
reset_it = 10

# The missing values are going to be 9 - the count of digits, the zeros can be
↳ ignored:
missing_count = 9 - digit_count

# OK, Let's start
init_time = datetime.now()

# Let's create a numpy array with the digits that we need to fill in:
candidate_digits = np.array([], dtype='int32')

for digit in digits:
    candidate_digits = np.append(candidate_digits, np.repeat(digit, missing_
↳ count[digit]))

def is_permutation(elements):
    # returns 0 if row is a permutation of digits from 1 to 9
    return 9 - len(np.unique(elements))

def fitness(solution):
    res = 0
    # check that rows are consistent
    for i in range(9):
        res += is_permutation(solution[i])

    # check that columns are consistent
    for j in range(9):
        res += is_permutation(solution[:,j])

    # check that blocks are consistent
    for k in range(9): #      0, 1, 2, 3, 4, 5, 6, 7, 8
        row_index = 3*(k//3) # 3 * (0, 0, 0, 1, 1, 1, 2, 2, 2)
        col_index = 3*(k%3)  # 3 * (0, 1, 2, 0, 1, 2, 0, 1, 2)
        res += is_permutation(solution[row_index:row_index + 3, col_index:col_index+3])

    # check consistency with problem definition
    # indices of clues
    res += sum(abs(problem[clues_idx]-solution[clues_idx]))
    
```

(continues on next page)

(continued from previous page)

```

# check if solution is complete
res += np.sum(solution == 0)
return res

# our genome consists of shuffling the candidate digits and placing the digits in the
# gaps

genome = np.zeros((population_size, 9, 9))

for k in range(population_size):
    # shuffle the candidates
    np.random.shuffle(candidate_digits)

    #init candidate as problem
    genome[k, :, :] = np.copy(problem)
    #fill in sudoku, by placing the digits where the problem is zero
    np.place(genome[k, :, :], genome[k, :, :] == 0, candidate_digits)

# The simplest genome generation strategy is however to just fill the zeros with
↪digits
# np.place(genome, genome==0, [1, 2, 3, 4, 5, 6, 7, 8, 9])

it = 0
iteration_error = np.zeros(num_iter)
while it < num_iter:
    # reset the fitness function
    fitness_array = np.zeros(population_size)

    for k in range(population_size):
        # calculate fitness function of candidate k
        fitness_array[k] = fitness(genome[k, :, :])

    # Find the fittest!
    fitness_array_sorted_idx = np.argsort(fitness_array)
    fittest = fitness_array_sorted_idx[0:num_survivals]

    iteration_error[it] = fitness_array[fittest[0]]
    reset = (it >= reset_it) and (sum(abs(iteration_error[it-reset_it:it] - iteration_
↪error[it])) == 0)
    if reset:

        digit_count = np.bincount(problem.flatten())

        # The missing values are going to be 9 - the count of digits, the zeros can
↪be ignored:
        missing_count = 9 - digit_count

        # Let's create a numpy array with the digits that we need to fill in:
        candidate_digits = np.array([], dtype='int32')

        for digit in digits:
            candidate_digits = np.append(candidate_digits, np.repeat(digit, missing_

```

(continues on next page)

(continued from previous page)

```

↪count[digit]))

    genome = np.zeros((population_size, 9, 9))

    for k in range(population_size):
        # shuffle the candidates
        np.random.shuffle(candidate_digits)

        #init candidate as problem
        genome[k, :, :] = np.copy(problem)
        #fill in sudoku, by placing the digits where the problem is zero
        np.place(genome[k, :, :], genome[k, :, :] == 0, candidate_digits)

    continue

sol = genome[fittest[0],:,:]
# print('Iteration: ' + str(it))
# print(iteration_error[it])
# print(sol)    # We might have found already a solution

if fitness_array[fittest[0]] == 0:
    break
else:
    # Let us create the next generation
    # take ten survivals
    mutations = np.zeros((population_size,9,9))
    # randomly combine rows of the best solutions
    for k in range(population_size):
        # Init mutation_k as problem
        mutation_k = np.copy(problem)

        # we select the parent randomly from the survivals
        np.random.shuffle(fittest)
        parent_a = genome[fittest[0],:,:]

        # we first select some values to fill in randomly
        pos_to_fill_in_a = np.random.choice(np.arange(problem_size), 95*problem_
↪size//100, replace=False)

        mutation_idx_a = (gaps_idx[0][pos_to_fill_in_a], gaps_idx[1][pos_to_fill_
↪in_a])

        # we fill in the values selecting them from the parent
        mutation_k[mutation_idx_a] = parent_a[mutation_idx_a]

        # Now we fill in the rest of the values randomly
        digit_count = np.bincount(mutation_k.flatten())

        missing_count = 9 - digit_count
        candidate_digits = np.array([], dtype='int32')
        for digit in digits:
            candidate_digits = np.append(candidate_digits, np.repeat(digit,
↪missing_count[digit]))

        np.random.shuffle(candidate_digits)
    
```

(continues on next page)

(continued from previous page)

```
np.place(mutation_k, mutation_k == 0, candidate_digits)

mutations[k,:,:] = np.copy(mutation_k)

# copy mutation to genome
genome = np.copy(mutations)

# We increase the number of iterations
it+=1

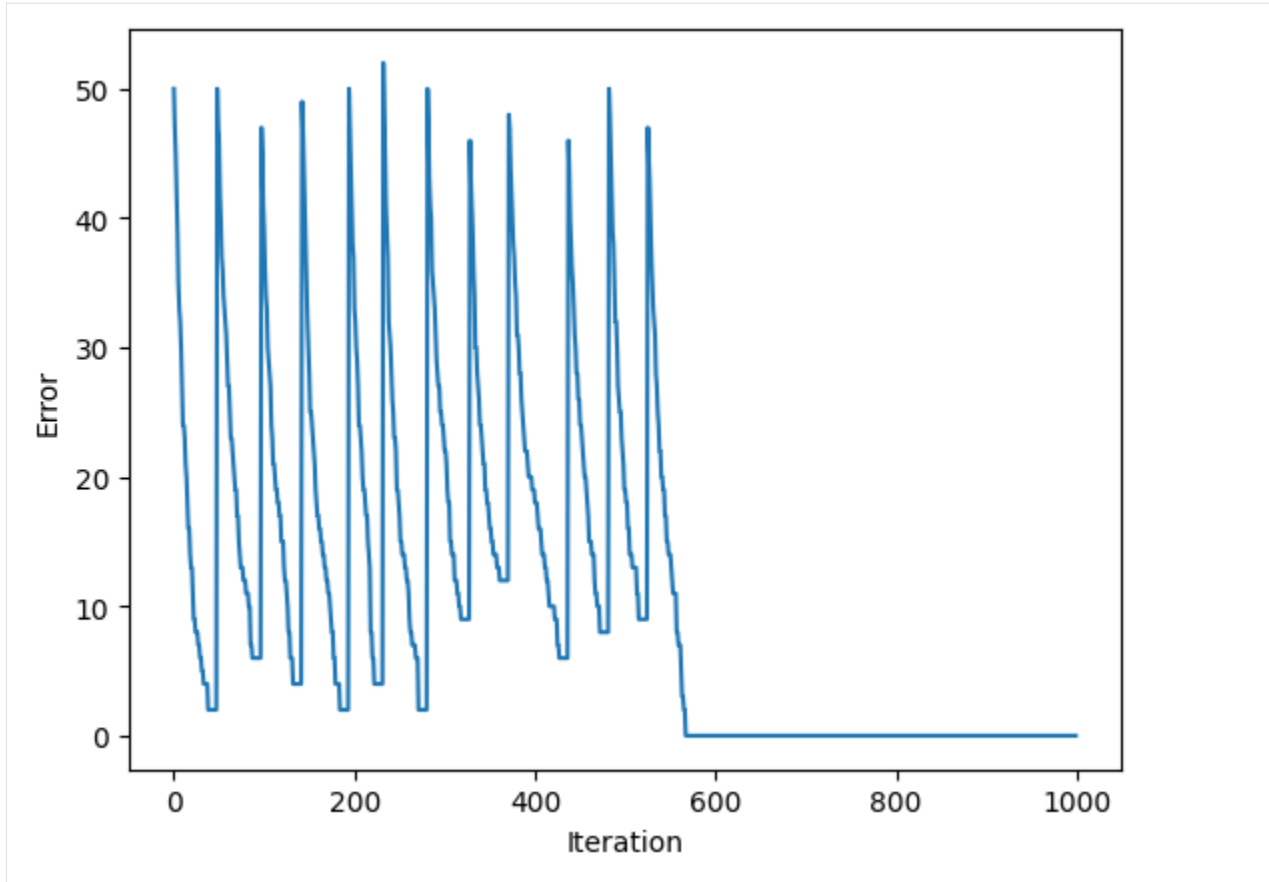
end_time = datetime.now()
duration = end_time - init_time
print("all righty, i finished the Sudoku in {0} seconds and {1} iterations: ".
      ↪format(duration.total_seconds(), it))
print("Best solution is:")
print(sol)
plt.plot(iteration_error)
plt.xlabel('Iteration')
plt.ylabel('Error')
```

```
all righty, i finished the Sudoku in 84.685776 seconds and 567 iterations:
```

```
Best solution is:
```

```
[[8. 3. 6. 2. 7. 9. 4. 1. 5.]
 [9. 5. 1. 3. 4. 8. 7. 6. 2.]
 [4. 7. 2. 1. 5. 6. 3. 8. 9.]
 [7. 9. 3. 4. 8. 5. 6. 2. 1.]
 [1. 6. 5. 7. 9. 2. 8. 3. 4.]
 [2. 8. 4. 6. 1. 3. 9. 5. 7.]
 [5. 1. 8. 9. 3. 4. 2. 7. 6.]
 [3. 2. 9. 5. 6. 7. 1. 4. 8.]
 [6. 4. 7. 8. 2. 1. 5. 9. 3.]]
```

```
[3]: Text(0, 0.5, 'Error')
```



### Questions

1. Run the algorithm above and write down the number of iterations and the execution time. Did the algorithm find an optimal solution? You can try again if you think the algorithm did not find an optimal solution. Do you think the algorithm is better than a human? (spoiler alert: the algorithm is not very good and will probably need several attempts to find a solution)
2. Reason with an AI assistant about the algorithm, and ask for an explanation of the code in the cell above in simple terms.
3. As described in the explanation of this approach, the algorithm may converge to a *local minimum*, a solution with a low number of errors, but that cannot be fixed with a permutation of digits. If the algorithm detects that there is no improvement, it will reset the entire population and start again (and we will see a drastic increase in the error). Looking at the graph that shows the error against the iterations, do you think the algorithm reset the entire population at any time? How many times do you think the reset function was executed?
4. This is a naive approach because to generate the solution, Ask an AI assistant what is the search space of the problem, and how many possible solutions there are using this naive approach. Now, look for the assessment of the search space of the problem in the section above and use this prompt template to ask the AI assistant again: > I found the following information about the search space of the problem: [Look for the assessment of the complexity of the problem in the section above and paste it here]. Can you tell explain it to me in simple terms?
5. The search space is just vast. As an AI assistant answer how big is it using this prompt:  
How extraordinarily large is the number of possible combinations obtained? Can you provide me some illustrative examples? Be imaginative!

## Approach 2: Columns approach

Approach 1 is naive in the sense that it does not require an in-depth knowledge of Sudoku to program the algorithm, but obviously, if we have some knowledge of the problem at hand, we can deliver more efficient solutions, but they might come at the expense of complexity, and therefore costs, development costs and/or computational costs.

Let us again look at the solution of the problem above:

[f88b4252b97f45c3b7267a17da84d8cf|Tim Stellmach - public domain, wikipedia](https://www.wikiwand.com/en/Tim_Stellmach)

We know that each column is a permutation of the digits in [1-9], so, what if instead of assigning to every unknown cell an index, we use two indices, one for the columns, and another one for the rows. for instance, now our solution will be

- Column 1:  $[s_{31}, s_{71}, s_{71}, s_{71}] = [1, 9, 2, 3]$
- Column 2:  $[s_{22}, s_{42}, s_{52}, s_{62}, s_{82}, s_{92}] = [7, 5, 2, 1, 8, 4]$
- Column 3:  $[s_{13}, s_{23}, s_{43}, s_{53}, s_{63}, s_{73}, s_{83}, s_{93}] = [4, 2, 9, 6, 3, 1, 7, 5]$
- Column 4:  $[s_{14}, s_{34}, s_{44}, s_{64}, s_{74}, s_{94}] = [6, 3, 7, 9, 5, 2]$
- Column 5:  $[s_{35}, s_{55}, s_{75}] = [4, 5, 3]$
- Column 6:  $[s_{16}, s_{36}, s_{46}, s_{56}, s_{96}] = [8, 2, 1, 4, 7, 6]$
- Column 7:  $[s_{17}, s_{27}, s_{37}, s_{47}, s_{57}, s_{67}, s_{87}, s_{97}] = [9, 3, 5, 4, 7, 8, 6, 1]$
- Column 8:  $[s_{18}, s_{28}, s_{48}, s_{58}, s_{68}, s_{88}] = [1, 4, 2, 9, 5, 3]$
- Column 9:  $[s_{19}, s_{29}, s_{39}, s_{79}] = [2, 8, 7, 4]$

Again, even if at the beginning of the problem we do not know the exact value that we need to assign to each cell, it is very easy to know which values are going to be missing in each column, and we know that in the solution, the values in each column is going to be a permutation of the missing values. So, now, our problem consists of finding 9 permutations of different sizes! what is now the number of potential solutions, the search space? well, it depends on the problem, but in this specific case, we need to multiply the number of possible combinations in each row, therefore the number of candidate solution in this case is:

$$N_1! * N_2! * N_3! * N_4! * N_5! * N_6! * N_7! * N_8! * N_9! = 4! * 6! * 8! * 6! * 3! * 6! * 8! * 3! * 4! = 12582400952578867200000$$

Compared to alternative 1:

$$\frac{51!}{6! \cdot 7! \cdot 6! \cdot 8! \cdot 5! \cdot 4! \cdot 6! \cdot 4! \cdot 5!} = 2465536997482928914408092530982635167547392$$

The search space is now tiny!

### Initialisation

Again, the first step is to generate an initial **genome** (all genes of a set of possible solutions). A gene is going to represent the value of a digit in a position of the Sudoku, and all the genes or chromosome of a solution, all the values of all digits missing in the problem. Basically, to generate the genome of the first population, we can generate just as many permutations of the values missing in each column as the size of the initial population. Let us note as  $c_j \quad j \in [1 - 9]$  the set of missing values in column  $j$ . Basically, we need to obtain  $P$  permutations of every  $c_j$ .

We can use the same function  $\tau(c_j)$  to re-arranges the items of an array  $c$  and returns a random permutation out of the  $N_j!$  possible permutations. so, our initial genome  $G_0$  is going to be  $P$  different sets of permutations of the 9 columns:

$$G_0 = [g_{01}, g_{02}, \dots, g_{0P}]$$

$$g_{0k} = [\tau(c_j) \quad \forall k \in [1, P] \forall j \in [1 - 9]]$$

Again, the numpy function `shuffle` provides such a re-arrangement.

### Fitness function

We can use the same fitness function as in the previous approach.

### Selection function

We can apply the same selection function as in the previous approach.

### Crossover and mutation

Now, let us take a different approach for the cross-over, what we are going to do is to take a number of columns from the parent, and copy them, and then create a mutation by shuffling some values of the remaining columns.

Let us note as  $R_c$  the number of columns that are passed to the children without mutations. Again, this is a combination of the 9 columns, and the number of possible combinations is:

$$C_{R_c}^9 = \frac{9!}{R_c!(9-R_c)!}$$

For instance, if we set this parameter to 8, there are 9 possible combinations.

Now, we are going to mutate the remaining columns. Let us refer to column  $j$  to one of the columns that will mutate. In this case, Let us note as  $N_j$  the unknown digits in column  $j$ ,  $R_j$  the number of digits (or genes) of the parent that we copy to column  $j$ , and  $N_j - R_j$  the number of genes that mutate in that column. There are:

$$C_{R_j}^{N_j} = \frac{N_j!}{R_j!(N_j-R_j)!}$$

Possible combinations of genes and  $(N_j - R_j)!$  permutations of the genes that mutate. So, again, there are:

$$C_{R_j}^{N_j} * (N_j - R_j)! = \frac{N_j!}{R_j!}$$

Again, this is a combination, but note that, now, the index of the column  $j$  is random (we did a random selection of columns), so the maximum number of digits missing are found in columns 3 and 7 that are both missing 8 numbers. So, for instance, if we set  $R_c$  to 8, so that we copy 8 columns from the parent, there will be 9 combinations of columns to pass as-is to the next generation. Now, say we fix  $(N_j - R_j)$  to 2, so that we shuffle at most 2 values. We will have, at most:

$$9 * \frac{8!}{6!} = 9 * 8 * 7 = 504$$

Possible ways to mutate a parent if the algorithm selects to mutate column 3 or column 7. And if we select 10 survivals, we may get an estimation of the total number of offspring as 5040.

There are two things that we need to observe here. First is that the number of combinations is significantly smaller, just as the search space, so we can deal with smaller populations in every generation, which will make each iteration faster, or take another strategies, like increasing the number of survivals in each generation, or the number of columns that we shuffle, to explore the search space faster.

The other important thing is that there is uncertainty in the total number of possible offspring, because not all the columns are missing the same number of values and, since the parameters of the algorithm are fixed, the algorithm is not going to be as stable as in the first approach.



## Termination

We will terminate using the same approach as before.

## Implementation

Ok, the following Python script implements this approach. To try it, you just need to provide the values of your Sudoku as a Python Array where the digits in [1-9] define the clues or filled digits in your Sudoku, and the 0s represent the gaps.

### Hyper-parameters

The parameters that control the behaviour of the algorithm, or hyperparameters: - Number of columns that are passed to the next generation ( $R_c$ ). Default: 8 - Number of genes that are passed to children in each generation (R). Default: 2 - Population size, the population size of each generation (P), default: 50 - Number of iterations, default: 1000: - Number of survivals in each generation, default 10 - Number of iterations without improvement, default 100

```
[4]: from random import sample
import numpy as np
from matplotlib import pyplot as plt
from datetime import datetime

# digits from 1 to 9
digits = np.arange(1,10)

problem = np.array([[8, 0, 6, 2, 0, 9, 0, 1, 5],
                   [9, 5, 0, 3, 0, 8, 7, 0, 0],
                   [4, 0, 0, 1, 0, 0, 3, 0, 9],
                   [7, 9, 3, 4, 8, 5, 0, 0, 0],
                   [0, 0, 0, 7, 0, 0, 0, 0, 4],
                   [0, 8, 4, 6, 0, 3, 9, 5, 7],
                   [0, 0, 0, 0, 3, 4, 2, 0, 0],
                   [3, 0, 9, 0, 0, 0, 1, 0, 8],
                   [6, 0, 7, 0, 2, 0, 0, 0, 3]])

# clues idx is a tuple that contains the coordinates of the clues.
# It can be used for indexing non-zero values in the problem
clues_idx = np.nonzero(problem)

# Conversely, gaps_idx is a tuple that contains the coordinates of the zeros.
# It is used for indexing the values we need to fill in.

gaps_idx = np.nonzero(problem == 0)

# We know that a valid solution is a collection of permutation of the digits from 1
↳to 9 repeated 9 times. Let us find the digits that
# are missing. digit_count counts, starting with zero the number of digits present in
↳the problem:
digit_count = np.bincount(problem.flatten())

# the number of gaps:
problem_size = digit_count[0]

# Hyper-parameters

# Population size
population_size = 50
```

(continues on next page)

(continued from previous page)

```

# columns (chromosomes) to pass to next generation (R_c)
keep_cols = 8

# maximum number of values to shuffle (N-R)
cells_to_shuffle = 2

# Number of iterations
num_iter = 1000

# number of survivals
num_survivals = 10

# maximum number of iterations without improvement
reset_it = 100

# The missing values are going to be 9 - the count of digits, the zeros can be
→ ignored:
missing_count = 9 - digit_count

def get_missing_per_column(column):
    column_count = np.bincount(column)
    column_count_padded = np.zeros(10)
    column_count_padded[0:len(column_count)] = column_count
    return digits[column_count_padded[1:]==0]

def get_missing(prob):
    m_per_column = np.zeros((9,9), dtype='int32')
    for j in range(9):
        missing_values = get_missing_per_column(prob[:,j])
        m_per_column[0:len(missing_values), j] = missing_values
    return m_per_column

def is_permutation(elements):
    # returns 0 if row is a permutation of digits from 1 to 9
    return 9 - len(np.unique(elements))

def fitness(solution):
    res = 0
    # check that rows are consistent
    for i in range(9):
        res += is_permutation(solution[i])

    # check that columns are consistent
    for j in range(9):
        res += is_permutation(solution[:,j])

    # check that blocks are consistent
    for k in range(9): # 0, 1, 2, 3, 4, 5, 6, 7, 8
        row_index = 3*(k//3) # 3 * (0, 0, 0, 1, 1, 1, 2, 2, 2)
        col_index = 3*(k%3) # 3 * (0, 1, 2, 0, 1, 2, 0, 1, 2)
        res += is_permutation(solution[row_index:row_index + 3, col_index:col_index+3])

    # check consistency with problem definition
    # indices of clues
    
```

(continues on next page)

(continued from previous page)

```

res += sum(abs(problem[clues_idx]-solution[clues_idx]))

# check if solution is complete
res += np.sum(solution == 0)
return res

init_time = datetime.now()

# Now our genome consists of shuffling the candidate digits of every row and placing
↳the digits in the
# gaps

genome = np.zeros((population_size, 9, 9))

# Initial missing values in problem
missing_per_column = get_missing(problem)

for k in range(population_size):

    #init candidate as problem
    genome[k, :, :] = np.copy(problem)

    #fill in sudoku, by placing the digits where the problem is zero
    for j in range(9):
        candidate_digits_j = missing_per_column[missing_per_column[:, j]>0,j]
        np.random.shuffle(candidate_digits_j)
        np.place(genome[k, :, j], genome[k, :, j] == 0, candidate_digits_j)

# The simplest genome generation strategy is however to just fill the zeros with
↳digits
# np.place(genome, genome==0, [1, 2, 3, 4, 5, 6, 7, 8, 9])

it = 0
iteration_error = np.zeros(num_iter)
while it < num_iter:
    # reset the fitness function
    fitness_array = np.zeros(population_size)

    for k in range(population_size):
        # calculate fitness function of candidate k
        fitness_array[k] = fitness(genome[k,:,:])

    # Find the fittest!
    fitness_array_sorted_idx = np.argsort(fitness_array)
    fittest = fitness_array_sorted_idx[0:num_survivals]

    iteration_error[it] = fitness_array[fittest[0]]
    reset = (it >= reset_it) and (sum(abs(iteration_error[it-reset_it:it] - iteration_
↳error[it])) == 0)
    if reset:
        genome = np.zeros((population_size, 9, 9))

        for k in range(population_size):
            # init candidate as problem

```

(continues on next page)

(continued from previous page)

```

genome[k, :, :] = np.copy(problem)

# fill in sudoku, by placing the digits where the problem is zero
for j in range(9):
    candidate_digits_j = missing_per_column[missing_per_column[:, j]>0,j]
    np.random.shuffle(candidate_digits_j)
    np.place(genome[k, :, j], genome[k, :, j] == 0, candidate_digits_j)

continue

sol = genome[fittest[0],:,:]

# print(sol)
# We might have found already a solution
if iteration_error[it] == 0:
    break
else:
    # Let us create the next generation
    # take ten survivals
    mutations = np.zeros((population_size,9,9))

    # randomly combine rows of the best solutions
    for k in range(population_size):
        # Init mutation_k as problem
        mutation_k = np.copy(problem)

        # we select the parent randomly from the survivals
        np.random.shuffle(fittest)
        parent_a = genome[fittest[0],:,:]

        # we first select columns to fill in randomly
        columns_to_fill_in_a = np.random.choice(np.arange(9), keep_cols,
        ↪replace=False)

        # we fill all the values selecting them from the parent
        mutation_k = parent_a

    # Now we create permutations of the columns that we want to mutate
    missing_column_k = np.setdiff1d(np.arange(9), columns_to_fill_in_a)
    for j in missing_column_k:
        # values missing in the original problem
        candidate_digits_j = missing_per_column[missing_per_column[:, j]>0,j]
        #if we are missing 0 or 1 value in a column, we will also copy it
        if len(candidate_digits_j) >= 2:
            # reorder the candidate digits
            np.random.shuffle(candidate_digits_j)
            # pick at most cells_to_shuffle digits
            digits_to_remove = candidate_digits_j[0:min(cells_to_shuffle,
            ↪len(candidate_digits_j))]
            # set them to 0
            np.place(mutation_k[:,j], np.isin(mutation_k[:,j], digits_to_

```

(continues on next page)

(continued from previous page)

```

↪remove), 0)

        # shuffle the digits
        np.random.shuffle(digits_to_remove)

        # set the values again
        np.place(mutation_k[:, j], mutation_k[:, j] == 0, digits_to_
↪remove)

        mutations[k, :, :] = np.copy(mutation_k)

        # copy mutation to genome
        genome = np.copy(mutations)

        # We increase the number of iterations
        it+=1

end_time = datetime.now()
duration = end_time - init_time
print("all righty, i finished the Sudoku in {0} seconds and {1} iterations: ".
↪format(duration.total_seconds(), it))
print("Best solution is:")
print(sol)
plt.plot(iteration_error)

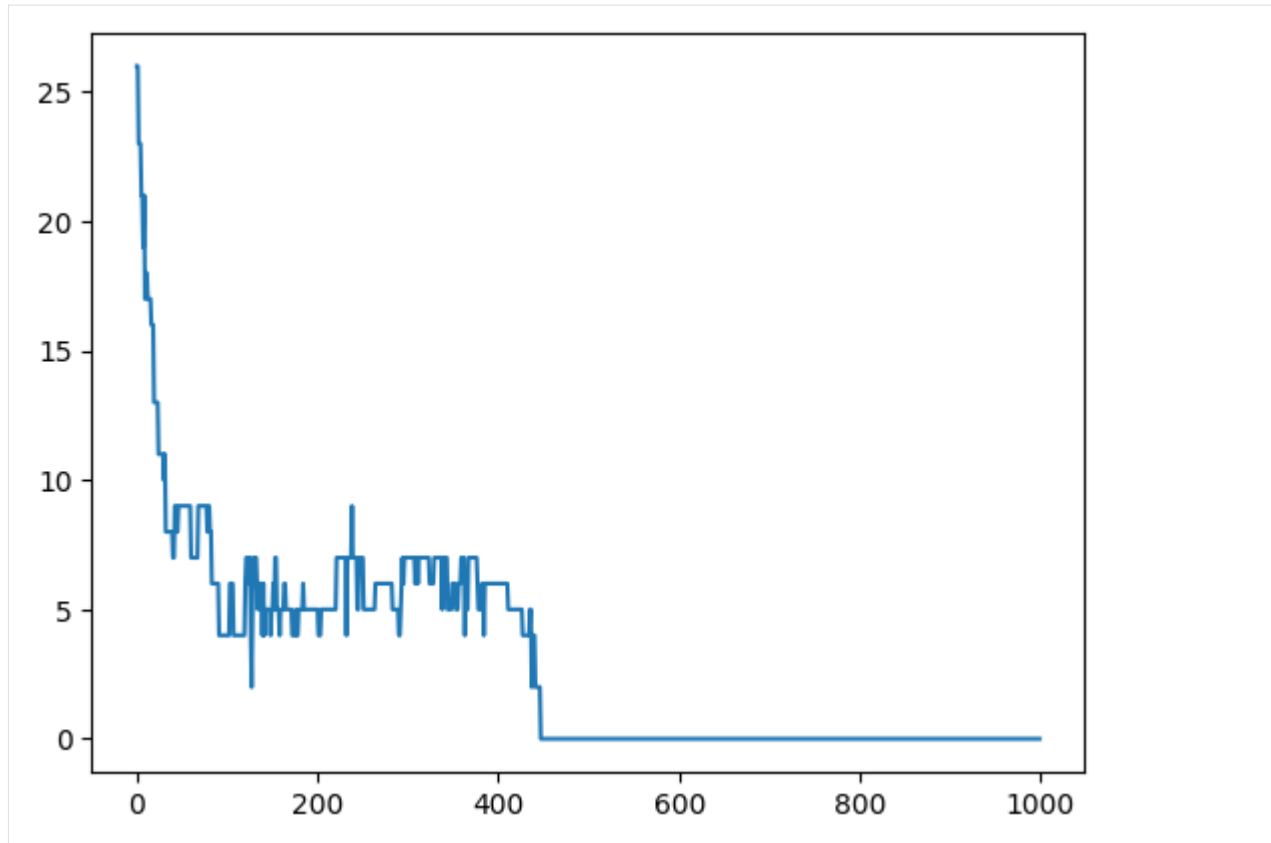
```

```

all righty, i finished the Sudoku in 7.524293 seconds and 447 iterations:
Best solution is:
[[8. 3. 6. 2. 7. 9. 4. 1. 5.]
 [9. 5. 1. 3. 4. 8. 7. 6. 2.]
 [4. 7. 2. 1. 5. 6. 3. 8. 9.]
 [7. 9. 3. 4. 8. 5. 6. 2. 1.]
 [1. 6. 5. 7. 9. 2. 8. 3. 4.]
 [2. 8. 4. 6. 1. 3. 9. 5. 7.]
 [5. 1. 8. 9. 3. 4. 2. 7. 6.]
 [3. 2. 9. 5. 6. 7. 1. 4. 8.]
 [6. 4. 7. 8. 2. 1. 5. 9. 3.]]

```

```
[4]: [<matplotlib.lines.Line2D at 0x2652ea5da80>]
```



### Analysis Questions

1. Run the algorithm above and write down the number of iterations and the execution time. Did the algorithm find an optimal solution? Do you think this approach is faster?
2. Ask an AI assistant to explain this algorithm to you in simple terms, and ask also if this algorithm is more efficient, and ask for simple explanations of why.
3. Look for the description of the size of the search space in the description above and ask again an AI assistant to provide some illustrative examples of how big this new search space is.
4. Ask an AI assistant again to describe what hyperparameters are and what are the hyperparameters of this algorithm. Ask if it can suggest a better set of hyperparameters to solve the problem faster. Try the suggested hyperparameters and compare the results. This process is called **hyperparameter tuning**, and it is a very important part of AI. Was the AI assistant able to suggest a better set of hyperparameters?
5. This strategy exploits the fact that columns are permutations of digits. What other strategies can you think of to solve the problem faster? How would you improve the algorithm? (Feel free to reason with your AI assistant about better ways to solve this problem)

### 3.1.5 Mixed integer Programming

#### Introduction

Mixed Integer Programming (MIP) can be regarded as an extension of Continuous Linear Programming where any of or all the defined decision variables cannot take fractional values. MIP introduces two new types of variables:

- **Integer variables:** Integer variables can only take Integer values. Normally, Integer variables are introduced by logical restrictions that make rational values unfeasible. For instance, in some instances of production mix problems, it does not make sense to produce fractional parts of a product and treating them as real number is just an approximation.
- **Binary variables:** Binary variables or Boolean variables can only take values 1 or 0. Normally, binary variables are used to factor into our model binary decisions, e.g. to decide whether to use a machine configuration or another, or whether to select one element from a set or not.

In CLP, the continuity of the decision variables ensures that the feasibility region and the objective function intersect in a vertex of the feasible region at the optimal value. But this is no longer the case when some decision variables are Integer values. Intuitively, one can think that the optimal value may be found in one of the feasible Integer values *around* the theoretical intersection, as long as the solution is feasible. This makes the number of candidate solutions larger and consequently, MIP problems are more difficult to solve. From a mathematical point of view, when the decision variables are not continuous, the feasible region and the objective function are no longer convex. Convexity of feasibility region and objective function ensures that the optimal solution is at one vertex. We will explain this concept further in Non-Linear Programming. For now, it is sufficient to understand that in MIP, the feasibility region is no longer convex since not all values are possible and this makes it harder to find a solution.

This limitation makes convergence to an optimal solution more complex. MIP solution is several orders of magnitude more complex than linear programming. For instance, the assumptions made to design the Simplex algorithm do no longer hold and although adaptations may find an optimal solution, convergence is so slow that they cannot be applied to practical problems.

#### Set up

In general, the set-up of MIP problems is the same as the set-up of CLP problems, except that some decision variables can only take Integer values. For instance:

#### decision variables

Let us note our decision variable as  $X = [x_1, x_2, \dots, x_n]$  where  $n$  is the number of decision variables. In continuous linear programming, the decision variables could take any real value, in mathematical notation this is expressed as:

$$x_j \in \mathbb{R}$$

Now, in **integer programming**, decision variables can only take integer numbers. Without loss of generalization, let us consider that our decision variables can only take positive integers, or natural numbers. We refer to this type of decision variables as **integer variables**, and are noted as:

$$x_j \in \mathbb{N}$$

A special type of integer variables are **binary variables** which are variables that can only take values 0, or 1. These variables are very convenient to model 'yes or no' decisions or 'true or false' (boolean) logic decisions into our problem. We note can use this notation for binary variables:

$$x_j \in [0, 1]$$

In general, not all decision variables have to be necessarily of the same type. Integer Programming (IP) problems only have integer or binary decision variables, whereas MIP problems can have any combination of decision variables.

### Objective Function

As with CLP, our objective function is a **linear** function of the decision variables, noted as  $z$  and which we would like to maximize or minimize. For instance, if our objective function is of type maximize, we can use:

$$\max z = \sum_{j=1}^n c_j \cdot x_j$$

### Constraints

The objective function will be subject to some limitations or constraints:

*s.t.*

$$\sum_{j=1}^n a_{1j} \cdot x_j \leq b_1$$

$$\sum_{j=1}^n a_{2j} \cdot x_j \leq b_2$$

...

$$\sum_{j=1}^n a_{mj} \cdot x_j \leq b_m$$

Let us now see some prominent examples of Integer Programming types. The next sections cover the formulation of some well known combinatorial optimization problems as IP problems.

### Knapsack Problem

The Knapsack problem is an optimization problem that is often found in practical applications. The classical formulation reads

Given a set of items, each with a weight and a value, we need to determine the number of each item to include in a collection so that the total value is maximised while the total weight is less than or equal than the maximum weight the knapsack can hold.

Just like packing a knapsack for a nice adventure!

Decision variables

The decision variables  $x_j$  are binary decision variables that determine whether we include item  $j$  in the Knapsack:

$$x_j \in [0, 1] \{0: \text{item not included}, 1: \text{item included}\} \forall j$$

That is, the decision variable can only take values 0, or 1. The value 0 means that the object is not selected and therefore has no impact in the objective function. Likewise, the value 1 means that the object is selected.

Objective coefficient

Each item  $j$  will have a different value  $v_j$ . We can express the objective function as the value of the items selected:

$$\max z = \sum_{j=1}^n v_j \cdot x_j$$



### Constraints

Now, our only constraint is related to the maximum weight  $W$  the knapsack can support. Each item will have a different weight  $w_j$ , and the sum of the weights  $w_j$  of the selected variables has to be less or equal than the maximum weight.

$$\sum_{j=1}^n w_j \cdot x_j \leq W$$

Note that, if a given  $x_j$  is zero, the product  $w_j \cdot x_j$  will also be zero and will have no impact on the total value of the left hand side summation.

Now, it is important to note that in this formulation, **weights** and **values** as abstract concepts, and the same formulation can be applied to many practical problems.

### Assignment Problems

Assignment problems are a fundamental class of combinatorial optimization problems. In essence, assignment problems consists of assigning elements of a set of resources  $R$  (like workers, machines, etc.) to a set of tasks  $T$  to optimize a certain objective (like minimizing cost or maximizing efficiency), subject to various constraints.

### Indices

Now, since we have two different sets, we will use two different **indices** to uniquely identify the elements in each set:

- **i**: We will use index  $i$  to identify each element in set  $R$ . Let us note the size of set  $R$  as  $m$ , therefore,  $i \in [1, 2, \dots, m]$
- **j**: We will use index  $j$  to identify each element in set  $T$ . The size of set  $R$  is noted as  $n$ , therefore,  $i \in [1, 2, \dots, n]$

### Decision Variables

Let us define a binary variable  $x_{ij}$  for each resource  $i$  and task  $j$ .  $x_{ij} = 1$  if resource  $i$  is assigned to task  $j$ , and 0 otherwise:

$$x_{ij} \in \{0, 1\} \text{ \{0: resource } i \text{ is not assigned to task } j, 1: \text{ otherwise} \} \forall i, \forall j$$

Note that there are a total of  $m * n$  decision variables!

### Objective function

The objective is to **minimize costs** or **maximize benefits** associated with assigning a particular resource to a particular task. As with Knapsack problems, we need to think of costs or benefits as **abstract concepts**, and the model can be applied to many practical applications.

For instance, let us note as  $c_{ij}$  the cost of assigning resource  $i$  to task  $j$ . The objective function then becomes:

$$\min z = \sum_{i=1}^m \sum_{j=1}^n c_{ij} * x_{ij}$$

### Constraints

Now, the constraints need to ensure that one element of R is assigned to one element T, that is, we need to ensure that there are not two or more resources assigned to a task:

$$\sum_{j=1}^n x_{ij} = 1 \quad \forall i$$

Note first that for a given  $i$ , the left hand side  $\sum_{j=1}^n x_{ij}$  is the sum of all the decision variables associated to a resource  $i$ . Since the different  $x_{ij}$  can only be 0 or 1, by establishing an upper bound limit of 1 to the sum of all the elements, we ensure that the resource can only be assigned at most to one task.

Similarly, we need to ensure that a task is only associated to one resource:

$$\sum_{i=1}^m x_{ij} = 1 \quad \forall j$$

Note that the expression is similar, but now the left hand side  $\sum_{i=1}^m x_{ij}$  represents the sum of all the decision variables associated to task  $j$

### Sequencing Problems

#### Traveling Salesman Problem

Recall the classic formulation of the Travelling Salesman Problem (TSP):

Given a list of *cities* and the *distances* between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city

In practical applications, *cities* represent different steps in a process, and *distances* represent the cost of a sequence of steps in the process, from one step to another.

Thus, for the setup of the TSP as an IP, we have the following variables and coefficients:

- $x_{ij}$ : Binary decision variables that represent a sequence from source  $i$  to destination  $j$ . 1 means that the step is selected in the final sequence, 0 otherwise.
- $c_{ij}$ : Represents the cost the step in the sequence, in the classical formulation, the cost of travelling from city  $i$  to city  $j$ .
- $u_i$ : Integer variable that represents the step of  $i$  in the sequence. If  $u_i$  is equal to  $t$  then city  $i$  is departed at step  $t$ .

The IP formulation is then as follows:

$$\min z = \sum_{i=1}^N \sum_{j=1}^N c_{ij} * x_{ij}$$

*s.t* We can only depart from 1 city:

$$\sum_{j=1}^N x_{ij} = 1 \quad \forall i$$

We can only arrive to a city from a city:

$$\sum_{i=1}^N x_{ij} = 1 \quad \forall j$$

We cannot travel in loops in a city:

$$x_{ij} = 0 \quad \forall i = j$$

We travel from city  $i$  to city  $j$  exactly at step  $u_i$ :

$$u_i - u_j + n * x_{ij} = n - 1 \quad \forall i \in 2 \leq i \neq j \leq n$$

There are no loops, we do not allow  $u_i$  to be arbitrarily large:

$$0 \leq u_i \leq n - 1 \quad \forall i \in 2 \leq i \leq n$$

## Algorithms

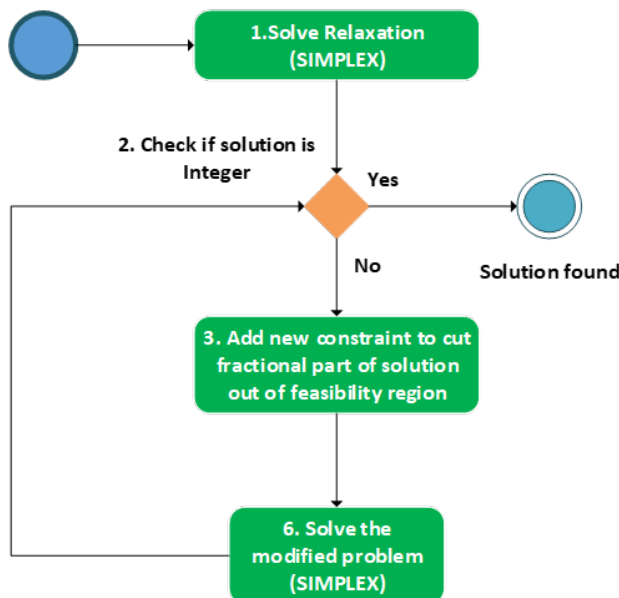
There are different techniques used to solve MIP problems, among which we can highlight the following:

### Relaxation

Relaxation is basically formulating the MIP problem as if all decision variables were continuous, and therefore reducing the MIP problem to a CLP problem. Relaxation allows the decision maker to use the Simplex method and perform a sensitivity analysis. The Solution may be not feasible, but it might provide some insights into the possible solution.

### Cutting Planes (Gomory's Algorithm)

The Gomory's algorithm is an extension of relaxation. First, the problem is formulated as a CLP. When the solution is found, new constraints are introduced to *cut* the fractional part of one of the Integer variables out of the feasibility region. This yields a new CLP that can be solved with the Simplex method. This process is repeated until a solution is found where all Integer variables actually take Integer values.



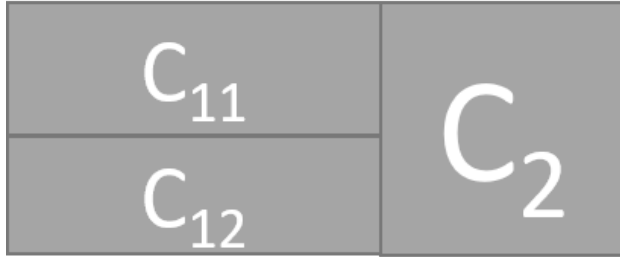
### Search (Branch and bound)

Branch and bound is one of the most common methods to solve combinatorial optimization and MIP problems. This type of algorithms explore the possible solutions iteratively, by dividing the set of possible solutions into smaller sets, then finding a *bound* of the objective function in each subset and selecting the subset, or branch with the optimal bound.

Let us see the set-up for a minimization problem. The objective function is:

$$\min z = f(x)$$

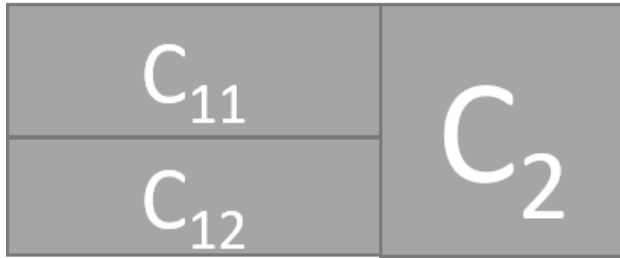
where  $x$  is the decision variable which can take values from a finite set  $C$ , i.e.  $x \in C$ .



Let us divide the set  $C$  into two subsets  $C_1, C_2$  and find a minimum bound of the objective function in each subset such that:

$$f(x) \leq c_1 \forall x \in C_1$$

$$f(x) \leq c_2 \forall x \in C_2$$

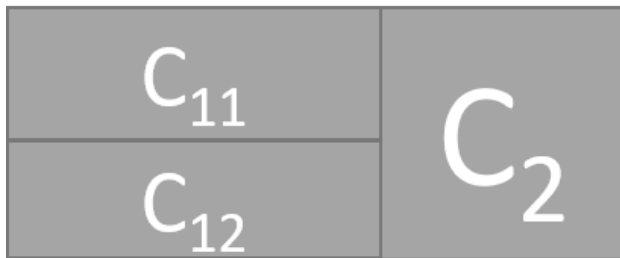


Now, let us assume that  $c_1 \leq c_2$ . Since we are looking for the minimum value of  $z$ , we can discard the entire set  $C_2$  from the search and continue looking for the solution in branch  $C_1$ .

Now again, let us divide  $C_1$  into two subsets  $C_{11}, C_{12}$ . We can find again a minimum bound for  $f(x)$  in each subset:

$$f(x) \leq c_{11} \forall x \in C_{11}$$

$$f(x) \leq c_{12} \forall x \in C_{12}$$



Again, we can compare bounds and select the branch with the minimum bound. Note that, at each iteration, we are selecting a better suboptimal solution. Let us note as  $\alpha$  the index of the subset of  $C$  at a given iteration. If  $C_\alpha$  contains only 1 element  $\alpha$  and  $c_\alpha = f(\alpha)$  is lower than the minimum bound of the other subsets, then  $\alpha$  is the optimal solution.

## Genetic Algorithms

Genetic algorithms can also be used to solve MIP problems. You can find a description [here](#)

### 3.1.6 Mixed Integer Programming with PuLP

Try me



#### Introduction

This tutorial shows how to program and solve Mixed Integer Programming (MIP) problems in Python using the PuLP library.

This tutorial builds upon the tutorial on PuLP we saw in the previous unit:

*Continuous Linear Programming with PuLP*

#### Requirements

To run this tutorial you will need to install PuLP, following the guides in the previous tutorial. For instance, if you are using this tutorial in Colabs, run the following cell to install PuLP:

```
[ ]: # Install PuLP
!pip install pulp
```

#### Example

We will use the *Allocating order to machines* problem to guide the tutorial. For convenience, this is the mathematical formulation:

#### indices

- **i**: index to identify each machine (resource element)  $i \in \{1, 2, 3, 4, 5\}$
- **j**: index to identify each production order (task element)  $j \in \{1, 2, 3, 4, 5\}$

#### Coefficients

$$C = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mn} \end{bmatrix} = \begin{bmatrix} 16 & 2 & 8 & 3 & 3 \\ 4 & 14 & 10 & 7 & 6 \\ 9 & 7 & 3 & 6 & 8 \\ 5 & 5 & 12 & 10 & 11 \\ 6 & 13 & 11 & 5 & 7 \end{bmatrix}$$

### Objective function

$$\min z = \sum_i \sum_j x_{ij} * c_{ij}$$

### Constraints

$$\sum_j x_{ij} \leq 1 \quad \forall i$$

$$\sum_i x_{ij} \leq 1 \quad \forall j$$

### Solution in Python

The following script models and solves the problem using PuLP:

```
[2]: # Import PuLP modeler functions
import pulp

# Create the model
model = pulp.LpProblem(name="Allocating orders to machines", sense=pulp.LpMinimize)

# Define indices
machines = [1, 2, 3, 4, 5]
orders = [1, 2, 3, 4, 5]

# Define decision variables
x = pulp.LpVariable.dicts("x",
                          [(i,j) for i in machines for j in orders],
                          cat=pulp.LpBinary,
                          )

# Define coefficient matrix
C = [[16, 2, 8, 3, 3],
     [4, 14, 10, 7, 6],
     [9, 7, 3, 6, 8],
     [5, 5, 12, 10, 11],
     [6, 13, 11, 5, 7]]

# Define objective function
model += pulp.lpSum([C[i][j]*x[(i+1,j+1)] for i in range(len(machines)) for j in_
↪range(len(orders))])

# Define constraints
for i in machines:
    model += pulp.lpSum([x[(i,j)] for j in orders]) == 1

for j in orders:
    model += pulp.lpSum([x[(i,j)] for i in machines]) == 1

# Solve the problem
model.solve()

# Print the status of the solution
print("Status:", pulp.LpStatus[model.status])

# Print the value of the objective function
print("Objective value:", pulp.value(model.objective))

# Print the value of the decision variables
```

(continues on next page)

(continued from previous page)

```
for v in model.variables():
    print(v.name, "=", v.varValue)
```

```
Status: Optimal
Objective value: 20.0
x_(1,_1) = 0.0
x_(1,_2) = 0.0
x_(1,_3) = 0.0
x_(1,_4) = 0.0
x_(1,_5) = 1.0
x_(2,_1) = 1.0
x_(2,_2) = 0.0
x_(2,_3) = 0.0
x_(2,_4) = 0.0
x_(2,_5) = 0.0
x_(3,_1) = 0.0
x_(3,_2) = 0.0
x_(3,_3) = 1.0
x_(3,_4) = 0.0
x_(3,_5) = 0.0
x_(4,_1) = 0.0
x_(4,_2) = 1.0
x_(4,_3) = 0.0
x_(4,_4) = 0.0
x_(4,_5) = 0.0
x_(5,_1) = 0.0
x_(5,_2) = 0.0
x_(5,_3) = 0.0
x_(5,_4) = 1.0
x_(5,_5) = 0.0
```

```
C:\Users\franc\PycharmProjects\operations-research-notebooks\venv\Lib\site-packages\
↳pulp\pulp.py:1316: UserWarning: Spaces are not permitted in the name. Converted to
↳ '_'
warnings.warn("Spaces are not permitted in the name. Converted to '_'")
```

### Code Explanation

The code is very similar to the efficient code we used in the previous tutorial. The only difference is in the way we define the decision variables. Let's see the differences:

#### Decision variables

We use the same function `pulp.LpVariable.dicts` to define the decision variables. There are two main differences: - **Definition of integer or binary decision variables:** We use the parameter `cat=pulp.LpBinary` to define the decision variables as binary. If we wanted to define them as integer, we would use `cat=pulp.LpInteger`. - **Definition of the indices:** We use a list comprehension to define the indices of the decision variables. The list comprehension is a very powerful tool in Python that allows us to define lists in a very compact way. The list comprehension we use is the following:

```
[(i,j) for i in machines for j in orders]
```

This list comprehension defines a list of tuples, where each tuple is defined as  $(i, j)$ , where  $i$  takes the values in the list `machines` and  $j$  takes the values in the list `orders`. This way we define all the possible combinations of  $i$  and  $j$  in a very compact way.

Thus, the code:

```
x = pulp.LpVariable.dicts("x",
                        [(i,j) for i in machines for j in orders],
                        cat=pulp.LpBinary,
                        )
```

Defines a total of  $5 \times 5 = 25$  decision variables, one for each combination of  $i$  and  $j$ . The decision variables can be accessed using a tuple. For instance,  $x[(1, 1)]$  is the decision variable associated with the combination  $i=1$  and  $j=1$ .

### Reading data from an Excel file

In the previous tutorial, we defined the coefficient matrix in the code. However, in real problems, the data is normally stored in a file. In this case, we will use an Excel file to store the data. We will see how to read the data from the Excel file and store it in a `DataFrame`. Then, we will see how to access the data in the `DataFrame` to define the coefficient matrix. The following code shows how to load the data from an Excel file named `costs.xlsx`:

```
[ ]: import pandas as pd
import pulp
from IPython.display import display

# import the data from the Excel file
dataset_df = pd.read_excel("costs.xlsx", index_col=[0,1])
c = dataset_df['cost']

# Pivot the data to show it in a more convenient way
dataset_df = dataset_df.reset_index()
dataset_df = dataset_df.pivot(index='machine', columns='order', values='cost')
display(dataset_df)

# Create the model
model = pulp.LpProblem(name="Allocating orders to machines", sense=pulp.LpMinimize)

# Define indices
machines = [1, 2, 3, 4, 5]
orders = [1, 2, 3, 4, 5]

# Define decision variables
x = pulp.LpVariable.dicts("x",
                        [(i,j) for i in machines for j in orders],
                        cat=pulp.LpBinary,
                        )

# Define objective function
model += pulp.lpSum([c[i][j]*x[(i,j)] for i in machines for j in orders])

# Define constraints
for i in machines:
    model += pulp.lpSum([x[(i,j)] for j in orders]) == 1

for j in orders:
    model += pulp.lpSum([x[(i,j)] for i in machines]) == 1

# Solve the problem
model.solve()
```

(continues on next page)



(continued from previous page)

```

# Print the status of the solution
print("Status:", pulp.LpStatus[model.status])

# Create a dataframe to store the solutions
x_df = pd.DataFrame.from_dict(x, orient="index", dtype=object, columns=["variable_
↳object"])

# Extract the values of the decision variables from the solution
x_df["solution_value"] = x_df["variable_object"].apply(lambda item: item.varValue)

# Drop the pulp decision variables
x_df = x_df.drop(columns=["variable_object"])

# Pivot the dataframe to show the solution in a more convenient way
x_df = x_df.reset_index()

x_df['machine'] = x_df['index'].apply(lambda i: i[0])
x_df['order'] = x_df['index'].apply(lambda i: i[1])

# Drop the index column
x_df = x_df.drop(columns=["index"])

# Reorder the columns to show the solution in a more intuitive way
x_df = x_df[["machine", "order", "solution_value"]]

# Pivot the dataframe to show the solution in a more convenient way
x_df = x_df.pivot(index='machine', columns='order', values='solution_value')
display(x_df)

# Save to an Excel file
with pd.ExcelWriter('solution.xlsx') as writer:
    x_df.to_excel(writer, sheet_name="X")

```

The following sections explain the code in detail.

## Code Explanation

### Preparing the dataset

Normally, the data for our problem is going to be stored in tabular format in file formats like Excel. In this case, we can use the pandas library to read the data from the Excel file and store it in a DataFrame. Then, we can use the DataFrame to access the data in a very convenient way. First, just copy the following table in an Excel file:

machine	order	cost
1	1	16
1	2	2
1	3	8
1	4	3
1	5	3
2	1	4
2	2	14
2	3	10
2	4	7
2	5	6

(continues on next page)

(continued from previous page)

3	1	9
3	2	7
3	3	3
3	4	6
3	5	8
4	1	5
4	2	5
4	3	12
4	4	10
4	5	11
5	1	6
5	2	13
5	3	11
5	4	5
5	5	7

Make sure you paste the table in the first sheet of the Excel file as **text**. It should look like this:

	A	B	C	D	E	F
1	machine	order	cost			
2	1	1	16			
3	1	2	2			
4	1	3	8			
5	1	4	3			
6	1	5	3			
7	2	1	4			
8	2	2	14			
9	2	3	10			
10	2	4	7			
11	2	5	6			
12	3	1	9			
13	3	2	7			
14	3	3	3			
15	3	4	6			
16	3	5	8			
17	4	1	5			
18	4	2	5			
19	4	3	12			
20	4	4	10			

Important! If you notice that all the text of a row is in the same column, try to Paste As text from the Edit menu in Excel.

Name the file `costs.xlsx`.

Uploading the dataset to your Python environment

Now, you need to make sure that the Excel file is in the working directory of your Python environment. If you are using Colabs, you can use the file menu to upload it as shown in this tutorial:

[Working with files](#)

Loading the dataset in a DataFrame

Now, we can use the `pandas` library to load the data in a `DataFrame`:

```
import pandas as pd
dataset_df = pd.read_excel("costs.xlsx", index_col=[0,1])
c = dataset_df['cost']
```

The `index_col` parameter indicates that the first two columns of the Excel file are the indices of the `DataFrame`.

Accessing the data in the DataFrame

Note that we have stored the costs in a variable named `c`. This is because we will use the `DataFrame` to access the data in a very convenient way. For instance, if we want to access the cost of machine 1 and order 1, we can use the following code:

```
print(c[1,1])
```

This will print the value of the cell in the first row and first column of the `DataFrame`, corresponding to machine 1 and production order 1.

Pivoting the DataFrame to show the data in a more convenient way

To show the data in a more convenient way, we can pivot the `DataFrame`:

```
dataset_df = dataset_df.reset_index()
dataset_df = dataset_df.pivot(index='machine', columns='order', values='cost')
display(dataset_df)
```

This will show the data in a very convenient way, environments like Colabs provide interactive widgets to further work with the data.

Defining the objective function

Now, we can modify the objective function slightly to access the data in the `DataFrame`:

```
model += pulp.lpSum([c[i][j]*x[(i,j)] for i in machines for j in orders])
```

Note that since the indices of `c` are the same as the indices of the decision variables, the code is very compact and close to mathematical notation.

Exporting to Excel

Finally, we can put the variables in a dataframe:

```
# Create a dataframe to store the solutions
x_df = pd.DataFrame.from_dict(x, orient="index", dtype=object, columns=["variable_
↪object"])

# Extract the values of the decision variables from the solution
x_df["solution_value"] = x_df["variable_object"].apply(lambda item: item.varValue)
```

This code first creates a dataframe with the decision variables and then extracts the values of the decision variables using a lambda function. Pandas will apply the lambda function to each row of the dataframe, passing the decision variable object as a parameter. The lambda function will return the value of the decision variable.

Finally, we can pivot the dataframe and export it to excel using the following code:

```
with pd.ExcelWriter('solution.xlsx') as writer:
    x_df.to_excel(writer, sheet_name="X")
```

This code will create an Excel file named `solution.xlsx` with a sheet named `X` containing the solution. If our problem has more decision variables, we can create more sheets and store the solutions in different sheets.

### Analysis Questions

1. Analyse the solution and explain the assignment of orders to machines that the model has found in your own words.
2. How would you modify the model to allow for the assignment of more than one order to each machine?
3. Use an AI assistant to obtain a Python script to solve the problem using PuLP. Compare the script with the one we have developed in this tutorial. What are the main differences?
4. Ask an AI assistant to provide a greedy algorithm to solve the problem in Python. Try the algorithm. Does the algorithm provide an optimal solution? Why?
5. Try to solve different instances of the problem, for instance adding more machines or more production orders. What changes are needed in the script. How can you make the script more scalable?

## 3.1.7 Transportation Problems

### Basic Set up

In the Introduction to MIP, we described the wide range of problems that can be modeled and solved with MIP, and described the setup of some widely used problem models. In this notebook, we introduce the transportation problem, as a special type of MIP problem particularly important for logistic operations. In the literature, the general transportation problem deals with the distribution of any commodity (generally goods) from a group of origins of the supply chain, or source facilities, called **sources** to the endpoints of the model of the supply chain, the receiving centers, called destinations.

There are different types of goods or services that can be modeled with this model. Sources can be production centers, or warehouses. Destinations can be warehouse, demand regions or end customers, depending on the model.

However, in the general problem formulation, we have  $i \in [1, \dots, n]$  source nodes and  $j \in [1, \dots, m]$  destination nodes. The objective is to minimize the overall costs of the distribution of goods at a minimum costs, subject to a set of constraints.

Regarding the constraints, let us consider two types of constraints:

- **supply constraints:** These constraints represent the limited capacity of the sources. For instance, production centers may have a limited production capacity, or source warehouses may have a limited storage capacity. Let us note  $s_i$  the source capacity of source node  $i$ .
- **demand constraints:** The demand constraints represent the distribution requirements at the destinations: what are the quantities of the goods required at each destination. Let us note as  $d_j$  the demand of destination node  $j$

Now, based on this, let us define our set of decision variables as:

- $x_{ij}$ : (Integer) units to transport from source  $i$  to destination  $j$

Assuming a distribution cost of  $c_{ij}$  for each transported unit yields:

$$\min z = \sum_{i=1}^n \sum_{j=1}^m c_{ij} * x_{ij}$$

s.t

$$\sum_{j=1}^m x_{ij} \leq s_i \quad \forall i$$

$$\sum_{i=1}^n x_{ij} \geq d_j \quad \forall j$$

That is, the total number of units that leave a source cannot exceed the capacity of the source, and the total number of units that reach a destination must be at least equal to the demand.

Note that the model has a feasible solution only if:

$$\sum_{i=1}^n s_i \geq \sum_{j=1}^m d_j$$

That, is, the problem is only feasible if the overall capacity is sufficient to meet the overall demand.

It is important to note that we may allow ourselves less slack by making any of the three inequalities above equalities.

## Sourcing design

In the basic set up above, it is assumed that it is possible to transport units from *any* source node to *any* destination. This implies that the design of the distribution network is known. But this may not be always the case. For instance, we may have different possible locations for source warehouse, each with different associated costs. In this case, our decision variables are not only the number of units to transport, but also determine from which subset of source nodes they must be transported. Thus, we can introduce a new set of decision variables:

- $Y_i$ : (Binary) determines whether units to any destination may be delivered from source node  $i$

We need to introduce new logical constraints to ensure that we do not deliver any units from source nodes that are not selected:

$$\sum_{j=1}^m x_{ij} \leq M * Y_i \quad \forall i$$

Where M is a large number. Since  $Y_i$  is binary, this constraint forces that, in case that it is equal to zero, no units are sourced from source node  $i$ .

This logical constraint can be merged with the source capacity constraint as:

$$\sum_{j=1}^m x_{ij} \leq s_i * Y_i \quad \forall i$$

This constraint takes into account the logical constraint and the capacity constraint. Since both constraint are equivalent, but the former is more restrictive and renders the first one irrelevant.

We can now take into account in the objective function fixed costs that do not depend on the number of units. For instance, let us consider that the source warehouses have a fixed operation costs  $f_i$ . The problem in the basic set up becomes:

$$\min z = \sum_{i=1}^n \sum_{j=1}^m c_{ij} * x_{ij} + \sum_{i=1}^n f_i * Y_i$$

s.t

$$\sum_{j=1}^m x_{ij} \leq s_i * Y_i \quad \forall i$$

$$\sum_{i=1}^n x_{ij} \geq d_j \quad \forall j$$

### Single source

The single source constraint imposes that all the demand of a destination node is transported from the same origin. With this constraint, now the decision variables  $x_{ij}$  become binary:

- $X_{ij}$ : (Binary) determines if source  $i$  delivers all demand to destination  $j$  { 1 if yes, 0 otherwise }

The demand constraint now becomes:

$$\sum_{i=1}^n d_j * X_{ij} = d_j \quad \forall j$$

Note that we now multiply each binary decision variable times the demand of each destination. We have changed the type of the constraint from  $\geq$  to  $=$  to avoid any slack. Although this change is optional, if not applied, the slack will be a multiple of the demand. Note that this is exactly the same requirement:

$$\sum_{i=1}^n X_{ij} = 1 \quad \forall j$$

We just divided by  $d_j$  the right hand side and the left hand side.

Also, we need to factor this into the objective function as:

$$\min z = \sum_{i=1}^n \sum_{j=1}^m c_{ij} * d_j * X_{ij}$$

Since  $X_{ij}$  is binary, and only equal to 1 for a destination node.

### Network design

In the set up above, we assume that the cost per unit from sources to destinations is fixed. In many cases however, this cost depends to a great extent on the design considerations taken in the distribution network between sources and destination. Similar to what we did with the sourcing design, we may be interested on evaluating the impact in the cost of the design of the distribution network, for instance, the location of intermediate nodes or *junctions* between the source nodes and destination nodes. In literature, this is known as the *transshipment problem* in the literature. Let us come back to the original problem, and assume that the source nodes are production plants, the destination nodes are retailer regions, and that we have intermediate warehouses that we can use to optimise costs. Let us assume that the production plants are fixed (i.e. the sourcing design is known), but that we can select a subset of warehouses from a set of candidate locations.

First, let us modify the indices so that the distribution stages (from production plants through warehouses to regions) follow an alphabetical order:

- $i$ : Production plants  $i \in [1, \dots, n]$
- $j$ : Possible warehouse locations  $j \in [1, \dots, m]$
- $k$ : Retailer regions  $k \in [1, \dots, l]$

Let us use the following notation for the coefficients of our problem:

- $d_k$ : yearly demand in retail region  $k$
- $a_{ij}$ : cost of transporting 1 unit from plant  $i$  to warehouse  $j$
- $b_{jk}$ : cost of transporting 1 unit from warehouse  $j$  to retailer region  $k$
- $F_j$ : yearly operation costs of warehouse  $j$
- $c_i$ : yearly production capacity of plant  $i$

Since we can decide the location of the warehouses, and the units to transport at each stage, the decision variables now become:

- $Y_j$ : (Binary) { 1 if a warehouse is placed in location  $j$  } { 0 otherwise }
- $s_{ij}$ : integer units transported from plant  $i$  to warehouse  $j$

- $t_{jk}$ : integer units transported from warehouse  $j$  to region  $k$

The objective function is to minimise costs:

$$\min z = \sum_{i=1}^n \sum_{j=1}^m a_{ij} * s_{ij} + \sum_{j=1}^m \sum_{k=1}^l b_{jk} * t_{jk} + \sum_{j=1}^m F_j * Y_j$$

Now, for the constraints, clearly, we need to satisfy all the constraints. The units sourced from each production plant cannot exceed the production capacity:

$$\sum_{j=1}^m s_{ij} \leq c_i \quad \forall i$$

We also need to ensure that the demand at each region is satisfied:

$$\sum_{j=1}^m t_{jk} \geq d_k \quad \forall k$$

Now, we need to ensure the material flow, that is, that the unit that leave a warehouse are not less than the units that enter the same warehouse:

$$\sum_{i=1}^n s_{ij} \geq \sum_{k=1}^l t_{jk} \quad \forall j$$

And that a warehouse is built if it supplies to any region:

$$\sum_{k=1}^l t_{jk} \leq M * Y_j \quad \forall j$$

## Known variations

### Combination of single-source in design constraints

Any combinations of the variations above can be found in a problem instance. For instance, we might be interested in evaluating our network design with a single source constraint on intermediate nodes. Note that when we introduce a single source constraint in our problem, the decision variable that models the transport of units becomes binary. For instance, in the network design problem above, the decision variables Any combinations of the variations above can be found in a problem instance. For instance, we might be interested in evaluating our network design with a single source constraint on intermediate nodes. Note that when we introduce a single source constraint in our problem, the decision variable that models the transport of units becomes binary. For instance, in the network design problem above, the decision variables  $t_{jk}$  become:

- $T_{jk}$ : (Binary) {1 units from warehouse  $j$  transported to region  $k$ , 0 otherwise}

Since the variable are now binary, the single source constraint modifies the demand constraint, which now becomes:

$$\sum_{j=1}^n T_{jk} = 1 \quad \forall k$$

Now, since all  $T_{jk}$  are binary, the logical constraint that rules the construction of warehouses can be written as:

$$T_{jk} \leq Y_j \quad \forall j, \forall k$$

### Other indices

We may also introduce other indices in the problem, most common indices are:

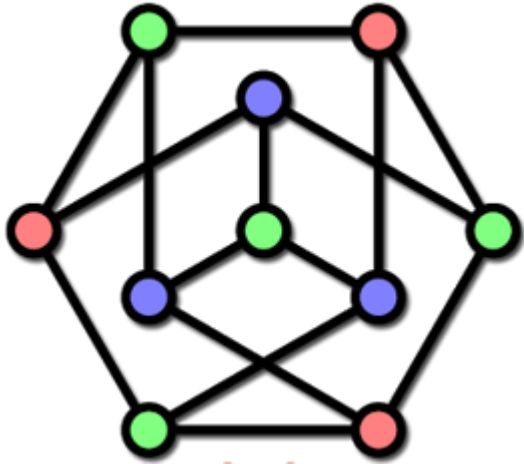
- **Type of product:** We can have different costs or demands depending on the type of product
- **Periods:** The costs, or the demands can be dynamic and depend on a discrete time period, like the year, the month, or the day, so we might be interested in optimising the distribution network for a set of periods.

When we introduce a new index into the model, the size of the problem (number of coefficients and decision variables) will grow accordingly.

### 3.1.8 Graph Theory

#### Introduction

Graph theory is a field of mathematics based on mathematical structures called graphs. A graph is just a diagram made of a set of nodes and a set of edges that join two nodes. The following image illustrates a graph.



Graphs are abstract objects that humans use to model and design many structures. We can easily relate graphs with roads and other types of distribution networks, like data networks, or pipelines. We use graphs to illustrate the organisation of personnel or the hierarchy of the departments in a company, or the layout of a warehouse or of the production floor plant, so in this sense, graph theory can be applied in many fields.

Formally, a graph is a pair of sets  $G = (N, E)$ , where  $E$  notes the set of edges and  $V$  the set of vertices, which have the following definitions:

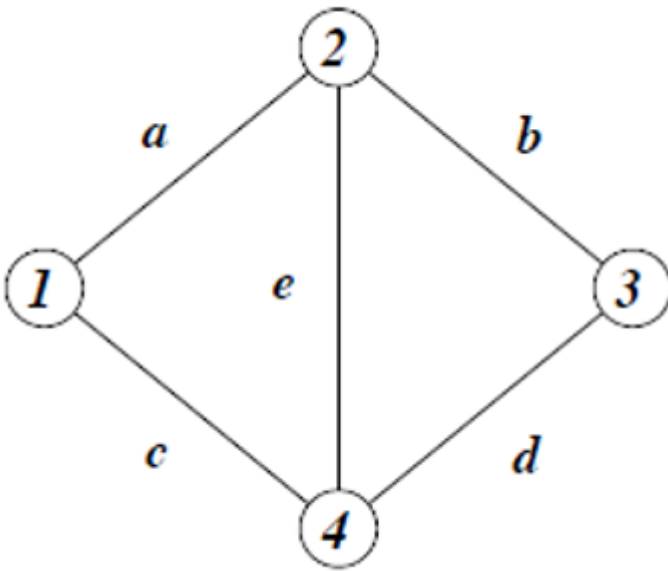
- **Nodes:** The nodes or vertices of the graph. Normally, it is a finite set and the **order** of a graph is the size of set, normally noted as  $N$ . It can also be infinite, as long as it is *numerable*, and we can identify and enumerate the nodes. Nodes can have different properties. For instance, they can have geographical coordinates, or can be grouped into different groups.
- **Edges:** Edges or arcs connecting two different nodes of the graph. Nodes are noted according to the indices of the nodes they connect. For instance, an edge connecting nodes  $i$  and  $j$  is noted as  $e_{ij}$ . Nodes  $i$  and  $j$  are called end vertices of  $e_{ij}$  and they are said to be **adjacent**. If  $P$  is the size of  $E$ , we refer to the graph as a  $P$ -Graph. For instance, a graph with 10 edges or arcs is referred to as a 10-Graph.

Another important definition is a **path**. A path is a set of edges that interconnect different nodes, traversing the graph possibly through other intermediate nodes.

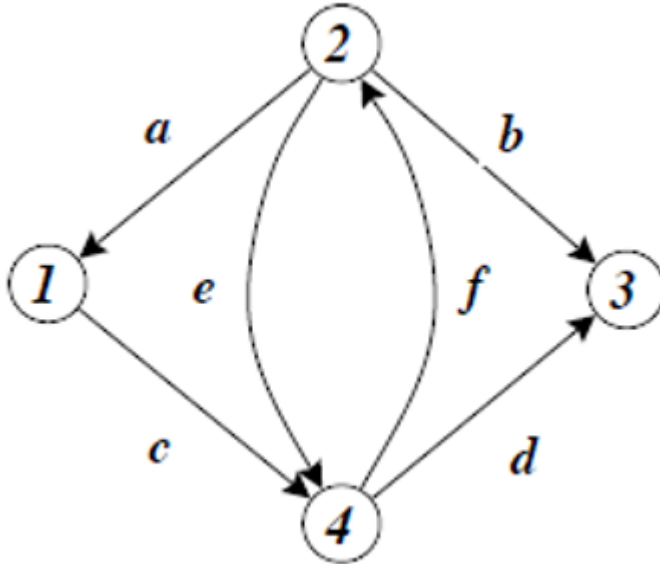
#### Types of graphs

Let us introduce the following definitions to classify graphs: - **Weighted graph:** A weighted graph is a graph where each edge  $e_{ij}$  has a different *weight*  $w_{ij}$ . The weight represents a real world property associated with the graph, and that represents the cost associated to the edge. For instance, the weight can represent the distance of an edge in a graph that represents a distribution network. Edges in a weighted graph can have other attributes, like the *capacity*  $b_{ij}$ . These attributes will be taken into account when selecting the edges to solve a specific optimization problem.

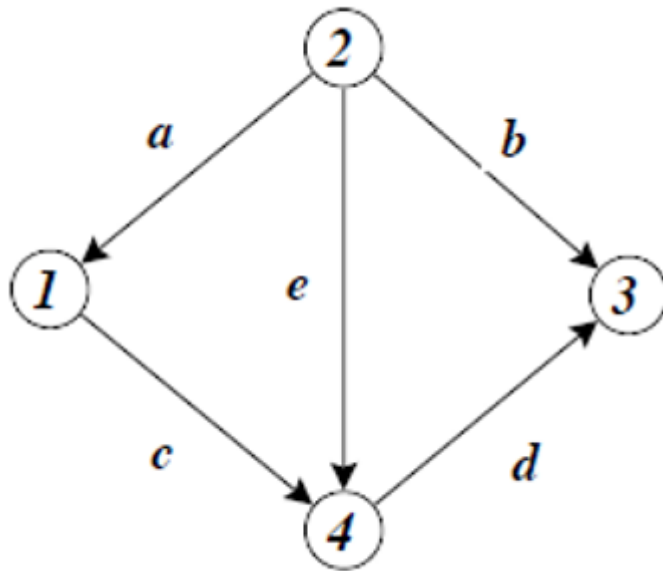




- **Directed graph:** In a directed graph,  $E$  is a set of ordered pairs of distinct vertices. Normally, a directed graph is also weighted and the weights of the edges connecting to nodes are not symmetric, that is  $w_{ij}$  not equal to  $w_{ji}$ .



- **Oriented graph:** An oriented graph is a directed graph with at most one edge interconnecting any two nodes  $i, j$ . That is, if there is an edge  $e_{ij}$ , there cannot be an edge  $e_{ji}$ .



## Types of problem

### Shorter path

Given a directed graph with positive edge weights (e.g. distance, cost), the shortest path problem aims to find a path between two given nodes that minimizes the total weight of the edges in the path, which is the sum of the weights of the edges in the path.

This problem is important in routing problems, where we need to determine the shortest path between two nodes, when there might be more than one possible alternative. It is therefore a partial problem in more complex problems in logistic applications, for instance if we want to establish the best route that minimises the total distance covered by a fleet. It can also be used to model other situations like minimising the total cost of a sequence of activities (replacing equipment).

### Minimum spanning tree

Given a graph with edge weights (e.g. distance, cost), the minimum spanning tree problem aims to find a *spanning tree* that minimizes the total weight (i.e. the sum of the weights of its edges).

A spanning tree is a path that interconnects all nodes starting from a root node.

This problem is typically applied to designing telecommunication networks with a minimum total cost, although it can also be used to design transport networks, cable TV, distributed systems, to interpret climatological data, among other applications.

### Maximal Flow problems

Given a directed graph with edge capacities (e.g. volume/second), maximal flow problems aim to find a *flow* from a given source node to a given sink node that maximizes the total flow.

The total flow represents the maximum capacity available in the path connecting the source node and the sink node.

Maximal flow problems are applied in logistics and distribution problems, but also in other types of applications like the commercialisation of products in a production–distribution network, programming employment, etc.

### Minimal cost flow problems

Given a graph with edge weight (e.g. cost) and demands for each node, find a flow satisfying all demands that minimizes the total cost.

The minimal cost flow problem can be used to model transportation problems using graphs.

## 3.1.9 Graph Theory in Python: Routing example

Try me



### Description

In this notebook, you will learn how to solve network problems using Python. For the set-up, we will use a map of the city of Valencia, and calculate the shortest distance between two points using the actual street map of the city. We will use the following libraries:

- **Networkx**: This Python library implements a lot of network theory utils and algorithms and is the main library used in the problems.
- **Osmnx**: This library allows us to create a graph from an Open Street Maps query (similar to what you would use to find a place in Google Maps).

Additionally, we will use the following libraries to render the maps:

- **Folium**: This library allows us to render the maps in our Notebooks

Remember you must **Trust** the notebook to test it. In Colabs, Run the following cell to install the libraries:

```
[ ]: !pip install networkx[default,extra]
!apt-get -qq install -y libspatialindex-dev
!pip install osmnx
!pip install folium
!pip install python-igraph
!pip install mapclassify
```

### Network creation

First we need to create the map:

```
[ ]: import networkx as nx
import osmnx as ox
from IPython.display import IFrame
%matplotlib inline

# Get the map of Valencia
G_nx = ox.graph_from_place('Valencia, VC, Spain', network_type='drive')

print("The streets of Valencia can be modeled with a graph of size: " + str(G_nx.
↪size()))
```

### Code Explanation

We have created a graph that represents the streets of Valencia, Spain, using the function `ox.graph_from_place`. The function takes as input a place name, and returns a graph of the streets in that place. The parameter `network_type` specifies the type of streets that we want to include in the graph. In this case, we are only interested in streets that can be used by cars. Other possible values are `walk` (pedestrian streets), `bike` (streets that can be used by bikes), `drive_service` (service roads), `all` (all the streets in the place).

Now, the object `G_nx` is a graph of the city, we do some adaptations that will allow us to work with the map more easily:

```
[ ]: # convert osmids IDs to a list
osmids = list(G_nx.nodes)

# Relabel the nodes with integers. This facilitates handling the tree
G_nx = nx.relabel.convert_node_labels_to_integers(G_nx)

# give each node its original osmid as attribute, since we relabeled them
osmid_values = {k:v for k, v in zip(G_nx.nodes, osmids)}
nx.set_node_attributes(G_nx, osmid_values, 'osmid')
```

### Rendering the map

Let us now plot the map in an interactive component:

```
[ ]: ox.graph_to_gdfs(G_nx, nodes=False).explore()
```

### Code explanation

The function `ox.graph_to_gdfs` converts the graph to a GeoDataFrame, which is a data structure that allows us to plot the graph in an interactive map. The parameter `nodes=False` indicates that we are not interested in plotting the nodes of the graph (only the edges). Any GeoDataFrame can be plotted using the function `explore()`. As we will see in the next section, we can also explore routes in the map using this function.

## Shortest path

Let us now define a function that uses this map to calculate the shortest path.

```
[ ]: def get_route(source, destination):
    """
    Given two coordinates (source and destination) calculate the shortest path
    ↪ between the closest nodes to the source and
    the closest node to the destination.
    A point is a tuple of two double values specifying latitude and longitude (e.g.
    ↪ point_1 = (39.464060144309364,
    -0.3624288516715025)).

    Args:
        source - A tuple containing latitude and longitude values of the source point.
        destination - A tuple containing latitude and longitude values of the source
    ↪ point.

    Returns:
        route: The shortest path along the map
        distance: the minimum distance in meters (summation of great-circle distance
    ↪ between nodes in the shortest path)
    """
    # Get the nearest node to source
    node_1 = ox.distance.nearest_nodes(G_nx, X=source[0], Y=source[1])

    # Get the nearest node to destination
    node_2 = ox.distance.nearest_nodes(G_nx, X=destination[0], Y=destination[1])

    # Get the shortest path between source and destination nodes
    route = ox.shortest_path(G_nx, node_1, node_2)

    # Return the route and the nearest nodes

    return route, [node_1, node_2]
```

### Code Explanation

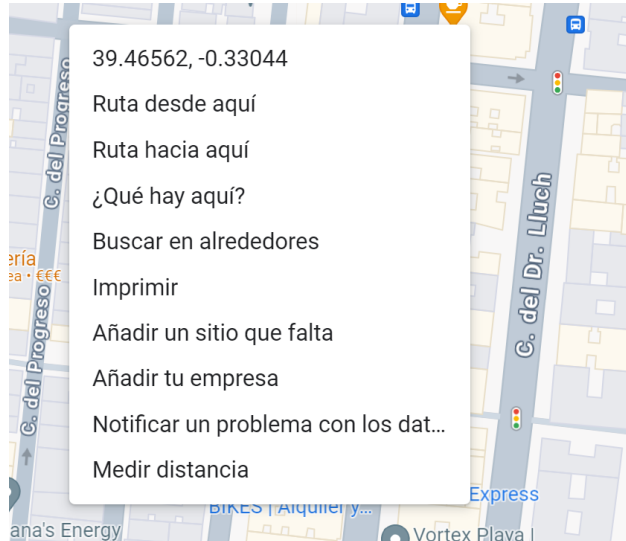
The function above uses the following functions from the libraries:

- `ox.distance.nearest_nodes(G_nx, X=source[0], Y=source[1])`: This function returns the nearest node to a given point. The point is specified by its latitude and longitude.
- `ox.shortest_path(G_nx, node_1, node_2)`: This function returns the shortest path between two nodes in the graph.

The function basically uses the previous functions to first find the nearest nodes to the source and destination points, and then calculates the shortest path between them. Finally, it returns the shortest path and the nearest nodes of the graph which are closer to the provided coordinates.

### Test between two points

Let us test the function!. From Google Maps, it is very easy to get the coordinates of a point. Just right-click on the map, and it will show the coordinates. After, you can hover over the coordinates and left click to copy the coordinates to your clipboard. Now, if you paste in Google Colabs, you will get the coordinates of the point.



Note that the coordinates are in the form (latitude, longitude). However, the function `get_route` expects the coordinates in the form (longitude, latitude). Therefore, we need to invert the coordinates before passing them to the function.

Here is an example of two points in Valencia:

- **EDEM**: Coordinates (-0.3288013882525529, 39.46211739713285)
- **CIGIP Research Center @ UPV**: Coordinates (-0.3344765250611888, 39.46211739713285)

```
[ ]: edem = (-0.3288013882525529, 39.46211739713285)
      cigip = (-0.3344765250611888, 39.46211739713285)

      route, nodes = get_route(edem, cigip)
```

### Converting to GeoDataFrame

Now, we can convert the route to a GeoDataFrame, which is a dataframe containing the information of the route. This will allow us for instance to calculate the total distance or to plot the route in a map. The function used to convert the route to a GeoDataFrame is `ox.utils_graph.route_to_gdf`. The function takes as input the graph, the route and the attribute that we want to use to calculate the distance. In this case, we are using the attribute `length`, which is the length of the street in meters.

```
[ ]: route_edges = ox.utils_graph.route_to_gdf(G_nx, route, "length")
      display(route_edges.head())
```

### Calculating the distance

Now, we can calculate the distance of the route by summing the length of the streets in the route. The length of the streets is stored in the column `length` of the `GeoDataFrame`. Therefore, we can use the function `sum` to calculate the total distance.

```
[ ]: total_distance = sum(route_edges["length"])
print("The total distance between the two points is: " + str(total_distance) + "
↪meters")
```

### Exploring the route

Finally, we can plot the route in a map using the function `explore` of the `GeoDataFrame`. The function takes as input the map where we want to plot the route. We can use the argument `color` to specify the color of the route, the argument `style_kwds` to define style properties, and the argument `m` to specify the map where we want to plot the route. If we do not specify the map, the function will create a new map. In the example below we set the weight of the route to 5 to plot it in a thicker line.

```
[ ]: route_edges.explore( style_kwds={"weight": 5})
```

### Getting the routes of a distribution network

We can use the function `get_route` and the information contained in `Geo Dataframes` to get the routes of a distribution network. For instance, let us calculate the routes for a distribution network with two sources and two destinations in Valencia.

```
[ ]: sources = [
    {
        "id": 1,
        "coordinates": (-0.3783881019008758, 39.47875115874353)},
    {
        "id": 2,
        "coordinates": (-0.3929897392129934, 39.4817119703732)
    }
]

destinations = [
    {
        "id": 1,
        "coordinates": (-0.3346191716152525, 39.47696577364303)},
    {
        "id": 2,
        "coordinates": (-0.3288013882525529, 39.46211739713285)
    }
]

# Init the routes and distances
distances = []

routes = []

# Iterate over the sources and destinations
for source in sources:
    for destination in destinations:
        route, nodes = get_route(source["coordinates"], destination["coordinates"])
        route_gdf = ox.utils_graph.route_to_gdf(G_nx, route, "length")
        routes.append(
            {
```

(continues on next page)

(continued from previous page)

```

        "source": source["id"],
        "destination": destination["id"],
        "route": route_gdf
    }
)
distance = sum(route_gdf["length"])
distances.append(
    {
        "source": source["id"],
        "destination": destination["id"],
        "distance": distance
    }
)

```

### Code Explanation

In the code above, we have defined two dictionaries with the information of sources and destinations, and then we have iterated over them to calculate the routes and distances, using the `get_route` function defined before. We have stored the different routes in a list of GeoDataFrames, and the distances in a list of dictionaries. The dictionaries contain the information of the source, destination and distance of each route.

Now, we can for instance display a dataframe with the distances:

```
[ ]: import pandas as pd
pd.DataFrame(distances)
```

We can also plot the routes using the following script:

```
[ ]: # Define a color map so that the routes have different colors depending on the source
# we use source ids as keys and colors as values. The colors are semi-transparent so
↳ that we can see the routes even if they overlap
color_map = {
    1: "#ff000055",
    2: "#0000ff55"
}

# Add the first route to a map
m = routes[0]["route"].explore(color=color_map[routes[0]["source"]], style_kwds={
↳ "weight": 5})

# Add the rest of the routes to the map
for route in routes[1:]:
    route["route"].explore(color=color_map[route["source"]], style_kwds={"weight": 5}
↳ , m=m)

# Display the map
m
```



## Analysis questions

1. Try the scripts with other points in the map. What is the shortest path between them?
2. What are the main limitations of this approach? How could we improve it?
3. Try to get the distance using a mapping application (e.g. Google Maps). What is the difference between the distance calculated by the script and the distance calculated by the application? Why?
4. Ask an AI assistant to describe the concept of Manhattan distance to you. Calculate the Manhattan distance between the two points. What is the difference between the distance calculated by the script and the distance calculated by the application? Why? Could you use the Manhattan distance to improve the script?

## 3.2 Exercises

In this section you have a collection of CLP problems sorted by difficulty:

- **Easy problems:** You can tackle these problems after the first lesson on MIP.
- **Normal problems:** These problems require that you are familiar with concepts like Duality and the Simplex method.
- **Hard problems:** This is the type of problems that definitively get you ready for the exam.

### 3.2.1 Tarmacking Shifts

The firm MMM has obtained a contract to tarmac the streets in the centre of Exeter. The Department of Traffic Engineering and Planning estimated that at least the number of employees indicated in the table below are required for each 4-h interval over a standard 24-h period. All of MMM's members of staff work 8-h ongoing shifts. There are six feasible shifts which begin on the starting hour of the 4-h intervals indicated in the table. All the members of staff are paid the same salary per hour, except those working between 20:00 h and 00:00 h, who receive a 50 % increase. Moreover, the salary per hour is 100 % higher between 00:00 h and 06:00 h.

Time period	Required Personnel
06–10	70
10–14	20
14–18	80
18–22	30
22–02	10
02–06	10

**a)** Consider a linear programming model that can determine how many employees are required in all six shifts in order to minimise the cost of MMM's salaries to meet the personnel requirements.

**b)** The temporary work ETS firm offers the members of staff more flexible working hours, with 4-h shifts beginning on the hour at the start of all the 4-h periods in the table. The cost of using ETS' staff is the same at all the times considered, but is 80 % higher than the daily cost of MMM's members of staff for each shift. MMM wishes to know if there would be any profit made if ETS was used. Amend the former model to determine the optimum mix of using MMM's and ETS' staff in order to minimise the costs incurred to MMM through salaries and to meet the personnel requirements.

### 3.2.2 Production Orders

#### Problem Definition

The firm H.G. Welds, dedicated to welding components for the automotive sector, has received five orders (P1, P2, P3, P4, P5), which have to be carried out. To meet these orders, the company has five welding machines available (M1, M2, M3, M4, M5). Each machine can carry out every task at the cost shown in the table below. The problem consists in determining optimum allocation which minimises the total cost of carrying out orders by assuming that each machine can do only one order and that all the orders must be carried out.

Order/Machine	M1	M2	M3	M4	M5
P1	16	4	9	5	6
P2	2	14	7	5	13
P3	8	10	3	12	11
P4	3	7	6	10	5
P5	3	6	8	11	7

**Formulate an integer linear programming model to determine the optimum allocation plan. Assume that processing each order on each machine requires an average preparation cost of 10 units. How is the model in the former sector amended?**

[ ]:

### 3.2.3 Planning Production and Inventories

Consider the production of a single product in a planning horizon in T periods. If production during a given period  $t(t = 1, \dots, T)$  is decided, a fixed cost  $cf_t$  is incurred. Any excess products manufactured during early periods can be stored to meet the demand for later periods. Besides, all the demand must be met during each period. Production capacity constraints are not considered.

As  $t = 1, \dots, T$ :

- $d_t$  is the demand for this product during each period,
- $cp_t$  are the cost profits of production during each period, and
- $ca_t$  are the cost profits of storage during each period.

a) Formulate an integer linear programming model which minimizes the total costs of production, storage and fixed costs

b) Assume that delays in deliveries have a cost  $crd_t$  per demand unit not delivered on time during each period. However, all the demand must be met during the last period T, or in other words, a delay in the demand during period T must be null. Amend the model in the former section to contemplate this option.

c) Assume that production can take place in a maximum of five periods, although these periods cannot occur consecutively. Amend the model in the former section to contemplate this option.

[ ]:

### 3.2.4 Problem definition

4 Trucks are available to deliver gas to 5 gas stations. Each truck has a different capacity and if the company decides to use it to supply to any gas stations, the cost can be modeled by a constant operating cost. The capacity and daily operating costs of each truck are shown in the table below.

	Capacity (Liters)	Daily Operating Costs
Truck 1	4000	45
Truck 2	5000	50
Truck 3	6000	55
Truck 4	11000	60

Each gas station can be supplied only by one truck, but a truck may deliver to more than one gas station. The daily demands of each gas station are shown in the table below.

	Daily Demand (liters)
Station 1	1000
Station 2	2000
Station 3	3000
Station 4	5000
Station 5	8000

Formulate an Integer Program Problem that can be used to minimise the distribution costs of the logistics operations needed to satisfy the demand of the five stations.

[ ]:

### 3.2.5 Arcadia deliveries

#### Problem definition

Arcadia Deliveries wants to determine the best location for their warehouses to supply the different retail regions of their best customer. There are 5 possible warehouse locations and 10 different retail regions. The following table shows the capacity and operation costs of the possible warehouse locations:

Data	warehouse 1	warehouse 2	warehouse 3	warehouse 4	warehouse 5
capacity	1000	2000	3000	4000	5000
Operation costs	50	60	70	80	90

The following table contains the demand at each region:

Region	Demand
Region 1	100
Region 2	200
Region 3	300
Region 4	400
Region 5	500
Region 6	600
Region 7	700
Region 8	800
Region 9	900
Region 10	1000

And the following cost contains the transportation costs from warehouses to regions:

Region	Warehouse 1	Warehouse 2	Warehouse 3	Warehouse 4	Warehouse 5
1	42	36	58	47	32
2	73	56	73	78	38
3	38	41	80	88	69
4	59	70	77	94	51
5	51	92	79	88	80
6	38	98	57	64	96
7	86	53	86	58	44
8	48	92	40	100	71
9	62	60	30	31	63
10	43	43	48	63	55

Find the optimal location for warehouses and determine which warehouses are going to deliver to every retail location, knowing that the customer requires that a region is supplied from only one source

[ ]:

### 3.2.6 Distribution of pharma

#### Problem Definition

The US Department of Energy conducts a series of experiments in two research facilities located in Hawkins (Atlanta) and Los Angeles. The department needs to supply these facilities with pharma products that are processed in two factories in Portland and Flint. The delivery costs are summarised in the following table:

Production plants	Research Facilities	Delivery costs
Portland	Los Angeles	€30
Portland	Hawkins	€40
Flint	Los Angeles	€60
Flint	Hawkins	€50

The supply and demand data is provided in the table below:

Supply	Available units	Demand	Required units
Portland	200	Los Angeles	300
Flint	600	Hawkins	400

Implement an MILP that determines how to supply the two research facilities at minimum cost.

[ ]:

### 3.2.7 Transport Planning

A firm must transport machines from production plants A, B and C to warehouses X, Y and Z. Five machines are required in X, 4 in Y and 3 in Z, whereas 8 machines are available in A, 5 in B and 3 in C. The transport costs (in euros) between sites are provided in the table below.

Plant/Warehouse	X	Y	Z
A	50	60	30
B	60	40	20
C	40	70	30

- a) Formulate an integer linear programming model that minimizes transport costs.
- b) Assume that the cost of transporting a machine from plant B increases by €10 for all the machines as of the third one; that is, the 4th, the 5th, etc. Reformulate the model in Section a) by considering this assumption.

### 3.2.8 Planning production in a plastic injection company

#### Problem definition

Patrick “Eel” O’Brian’s (PEOB) Ltd. is a company that manufactures plastic components for the automotive sector. As an intern at PEOB Ltd, your first assignment is to write an Integer Programming Model to calculate the optimal production plan for the plastic injection machine. The injection machine heats plastic and injects it into a mould to get a specific shape. Each mould can thus only be used to manufacture specific component types. Your first version will take into account the storage costs and the delayed orders cost which are defined below.

**Storage costs** The storage or inventory costs represent the cost of storing the inventory levels of every component type at every planning period and are modeled as a fixed cost per unit and planning period. The storage costs are different for every component type.

**Delayed orders costs** At every planning period, you may delay part of the demand for the next planning period. Customers will apply a fixed penalty for every delayed unit at every period. The delayed costs are also different for every component type.

Your model needs to take into account the following additional data:

**Initial Inventory Levels (units):** There is an initial inventory level for every component type available at the first planning period that needs to be taken into account.

**Minimum and Maximum Inventory Levels (units):** There is a maximum and a minimum inventory level for every component type.

**Machine capacity (units):** The machine capacity represents the number of units of a given component type that the machine can produce using a given mould. If a mould cannot be used to manufacture a component type, the machine capacity is zero for that combination of component type and mould.

**Demand (Units):** The company has several confirmed orders for the following periods and therefore, our model needs to take into account the demand for every product at every planning period.

- a. Write down the indexes and decision variables
- b. Write down the objective function
- c. Write down the constraints

You have successfully validated your model, and it is already providing valuable information for the company. Now the company would like to extend the model to take into account as well the set-up costs:

**Set up costs** As explained above, the injection machine uses different moulds to manufacture different parts. Each time that a mould is changed to make a different component, the operators need to set up the machine. The company estimates that this setup cost only depends on the mould that is used and is only applied when a product is changed.

- d. Modify the model to take into account the set-up costs.

### 3.2.9 Machinery Transport Planning

#### Problem Definition

A firm must transport machines from production plants A, B and C to warehouses X, Y and Z. Five machines are required in X, four in Y and three in Z and there are eight machines available in A, five in B and three in C. **a)** Create and solve a Graph Theory model to determine the maximal flow of the machines that can be transported and the run of this flow. **b)** By assuming a mean cost of €45 per transported machine, what would the total cost be of transporting the machines obtained in the former section?

### 3.2.10 Overbooking on Airlines

#### Problem Definition

A low-cost airline, OrgaAir, operates four daily flights from Valencia to London at 10:00, 12:00, 14:00 and 16:00 hours. The first two flights hold 100 passengers and the last two can fly up to 150 passengers each. Should overbooking occur, which implies having sold more seats than the plane actually has, the airline can place a passenger on a later flight. Evidently each delayed traveller is compensated by being paid €200 plus €20 for each hour delayed. The firm places delayed travellers on their own operating flights, or on one of the flights from other airlines departing at 20:00 hours, which always have available seats (a capacity of 999 passengers is always considered) at no extra charge. Let us assume that at the beginning of the day we know that OrgaAir has sold 110, 160, 100 and 100 seats on its four daily flights, respectively. Model this problem as a maximal flow model at a minimum cost in order to minimise the airline's total overbooking cost.

### 3.2.11 Production Planning

#### Problem Definition

PLACASA, a supplier of plastic injection parts for the automobile sector, foresees the demand of an article for the following 4 months of 100, 140, 210 and 180 units, respectively. The firm can maintain only sufficient stock to meet the demand of each month, or it can have excess stock to meet the demand of two successive months or more with a storage cost of 1.2€ per month and unit of excessive stock. PLACASA calculates that the production cost for the following months, is €15, €12, €10 and €14 per unit. A preparation cost of €200 is incurred every time a production order is placed. The firm wishes to develop a production plan that minimises the total costs of production orders, of production and of keeping an article in stock. Formulate and solve the shortest path model to find an optimum production plan.

[ ]:

### 3.2.12 Production Sequencing

#### Problem Definition

A manufacturer of the metal sector in Stafford manufactures four types of products in sequence on two machines. The Table below provides the necessary technical production details. Taking into account only the data on the production times per product and the maximum daily production capacity of each machine, and taking a maximum daily production of 100 units of each product type, consider and solve a maximal flow model in a graph to determine the maximum number of minutes during which machines 1 and 2 can operate on a production-in-sequence basis given the existing capacity constraints.

Machine	Product 1	Product 2	Product 3	Product 4	Daily Production Capacity (minutes)
1	2	3	4	2	500
2	3	2	1	2	380

### 3.2.13 Planet Express Deliveries

#### Problem definition

Planet Express deliveries wants to optimize their transportation network from a set of source warehouses to a set of destinations. In particular, they want to minimize the transportation costs through a set of intermediate warehouses, so that items are transported from the source warehouses, through the intermediate warehouses, to the destination regions.

Let us first consider that transportation costs per item from the source warehouse  $i$  to intermediate warehouse  $j$  as  $s_{ij}$  and that the transportation costs per item from intermediate warehouse  $j$  to destination region  $k$  is  $t_{jk}$ . Let us note the capacity of source warehouses as  $a_i$  items, the capacity of intermediate warehouses  $j$  as  $b_j$  items, and the demand for region  $k$  as  $d_k$  items. Let us also consider that  $m$  is the number of source warehouses,  $n$  is the number of intermediate warehouses and  $p$  is the number of destination regions.

**a** Consider that  $m = p = 3$  and  $n = 5$  and draw a network to solve the transportation problem as a maximal flow at minimum cost problem.

**b** Model the problem as a Mixed Integer Programming (MIP), not taking into account the values of  $m, p$ , and  $n$  defined in the previous section

**c** Planet Express is now considering the use of warehouses from an external provider. The provider of the service has a total of  $n_e$  external warehouses. They offer a fixed cost  $F_l$  when an external warehouse  $l$  is used, plus a cost per item

received and processed in warehouse  $l$ ,  $r_l$ . External warehouse  $l$  has a capacity of  $c_l$  items. Modify your model to incorporate this offer into your distribution network.

### 3.2.14 3D Printing Precision Parts

#### Problem Definition

You work for a company that manufactures high quality polymer parts for OrgoCorp, a galactic bioengineering company. The quality requirements are so high that you need to use very sophisticated 3D printing machines. You need to develop a mathematical programming model to organize monthly production considering:

- **Fixed operational costs (euros):** Each 3D printing machine  $m$  has a different fixed operational costs  $F_m$  that needs to be considered when a printing machine is used to print any units.
- **Unitary production costs (euros):** Each 3D printing machine  $m$  produces parts of product type  $p$  at a different unitary costs  $C_{mp}$
- **Capacity (minutes):** Each 3D printing machine  $m$  has a different capacity  $S_m$  (minutes)
- **Speed (minutes/part):** Each 3D printing machine  $m$  prints a unit of product type  $p$  at a different unitary speed
- **Demand (units):** OrgoCorp has confirmed a demand for  $d_p$  units of every product type  $p$  Build a model that takes into account these requirements, identifying indices, decision variables, objective function and constraints.

The CEO of OrgoCorp, The High Evolutionary, would like to set up a new contract allowing more flexibility in the delivery of units, such that:

- **Demand per period (units):** Now you need to consider that the demand for every product type  $p$  is not constant and depends on the planning period  $t$ ,  $D_{pt}$
- **Delayed demand (units):** You may delay the production for one period. For every unit of product type  $p$  that is delayed, OrgoCorp will charge an extra delayed demand cost of  $B_{pt}$  Modify the model to take into account this new contract

### 3.2.15 Spellman’s Distribution Network Optimization

Spellman’s would like Sabrina to focus on the optimization of their distribution network, to support the distribution of the 2 types of drinks from 3 sourcing facilities to 4 destination warehouses, through 6 intermediate warehouses. Sabrina needs to take into account the transportation costs (both from sourcing warehouses to intermediate warehouses and from intermediate warehouses to final destinations). Additionally, Sabrina can decide how many of the 6 intermediate solutions to use, given that they have a fixed operational cost that needs to be paid when the warehouse is used.

The following tables contain the transportation problem data:

#### Capacities of Sourcing Facilities (in units)

Sourcing Warehouse	Product A Capacity (units)	Product B Capacity (units)
S1	300	400
S2	250	150
S3	350	250



**Demands at Final Destinations (in units)**

Destination Warehouse	Product A Demand (units)	Product B Demand (units)
D1	150	150
D2	200	250
D3	175	175
D4	225	175

**Sourcing to Intermediate Warehouses Transportation Costs (Euros)**

From/To	I1	I2	I3	I4	I5	I6
S1	2	4	5	7	3	6
S2	3	2	6	8	4	5
S3	1	3	4	6	5	7

**Intermediate to Destination Warehouses Transportation Costs (Euros)**

From/To	D1	D2	D3	D4
I1	5	2	3	4
I2	4	1	5	2
I3	3	4	2	5
I4	5	3	4	1
I5	2	5	3	4
I6	4	2	1	5

**Operational Costs of Intermediate Warehouses (Euros)**

Warehouse	Operational Cost
I1	1000
I2	800
I3	1200
I4	1100
I5	700
I6	900

Note that the distribution costs are independent of the product type.

1. Help Sabrina write an Integer Programming Problem to model the transportation problem .
2. How would a Greedy algorithm solve the problem? Motivate your response

### 3.2.16 Professors Xavier’s School for Gifted Youngsters

#### Problem Definition

Professor Xavier’s School for Gifted Youngsters is organizing a special summer session focusing on enhancing the students’ unique abilities. Students from across the country have been invited, and the school needs to arrange their transportation. The goal is to ensure that all students can arrive at the school in time for the session while minimizing transportation costs.

1. First, let us consider an abstract model where gifted students reside in  $n$  different locations, can be transported in  $m$  different modes of transport. Each mode of transport from each location to the school has a different associated cost. Additionally, each transportation mode has a different capacity, and each location has a different number of gifted students that need transportation to the school. Write down an Integer Programming (IP) problem that minimizes the overall transportation costs.
2. Given the following number of students:

Location	Number of Students
L1	20
L2	30
L3	25
L4	15
L5	40
L6	35

And the following capacities and costs of transport per student from the different locations for three different modes of transport:

Mode	Capacity	L1	L2	L3	L4	L5	L6
T1	50	10	8	12	11	9	7
T2	60	9	7	11	10	8	6
T3	40	8	9	10	12	7	11

Draw a graph that allows to model the transportation problem as a maximum flow at minimum cost transportation problem.

### 3.3 Solved Exercises

In this section you have the solution to the different exercises.

#### 3.3.1 Tarmacking Shifts (solved)

The firm MMM has obtained a contract to pave the streets in the centre of Exeter. The Department of Traffic Engineering and Planning estimated that at least the number of employees indicated in the table below are required for each 4-h interval over a standard 24-h period. All of MMM’s members of staff work 8-h ongoing shifts. There are six feasible shifts which begin on the starting hour of the 4-h intervals indicated in the table. All the members of staff are paid the same salary per hour, except those working between 20:00 h and 00:00 h, who receive a 50 % increase. Moreover, the salary per hour is 100 % higher between 00:00 h and 06:00 h.

Time period	Required Personnel
06–10	70
10–14	20
14–18	80
18–22	30
22–02	10
02–06	10

a) Consider a linear programming model that can determine how many employees are required in all six shifts in order to minimise the cost of MMM's salaries to meet the personnel requirements.

### Decision variables

In this case, we need to decide how many employees work on each shift. Therefore, the decision variables are:

$X_j$ : (Integer) Number of employees in shift  $j$ ,  $j = [1, \dots, 6]$

### Objective function

The objective is to minimise the cost:

$$\min z = \sum_{j=1}^6 C_j * X_j$$

The cost per shift depends on the hourly costs. If we define the base hour cost as  $C$ , and taking into account that employees get 50% extra between 20:00 and 00:00 and double the money between 00:00 and 06:00, we can calculate the cost of every shift as:

$$X_1 (06:00 - 14:00) \rightarrow C_1 = 8 \cdot C$$

$$X_2 (10:00 - 18:00) \rightarrow C_2 = 8 \cdot C$$

$$X_3 (14:00 - 22:00) \rightarrow C_3 = 6 \cdot C + 1.5 \cdot 2 \cdot C = 9 \cdot C$$

$$X_4 (18:00 - 02:00) \rightarrow C_4 = 2 \cdot C + 4 \cdot 1.5 \cdot C + 2 \cdot 2 \cdot C = 12 \cdot C$$

$$X_5 (22:00 - 06:00) \rightarrow C_5 = 2 \cdot 1.5 \cdot C + 6 \cdot 2 \cdot C = 15 \cdot C$$

$$X_6 (02:00 - 10:00) \rightarrow C_6 = 4 \cdot 2 \cdot C + 4 \cdot C = 12 \cdot C$$

### Constraints

Now, we have six constraints, and taking into account that the employees work for two time periods straight:

$$X_6 + X_1 \geq 70$$

$$X_1 + X_2 \geq 20$$

$$X_2 + X_3 \geq 80$$

$$X_3 + X_4 \geq 30$$

$$X_4 + X_5 \geq 10$$

$$X_5 + X_6 \geq 10$$

b) The temporary work ETS firm offers the members of staff more flexible working hours, with 4-h shifts beginning on the hour at the start of all the 4-h periods in the table. The cost of using ETS' staff is the same at all the times considered, but is 80 % higher than the daily cost of MMM's members of staff for each shift. MMM wishes to know if there would be any profit made if ETS was used. Amend the former model to determine the optimum mix of using MMM's and ETS' staff in order to minimise the costs incurred to MMM through salaries and to meet the personnel requirements.

Now, the ETS employees only work on 4h shifts, their hourly costs are fixed, but at the same time, 80% higher than the base hourly cost of normal employees. We have again six new shifts, and therefore, 6 new decision variables:

$Xe_j$ : (Integer) Number of ETS employees in shift  $j$ ,  $j = [1, \dots, 6]$

Now, we need to amend the objective function to account for this new employees, since the hourly costs are fixed, there is no need to calculate it:

$$\min z = \sum_{j=1}^6 C_j * X_j + \sum_{j=1}^6 1.8 \cdot C * Xe_j$$

As for the constraints, we simply introduce the ETS employees in every shift:

$$X_6 + X_1 + Xe_1 \geq 70$$

$$X_1 + X_2 + Xe_2 \geq 20$$

$$X_2 + X_3 + Xe_3 \geq 80$$

$$X_3 + X_4 + Xe_4 \geq 30$$

$$X_4 + X_5 + Xe_5 \geq 10$$

$$X_5 + X_6 + Xe_6 \geq 10$$

### 3.3.2 Planning Production and Inventories

#### Problem definition

Consider the production of a single product in a planning horizon in  $T$  periods. If production during a given period  $t$  ( $t = 1, \dots, T$ ) is decided, a fixed cost  $cf_t$  is incurred. Any excess products manufactured during early periods can be stored to meet the demand for the next period. Besides, all the demand must be met during each period. Production capacity constraints are not considered.

As  $t = 1, \dots, T$ :

- $d_t$  is the demand for this product during each period,
- $cp_t$  are the cost profits of production per unit during each period, and
- $ca_t$  are the cost profits of storage per unit during each period.

a) Formulate an integer linear programming model which minimizes the total costs of production, storage and fixed costs

b) Assume that we are allowed to have a delay of one period. Delays in deliveries have a cost  $crd_t$  per demand unit not delivered on time during each period. However, all the demand must be met during the last period  $T$ , or in other words, a delay in the demand during period  $T$  must be null. Amend the model in the former section to contemplate this option.

c) Assume that production can take place in a maximum of five periods, although these periods cannot occur consecutively. Amend the model in the former section to contemplate this option.

#### Problem Model

##### Decision Variables

$X_t$  = Units to produce in period  $t$

$I_t$  = Units to store in period  $t$

$Y_t$  = Binary variable to determine whether we produce in one period or not

### Objective function

In section a. we consider production costs, the fixed costs, plus the storage costs:

$$\min z = \text{production\_cost} + \text{fixed\_costs} + \text{storage costs}$$

The objective function is therefore:

$$\min z = \sum_{t=1}^T cp_t \cdot X_t + \sum_{t=1}^T cf_t \cdot Y_t + \sum_{t=1}^T I_t ca_t$$

### Constraints

In this case, we need to make sure that we meet the demand in every period, then:

$$I_{t-1} + X_t - I_t \geq d_t, \forall t$$

We also need to make sure that the binary decision variable makes sense:

$$X_t \leq M * Y_t, \forall t$$

(Where M is a large number. If we do not make any units in a period, then the binary decision variable must be zero).

Also, they must be non-negative:

$$X_t \geq 0, \forall t$$

$$I_t \geq 0, \forall t$$

### b) Delays

To take into account the delays, we need to introduce a new term in the objective function. We introduce a new decision variable:

$$Du_t = \text{Units to delay at period } t$$

And the objective function becomes:

$$\min z = \sum_{t=1}^T cp_t \cdot X_t + \sum_{t=1}^T cf_t \cdot Y_t + \sum_{t=1}^T I_t ca_t + \sum_{t=1}^T Du_t crd_t$$

We also need to amend the first constraint:

$$I_{t-1} + X_t - I_t - Du_{t-1} + Du_t \geq d_t, \forall t$$

Also, we need to make sure that there are no delays at the end of the period:

$$Du_T = 0$$

### c) Maximum number of periods

If we have a maximum number of periods, we can introduce another constraint using the binary variable:

$$\sum_{t=1}^T Y_t \leq 5$$

Also, since we cannot produce in two consecutive periods, the sum of two consecutive values of the binary variable cannot be higher than one:

$$Y_t + Y_{t-1} \leq 1, \forall t$$

[ ]:

### 3.3.3 Production Orders

#### Problem Definition

The firm H.G. Welds, dedicated to welding components for the automotive sector, has received five orders (P1, P2, P3, P4, P5), which have to be carried out. To meet these orders, the company has five welding machines available (M1, M2, M3, M4, M5). Each machine can carry out every task at the cost shown in the table below. The problem consists in determining optimum allocation which minimises the total cost of carrying out orders by assuming that each machine can do only one order and that all the orders must be carried out.

Order/Machine	M1	M2	M3	M4	M5
P1	16	4	9	5	6
P2	2	14	7	5	13
P3	8	10	3	12	11
P4	3	7	6	10	5
P5	3	6	8	11	7

**Formulate an integer linear programming model to determine the optimum allocation plan. Assume that processing each order on each machine requires an average preparation cost of 10 units. How is the model in the former sector amended?**

#### Problem model

This is an instance of an assignment problem.

#### Decision variables

$x_{ij} = 1$  if Machine  $M_i$  carries out order  $P_j$ , and 0 otherwise;

#### Objective function

It represents the total cost of carrying out tasks, and can be expressed as the sum of all the costs  $c_{ij}$  which are the elements of the cost matrix in the table. Note that the rows in the table represent the costs associated to a production order, whereas in our matrix, each row represents the costs associated to a machine, so we need to **transpose** the table provided:

$$C = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mn} \end{bmatrix} = \begin{bmatrix} 16 & 2 & 8 & 3 & 3 \\ 4 & 14 & 10 & 7 & 6 \\ 9 & 7 & 3 & 6 & 8 \\ 5 & 5 & 12 & 10 & 11 \\ 6 & 13 & 11 & 5 & 7 \end{bmatrix}$$

Now, the objective function is to **minimize costs**, therefore:

$$\begin{aligned} \min z = \sum_i \sum_j x_{ij} * c_{ij} &= 16x_{11} + 2x_{12} + 8x_{13} + 3x_{14} + 3x_{15} + \dots \\ &\dots + 4x_{21} + 14x_{22} + 10x_{23} + 7x_{24} + 6x_{25} + \dots \\ &\dots + 9x_{31} + 7x_{32} + 3x_{33} + 6x_{34} + 8x_{35} + \dots \\ &\dots + 5x_{41} + 5x_{42} + 12x_{43} + 10x_{44} + 11x_{45} + \dots \\ &\dots + 6x_{51} + 13x_{52} + 11x_{53} + 5x_{54} + 7x_{55} \end{aligned}$$

Constraints: It must be ensured that each machine does only one of the tasks:

$$\sum_j x_{ij} \leq 1 \forall i$$

That is:

$$x_{11} + x_{12} + x_{13} + x_{14} + x_{15} \leq 1$$

$$x_{21} + x_{22} + x_{23} + x_{24} + x_{25} \leq 1$$

$$x_{31} + x_{32} + x_{33} + x_{34} + x_{35} \leq 1$$

$$x_{41} + x_{42} + x_{43} + x_{44} + x_{45} \leq 1$$

$$x_{51} + x_{52} + x_{53} + x_{54} + x_{55} \leq 1$$

We also need to ensure that all the orders are carried out:

$$\sum_i x_{ij} \leq 1 \forall j$$

That is:

$$x_{11} + x_{21} + x_{31} + x_{41} + x_{51} \geq 1$$

$$x_{12} + x_{22} + x_{32} + x_{42} + x_{52} \geq 1$$

$$x_{13} + x_{23} + x_{33} + x_{43} + x_{53} \geq 1$$

$$x_{14} + x_{24} + x_{34} + x_{44} + x_{54} \geq 1$$

$$x_{15} + x_{25} + x_{35} + x_{45} + x_{55} \geq 1$$

[ ]:

### 3.3.4 Problem definition

4 Trucks are available to deliver gas to 5 gas stations. Each truck has a different capacity and if the company decides to use it to supply to any gas stations, the cost can be modeled by a constant operating cost. The capacity and daily operating costs of each truck are shown in the table below.

	Capacity (Liters)	Daily Operating Costs
Truck 1	4000	45
Truck 2	5000	50
Truck 3	6000	55
Truck 4	11000	60

Each gas station can be supplied only by one truck, but a track may deliver to more than one gas station. The daily demands of each gas station are shown in the table below.

	Daily Demand (liters)
Station 1	1000
Station 2	2000
Station 3	3000
Station 4	5000
Station 5	8000

Formulate an Integer Program Problem that can be used to minimise the distribution costs of the logistics operations needed to satisfy the demand of the five stations.

### Model definition

#### Indexes

- $i$ : Gas stations
- $j$ : Trucks

#### Decision Variables

- $Y_j$  (Binary): Truck  $j$  is used
- $X_{ij}$  (Binary): Truck  $j$  delivers to gas station  $i$

#### Objective Function

Minimise cost:

$$\min z = \sum Y_j \cdot C_{o_j}$$

Where  $C_{o_j}$  is the fixed operational cost of using truck  $j$ .

#### Constraints

##### Single source delivery constraint

First, we need to consider the single source constraint, to make sure that only 1 truck delivers to a gas station:

$$\sum_j X_{ij} = 1 \quad \forall i$$

##### Demand constraint

However, we also need to take into account the **demand**, to make sure that we satisfy the daily demand in each gas station:

$$\sum_j d_i * X_{ij} = d_i \quad \forall i$$

Note that these two constraints (single source delivery constraint and demand constraint) are **equivalent**. We can just use the former in our model.

##### Capacity constraint

We also need to take into account the **capacity constraint** to make sure that the amount delivered from any truck is lower than its capacity:

$$\sum_i d_i * X_{ij} \leq c_j \quad \forall j$$

##### Logical constraint

We need to introduce a logical constraint as well, to make sure that the binary decision variables are consistent with each other. Basically, we need to ensure that we only use one truck (i.e.  $Y_j = 1$ ) when it delivers to any gas station (i.e. any  $X_{ij} = 1$  for the same  $j$ ):

$$X_{ij} \leq Y_j \quad \forall i, \forall j$$

Another way to express the logical constraint is:

$$\sum_i X_{ij} \leq M * Y_j \quad \forall i, \forall j$$

Where  $M$  is a very large number. In this form, basically what we say is that if a truck is not used (i.e.  $Y_j = 0$ ), then the sum of the gas stations it delivers to must be 0. Otherwise, if it is used (i.e.  $Y_j = 1$ ), the sum has to be lower than a very large number  $M$ .

Note that, we can also merge this logical constraint with the capacity constraint to yield:

$$\sum_i d_i * X_{ij} \leq c_j Y_j \quad \forall j$$



In this form, the right hand side of the logical constraint is modified to take into account the quantity that each truck delivers when it is used, and the right hand side is modified to take into account the maximum capacity of the truck, when it is used.

With this, our model can be written as:

$$\min z = \sum Y_j \cdot C_{o_j}$$

s.t.

$$\sum_j d_i * X_{ij} = d_i \quad \forall i$$

$$\sum_i d_i * X_{ij} \leq c_j Y_j \quad \forall j$$

## Solution in Python

### Requirements

Run the following script to install the required packages

```
[ ]: !pip install pulp
      !pip install pandas
```

```
[4]: import pulp
      import pandas as pd
      #And we will use numpy to perform array operations
      import numpy as np
      #We will use display and Markdown to format the output of code cells as Markdown
      from IPython.display import display, Markdown

      trucks = [1, 2, 3, 4]
      stations = [1, 2, 3, 4, 5]

      capacities = [4000, 5000, 6000, 11000]

      costs = [45, 50, 55, 60]

      demands = [1000, 2000, 3000, 5000, 8000]

      # Instantiate model
      model = pulp.LpProblem("Truck Deliveries", pulp.LpMinimize)

      x_vars = pulp.LpVariable.dicts("X",
                                     [(i, j) for i in stations for j in trucks],
                                     lowBound=0,
                                     cat='Binary')

      y_vars = pulp.LpVariable.dicts("Y",
                                     [j for j in trucks],
                                     lowBound=0,
                                     cat='Binary')

      model += (
          pulp.lpSum([
              costs[j] * y_vars[trucks[j]]
              for j in range(len(trucks))]
          ), "Fixed operating costs"

      # Demand
```

(continues on next page)

(continued from previous page)

```

for i in range(len(stations)):
    model += pulp.lpSum([
        demands[i]*x_vars[(stations[i], trucks[j])]
        for j in range(len(trucks))]) == demands[i], "station_" + str(stations[i])

# Capacity constraints
for j in range(len(trucks)):
    model += pulp.lpSum([
        demands[i]*x_vars[(stations[i], trucks[j])]
        for i in range(len(stations))]) <= capacities[j]*y_vars[trucks[j]], "truck_" +
    str(trucks[j])

# Solve our problem
model.solve()
print(pulp.LpStatus[model.status])

# Solution
max_z = pulp.value(model.objective)
print(max_z)

```

```

Optimal
155.0

```

```

[7]: var_Y_df = pd.DataFrame.from_dict(y_vars, orient="index",
                                     columns = ["Variables"], dtype=object)

var_Y_df["Solution"] = var_Y_df["Variables"].apply(lambda item: item.varValue)
display(var_Y_df)

var_X_df = pd.DataFrame.from_dict(x_vars, orient="index",
                                  columns = ["Variables"], dtype=object)
var_X_df["Solution"] = var_X_df["Variables"].apply(lambda item: item.varValue)
index = pd.MultiIndex.from_product([stations, trucks], names=['Stations', 'Trucks'])
var_df2 = pd.DataFrame(var_X_df["Solution"], index=index, columns = ["Solution"])
display(var_df2.unstack())

```

	Variables	Solution
1	Y_1	1.0
2	Y_2	1.0
3	Y_3	0.0
4	Y_4	1.0

	Solution			
Trucks	1	2	3	4
Stations				
1	1.0	0.0	0.0	0.0
2	0.0	0.0	0.0	1.0
3	1.0	0.0	0.0	0.0
4	0.0	1.0	0.0	0.0
5	0.0	0.0	0.0	1.0

```
[ ]:
```

### 3.3.5 Arcadia deliveries

Try me



#### Problem definition

Arcadia Deliveries wants to determine the best location for their warehouses to supply the different retail regions of their best customer. There are 5 possible warehouse locations and 10 different retail regions. The following table shows the capacity and operation costs of the possible warehouse locations:

Data	warehouse 1	warehouse 2	warehouse 3	warehouse 4	warehouse 5
capacity	1000	2000	3000	4000	5000
Operation costs	50	60	70	80	90

The following table contains the demand at each region:

Region	Demand
Region 1	100
Region 2	200
Region 3	300
Region 4	400
Region 5	500
Region 6	600
Region 7	700
Region 8	800
Region 9	900
Region 10	1000

And the following cost contains the transportation costs from warehouses to regions:

Region	Warehouse 1	Warehouse 2	Warehouse 3	Warehouse 4	Warehouse 5
1	42	36	58	47	32
2	73	56	73	78	38
3	38	41	80	88	69
4	59	70	77	94	51
5	51	92	79	88	80
6	38	98	57	64	96
7	86	53	86	58	44
8	48	92	40	100	71
9	62	60	30	31	63
10	43	43	48	63	55

Find the optimal location for warehouses and determine which warehouses are going to deliver to every retail location, knowing that the customer requires that a region is supplied from only one source

### Problem model

This is a transportation problem with a single-source facility constraint, where the sources are the sourcing warehouses and the destination the retail regions.

The objective is to minimize the transportation costs, taking into account transportation costs and the operation costs.

**Indices** Let us define the following indices:

- $i$ : warehouse location ( $i \in [1, \dots, 5]$ )
- $j$ : retail region ( $j \in [1, \dots, 10]$ )

**Data** Looking at the data, we can define the following vector containing the operation costs:

$$f = [f_1, f_2, \dots, f_5] = [50, 60, 70, 80, 90]$$

and the following matrix containing the transportation costs from every warehouse to every retail region:

$$C = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{15} \\ c_{21} & c_{22} & \dots & c_{25} \\ \dots & & & \\ c_{101} & c_{102} & \dots & c_{105} \end{bmatrix}$$

Or, using the values given in the table above:

$$C^T = \begin{bmatrix} 42 & 36 & 58 & 47 & 32 \\ 73 & 56 & 73 & 78 & 38 \\ 38 & 41 & 80 & 88 & 69 \\ 59 & 70 & 77 & 94 & 51 \\ 51 & 92 & 79 & 88 & 80 \\ 38 & 98 & 57 & 64 & 96 \\ 86 & 53 & 86 & 58 & 44 \\ 48 & 92 & 40 & 100 & 71 \\ 62 & 60 & 30 & 31 & 63 \\ 43 & 43 & 48 & 63 & 55 \end{bmatrix}$$

Besides the costs, each warehouse has a limited sourcing capacity, which can be identified as a vector:

$$s = [s_1, s_2, \dots, s_5] = [1000, 2000, 3000, 4000, 5000]$$

And the following vector contains the demand for every region:

$$d = [d_1, d_2, \dots, d_{10}] = [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]$$

Let us define the following decision variables:

- $X_{ij}$ : (Binary) determines if warehouse  $i$  delivers all demand to retail region  $j$  {1 if yes, 0 otherwise}
- $Y_i$ : (Binary) determines if warehouse  $i$  delivers to any retail region {1 if yes, 0 otherwise}

**Objective function** The objective function is:

$$\min z = \sum_{i=1}^5 \sum_{j=1}^{10} c_{ij} * d_j * X_{ij} + \sum_{i=1}^5 Y_i * f_i$$

**Constraints** The constraints are:

**Sourcing capacity:**

$$\sum_{j=1}^m d_j * X_{ij} \leq s_i * Y_i \quad \forall i$$

These five constraints represent the limited capacity of the warehouses, we multiply the capacity with the corresponding  $Y_i$  variable to make sure the solution is logical and that the right-hand-side is zero if  $Y_i$  is zero.

**Demand**

$$\sum_{i=1}^n X_{ij} = 1 \quad \forall j$$

Since we source all the demand from a single source, we only need to make sure that the demand is supplied from 1 location for every warehouse.

## Implementation in Python

### Requirements

Install the following requirements to run this cell:

```
[ ]: !pip install pandas
      !pip install pulp
      !pip install IPython
```

### Implementation

```
[6]: # Import libraries
import pandas as pd
import pulp
import numpy as np
from IPython.display import display, Markdown

# Instantiate our problem class

model = pulp.LpProblem("Minimise_transportation_costs", pulp.LpMinimize)

# Define decision variables

# Construct our decision variable lists
warehouse = [1, 2, 3, 4, 5]
region = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

X = pulp.LpVariable.dicts("X_",
                          ((i, j) for i in warehouse for j in region),
                          lowBound=0,
                          cat='Binary')

Y = pulp.LpVariable.dicts("Y_", (i for i in warehouse),
                          lowBound=0,
                          cat='Binary')

# Data
transportation_costs = np.array([[42, 73, 38, 59, 51, 38, 86, 48, 62, 43], [36, 56,
↪41, 70, 92, 98, 53, 92, 60, 43], [58, 73, 80, 77, 79, 57, 86, 40, 30, 48], [47, 78,
↪88, 94, 88, 64, 58, 100, 31, 63], [32, 38, 69, 51, 80, 96, 44, 71, 63, 55]])

operation_costs = np.array([50, 60, 70, 80, 90])

capacity = np.array([1000, 2000, 3000, 4000, 5000])

demand = np.array([100, 200, 300, 400, 500, 600, 700, 800, 900, 1000])

model += pulp.lpSum([demand[j]*transportation_costs[i,j]*X[(warehouse[i],region[j])]
↪for i in range(len(warehouse)) for j in range(len(region))] + pulp.
↪lpSum([operation_costs[i]*Y[warehouse[i]] for i in range(len(warehouse))])
```

(continues on next page)

(continued from previous page)

```
# Capacity constraints
for i in range(len(warehouse)):
    model += pulp.lpSum([demand[j]*X[(warehouse[i],region[j])] for j in
    ↪range(len(region))]) <= capacity[i]*Y[warehouse[i]], "source_capacity_" + str(i)

# Demand constraints
for j in range(len(region)):
    model += pulp.lpSum([demand[j]*X[(warehouse[i],region[j])] for i in
    ↪range(len(warehouse))]) >= demand[j], "demand_region_" + str(j)

model.solve()
print(pulp.LpStatus[model.status])

total_cost = pulp.value(model.objective)
display(Markdown("Total cost is **%0.2f €**"%total_cost))

#Let us now show the solution as pandas dataframes
x_df = pd.DataFrame(columns=warehouse)

display(Markdown("##Decision Variables##"))
display(Markdown("The following table displays the values of the decision variables
↪ $X_{ij}$ $. The orientation is the same as in the table that provided the data in the
↪first place"))
for j in range(len(region)):
    row_name = region[j]
    row_values = {}
    for i in range(len(warehouse)):
        row_values[warehouse[i]]=int(X[(warehouse[i],region[j])].value())

    row_series = pd.Series(row_values, name=row_name)
    x_df = x_df.append(row_series)

display(x_df)

y_df = pd.DataFrame(columns=warehouse)
y_values = {}
for i in range(len(warehouse)):
    y_values[warehouse[i]]=int(Y[(warehouse[i])].value())

y_df = y_df.append(pd.Series(y_values, name="Y"))

display(Markdown("And the following table displays the values of the decision
↪variables  $Y_i$ "))
display(y_df)
display(Markdown("Note that  $Y_i$  is only 1 if the warehouse is used to deliver to
↪any region"))
```

Optimal

Total cost is **235370.00 €**

##Decision Variables##

The following table displays the values of the decision variables  $X_{ij}$ . The orientation is the same as in the table that provided the data in the first place

	1	2	3	4	5
1	0	0	0	0	1

(continues on next page)

(continued from previous page)

2	0	0	0	0	1
3	1	0	0	0	0
4	0	0	0	0	1
5	1	0	0	0	0
6	0	0	1	0	0
7	0	0	0	0	1
8	0	0	1	0	0
9	0	0	1	0	0
10	0	1	0	0	0

And the following table displays the values of the decision variables  $Y_i$

	1	2	3	4	5
$Y$	1	1	1	0	1

Note that  $Y_i$  is only 1 if the warehouse is used to deliver to any region

### 3.3.6 Distribution of pharma

Try me



#### Problem Definition

The US Department of Energy conducts a series of experiments in two research facilities located in Hawkins (Atlanta) and Los Angeles. The department needs to supply these facilities with pharma products that are processed in two factories in Portland and Flint. The delivery costs are summarised in the following table:

Production plants	Research Facilities	Delivery costs
Portland	Los Angeles	€30
Portland	Hawkins	€40
Flint	Los Angeles	€60
Flint	Hawkins	€50

The supply and demand data is provided in the table below:

Supply	Available units	Demand	Required units
Portland	200	Los Angeles	300
Flint	600	Hawkins	400

**Implement an MILP that determines how to supply the two research facilities at minimum cost.**

### Problem Model

$X_{11}$  = Units delivered from Portland to Los Angeles.

$X_{12}$  = Units delivered from Portland to Hawkins.

$X_{21}$  = Units delivered from Flint to Los Angeles.

$X_{22}$  = Units delivered from Flint to Hawkins. Objective function:

Minimize  $z = 30X_{11} + 40X_{12} + 60X_{21} + 50X_{22}$

Constraints:

$X_{11} + X_{12} \leq 200$  Supply from Portland

$X_{21} + X_{22} \leq 600$  Supply from Flint

$X_{11} + X_{21} \geq 300$  Demand in Los Angeles

$X_{12} + X_{22} \geq 400$  Demand in Hawkins

All variables are Integer and greater or equal than zero.

### Solution in Python

The following scripts implement the problem model and solve it using Python.

#### Requirements

Install the following requirements first

```
[ ]: !pip install pandas
!pip install pulp
!pip install IPython
```

```
[7]: import pandas as pd
import pulp
import numpy as np
from IPython.display import display, Markdown
```

```
[14]: # Instantiate our problem class
model = pulp.LpProblem("Cost minimising problem", pulp.LpMinimize)
```

```
[15]: # Construct our decision variable lists
supply = ['Portland', 'Flint']
demand = ['Los Angeles', 'Hawkins']

variables = pulp.LpVariable.dicts("units ",
                                  ((i, j) for i in supply for j in demand),
                                  lowBound=0,
                                  cat='Integer')

#coefficients
coefficients = {}
coefficients['Portland', 'Los Angeles'] = 30
coefficients['Portland', 'Hawkins'] = 40
coefficients['Flint', 'Los Angeles'] = 60
coefficients['Flint', 'Hawkins'] = 50
```

(continues on next page)



(continued from previous page)

```
# Objective Function
model += (
    pulp.lpSum([
        coefficients[(i,j)] * variables[(i, j)]
        for i in supply for j in demand])
)

#Constraints
model += pulp.lpSum([variables['Portland', j] for j in demand]) <= 200, "Supply From_
↪Portland"
model += pulp.lpSum([variables['Flint', j] for j in demand]) <= 600, "Supply From_
↪Flint"

model += pulp.lpSum([variables[i, 'Los Angeles'] for i in supply]) >= 300, "Demand in_
↪Los Angeles"
model += pulp.lpSum([variables[i, 'Hawkins'] for i in supply]) >= 400, "Demand in_
↪Hawkins"
```

```
[16]: # Solve our problem
model.solve()
pulp.LpStatus[model.status]
```

```
[16]: 'Optimal'
```

```
[17]: total_cost = pulp.value(model.objective)
display(Markdown("Total cost is %0.2f €"%total_cost))

display(Markdown("The following table shows the decision variables: "))
var_df = pd.DataFrame.from_dict(variables, orient="index",
                                columns = ["Variables"], dtype=object)

var_df["Solution"] = var_df["Variables"].apply(lambda item: item.varValue)
display(var_df)
```

Total cost is 32000.00 €

The following table shows the decision variables:

	Variables	Solution (GRB)	\
(Portland, Los Angeles)	units__('Portland',_'Los_Angeles')	200.00	
(Portland, Hawkins)	units__('Portland',_'Hawkins')	0.00	
(Flint, Los Angeles)	units__('Flint',_'Los_Angeles')	100.00	
(Flint, Hawkins)	units__('Flint',_'Hawkins')	400.00	
	Reduced cost (GRB)	Objective Coefficient (GRB)	\
(Portland, Los Angeles)	0.00	30.00	
(Portland, Hawkins)	20.00	40.00	
(Flint, Los Angeles)	0.00	60.00	
(Flint, Hawkins)	0.00	50.00	
	Objective Lower bound (GRB)	\	
(Portland, Los Angeles)	-Inf		
(Portland, Hawkins)	20.00		
(Flint, Los Angeles)	40.00		
(Flint, Hawkins)	0.00		

(continues on next page)

(continued from previous page)

Objective Upper bound (GRB)	
(Portland, Los Angeles)	50.00
(Portland, Hawkins)	Inf
(Flint, Los Angeles)	Inf
(Flint, Hawkins)	70.00

The following table shows the constraints:

	Constraint	Right Hand Side	Shadow Price	Slack	Min RHS	Max RHS
0	Supply_From_Portland	200.00	-30.00	0.00	100.00	300.00
1	Supply_From_Flint	600.00	0.00	100.00	500.00	Inf
2	Demand_in_Los_Angeles	300.00	60.00	0.00	200.00	400.00
3	Demand_in_Hawkins	400.00	50.00	0.00	-0.00	500.00

[ ]:

### 3.3.7 Transport Planning

A firm must transport machines from production plants A, B and C to warehouses X, Y and Z. Five machines are required in X, 4 in Y and 3 in Z, whereas 8 machines are available in A, 5 in B and 3 in C. The transport costs (in euros) between sites are provided in the table below.

Plant/Warehouse	X	Y	Z
A	50	60	30
B	60	40	20
C	40	70	30

a) Formulate an integer linear programming model that minimizes transport costs. This is a simple transportation problem, where the objective is to minimize the transportation costs. Let us note the transportation costs from plant  $i$  to warehouse  $j$  as  $c_{ij}$ , our model is:

#### Indices

- $i$  Production plants,  $i \in [A, B, C]$
- $j$  warehouses,  $j \in [X, Y, Z]$

#### Decision variables

- $x_{ij}$  (Integer) Number of machines to transport from production plant  $i$  to warehouse  $j$

#### Objective function

$$z = \sum_i \sum_j c_{ij} * x_{ij}$$

#### Constraints Demand constraint

$$\sum_i x_{ij} \geq d_j \quad \forall j$$

b) Assume that the cost of transporting a machine from plant B increases by €10 for all the machines as of the third one; that is, the 4th, the 5th, etc. Reformulate the model in Section a) by considering this assumption. In this case, we need to define a new decision variable to factor in the extra costs incurred if we source more than three machines from plant B. Let us note these new decision variables as  $x^*_{Bj}$  and represents the machines that are sourced from factory B from the third machine. Let us also note as  $c^*_{Bj}$  the additional costs incurred from the third machine:

Plant/Warehouse	X	Y	Z
B	70	50	30

Now, our problem model becomes:

**Decision variables**

- $x_{ij}$  (Integer) Number of machines to transport from production plant  $i$  to warehouse  $j$
- $x_{*Bj}$  (Integer) Number of machines to transport from production plant B to warehouse  $j$  from the third one
- $Y$  (Binary) Used to determine whether the number of machines sourced from B is greater than 3.

**Objective function**

$$z = \sum_i \sum_j c_{ij} * x_{ij} + \sum_j c_{*Bj} * x_{*Bj}$$

**Constraints Demand constraint**

$$\sum_i x_{ij} + x_{*Bj} \geq d_j \quad \forall j$$

Logical constraints  $\sum_j x_{Bj} \leq 3$

$$\sum_j x_{Bj} - 3 + M * Y \geq 0$$

$$x_{*Bj} + M * Y \geq 0 \quad \forall j$$

$$x_{*Bj} \leq M * (1 - Y) \quad \forall j$$

Where M is a very large number. To explain the introduction of the binary decision variables and the set of constraints, note that what we want to accomplish is the following logic:

“if  $\sum_j x_{Bj} = 3$  then  $x_{*Bj} \geq 0$  else  $x_{*Bj} = 0$ ”

This is equivalent to the following:

$$((\sum_j x_{Bj} = 3) \wedge (x_{*Bj} \geq 0)) \vee ((\sum_j x_{Bj} \leq 3) \wedge (x_{*Bj} = 0))$$

where  $\wedge$  is the logical AND operator, and  $\vee$  is the logical or. That is, we want to ensure that  $x_{*Bj}$  is greater than zero when  $\sum_j x_{Bj} = 3$  or that  $x_{*Bj} = 0$  when  $\sum_j x_{Bj} \leq 3$ . We introduce the new binary decision variable  $Y$  to decide which of this two (mutually exclusive) conditions is true. Then, we multiply the binary decision variable by a very large number to render the constraints of the other condition irrelevant.

We introduce the constraints:

$$\sum_j x_{Bj} - 3 + M * Y \geq 0$$

$$x_{*Bj} + M * Y \geq 0 \quad \forall j$$

These constraints correspond to the left side of the  $\vee$  operator.

Similarly, to account for the right hand side, we introduce the constraints:

$$\sum_j x_{Bj} - 3 \leq M * (1 - Y)$$

$$x_{*Bj} \leq M * (1 - Y) \quad \forall j$$

Note that, if  $Y = 0$ , the logical constraints become:

$$\sum_j x_{Bj} \leq 3$$

$$\sum_j x_{Bj} - 3 \geq 0 \text{ (and as per the previous constraint, the sum can only be equal to 3).}$$

$$x_{*Bj} \geq 0 \quad \forall j$$

$$\sum_j x_{Bj} - 3 \leq M \text{ (irrelevant because M is a really large number)}$$

$x_{Bj} \leq M \quad \forall j$  (irrelevant because  $M$  is a really large number)

Whereas if  $Y = 1$ , the logical constraints become:

$$\sum_j x_{Bj} \leq 3$$

$$\sum_j x_{Bj} - 3 + M \geq 0 \text{ (irrelevant because } M \text{ is a really large number)}$$

$$x_{Bj} + M \geq 0 \quad \forall j \text{ (irrelevant because } M \text{ is a really large number)}$$

$$\sum_j x_{Bj} - 3 \leq 0 \text{ (this is the same as the previous constraint, so also irrelevant)}$$

$$x_{Bj} \leq 0 \quad \forall j \text{ (and since they cannot be negative, they must be equal to zero)}$$

Since the constraint  $\sum_j x_{Bj} - 3 \leq M * (1 - Y)$  is irrelevant when  $Y=0$  and when  $Y=1$ , it is removed, and we get the set of constraints in the solution above.

### Solution in Python

The following script solves the problem using Python. **### Requirements** First, install the requirements:

```
[ ]: !pip install pulp
      !pip install pandas
      !pip install IPython

[ ]: import pulp
      import pandas as pd
      #And we will use numpy to perform array operations
      import numpy as np
      #We will use display and Markdown to format the output of code cells as Markdown
      from IPython.display import display, Markdown

      # Warehouses and dictionaries
      plants = ('A', 'B', 'C')
      warehouses = ('X', 'Y', 'Z')
      #Transportation costs
      transportation_costs = [[50, 60, 30], [60, 40, 20], [40, 70, 30]]

      # Instantiate model
      model = pulp.LpProblem("Transport Planning", pulp.LpMinimize)

      # Demand
      demand = [5, 4, 3]

      # Capacities
      capacities = [8, 5, 3]

      variables = pulp.LpVariable.dicts("x",
                                       [(i, j) for i in plants for j in warehouses],
                                       lowBound=0,
                                       cat='Integer')

      model += (
          pulp.lpSum([
              transportation_costs[i][j] * variables[(plants[i], warehouses[j])]
              for i in range(len(plants)) for j in range(len(warehouses))]
          ), "Transportation Cost"

      # Capacity constraints
```

(continues on next page)

(continued from previous page)

```

for i in range(len(plants)):
    model += pulp.lpSum([
        variables[(plants[i], warehouses[j])]
        for j in range(len(warehouses))] <= capacities[i], plants[i]

# Demand
for j in range(len(warehouses)):
    model += pulp.lpSum([
        variables[(plants[i], warehouses[j])]
        for i in range(len(plants))] >= demand[j], warehouses[j]

```

```

[9]: import pulp
import pandas as pd
#And we will use numpy to perform array operations
import numpy as np
#We will use display and Markdown to format the output of code cells as Markdown
from IPython.display import display, Markdown

# Warehouses and dictionaries
plants = ('A', 'B', 'C')
warehouses = ('X', 'Y', 'Z')
#Transportation costs
transportation_costs = [[50, 60, 30], [60, 40, 20], [40, 70, 30]]

# Instantiate model
model = pulp.LpProblem("Transport Planning", pulp.LpMinimize)

# Demand
demand = [5, 4, 3]

# Capacities
capacities = [8, 5, 3]

variables = pulp.LpVariable.dicts("x",
                                  [(i, j) for i in plants for j in warehouses],
                                  lowBound=0,
                                  cat='Integer')

model += (
    pulp.lpSum([
        transportation_costs[i][j] * variables[(plants[i], warehouses[j])]
        for i in range(len(plants)) for j in range(len(warehouses))]
    ), "Transportation Cost"

# Capacity constraints
for i in range(len(plants)):
    model += pulp.lpSum([
        variables[(plants[i], warehouses[j])]
        for j in range(len(warehouses))] <= capacities[i], plants[i]

# Demand
for j in range(len(warehouses)):
    model += pulp.lpSum([
        variables[(plants[i], warehouses[j])]
        for i in range(len(plants))] >= demand[j], warehouses[j]

```

```
[10]: # Solve our problem
model.solve()
print(pulp.LpStatus[model.status])

# Solution
max_z = pulp.value(model.objective)
print(max_z)
```

```
Optimal
460.0
```

```
[19]: var_df = pd.DataFrame.from_dict(variables, orient="index",
                                   columns = ["Variables"], dtype=object)

var_df["Solution"] = var_df["Variables"].apply(lambda item: item.varValue)
index = pd.MultiIndex.from_product([plants, warehouses], names=['Plants', 'Warehouses
↪'])
var_df2 = pd.DataFrame(var_df["Solution"], index=index, columns = ["Solution"])
display(var_df2.unstack())
```

		Solution		
Warehouses		X	Y	Z
Plants				
A		2.0	0.0	2.0
B		0.0	4.0	1.0
C		3.0	0.0	0.0

```
[ ]:
```

### 3.3.8 Plastic Injection Production Planning

#### Problem definition

Patrick “Eel” O’Brian’s (PEOB) Ltd. is a company that manufactures plastic components for the automotive sector. As an intern at PEOB Ltd, your first assignment is to write an Integer Programming Model to calculate the optimal production plan for the plastic injection machine. The injection machine heats plastic and injects it into a mould to get a specific shape. Each mould can thus only be used to manufacture specific component types. Your first version will take into account the storage costs and the delayed orders cost which are defined below.

**Storage costs** The storage or inventory costs represent the cost of storing the inventory levels of every component type at every planning period and are modeled as a fixed cost per unit and planning period. The storage costs are different for every component type.

**Delayed orders costs** At every planning period, you may delay part of the demand for the next planning period. Customers will apply a fixed penalty for every delayed unit at every period. The delayed costs are also different for every component type.

Your model needs to take into account the following additional data:

**Initial Inventory Levels (units):** There is an initial inventory level for every component type available at the first planning period that needs to be taken into account.

**Minimum and Maximum Inventory Levels (units):** There is a maximum and a minimum inventory level for every component type.

**Machine capacity (units):** The machine capacity represents the number of units of a given component type that the machine can produce using a given mould. If a mould cannot be used to manufacture a component type, the machine capacity is zero for that combination of component type and mould.

**Demand (Units):** The company has several confirmed orders for the following periods and therefore, our model needs to take into account the demand for every product at every planning period.

a. Write down the indexes and decision variables **Indices**

- $t$  = periods to produce  $t \in [0, \dots, T]$
- $m$  = moulds  $m \in [0, \dots, M]$
- $p$  = products  $p \in [0, \dots, P]$

**Decision Variables**

- $Y_{pt}$  = units to store for product  $p$  on period  $t$  (Integer)
- $D_{pt}$  = delayed production for product  $p$  on period  $t$  (Integer)
- $X_{pt}$  = production for product  $p$  on period  $t$  (Integer)
- $S_{mt}$  = (Binary) {1 if mould  $m$  is used on period  $t$ , 0 otherwise}

b. Write down the objective function The objective function is to minimise storage costs and delayed order costs (eur):

$$\min z = \sum_{t=0}^T \sum_{p=0}^P C_d \cdot D_{pt} + \sum_{t=0}^T \sum_{p=0}^P C_s \cdot Y_{pt} \text{ (eur)}$$

Where:

- $C_s$  represents the storage costs per unit and period (eur)
- $C_d$  represents the delayed order costs per unit and period (eur)

c. Write down the constraints

**Problem Data**

- $M_{mp}$  Machine Capacity using mould  $m$  to produce product  $p$ (units)
- $R_{pt}$  Demand for product  $p$  in period  $t$
- $Y_{p0}$  Initial inventory level of product  $p$
- $Y_{pmax}$  Maximum inventory level of product  $p$
- $Y_{pmin}$  Minimum inventory level of product  $p$

**Machine Capacity Constraint** The machine capacity constraint ensures that the model is consistent and that the machine capacity is not exceeded.

$$X_{pt} = \sum_{m=0}^M M_{mp} \cdot S_{mt} \quad \forall p, t$$

That is, the number of units produced in period  $t$  of product  $p$  is equal to the sum of the machine capacity of the machine using the mould  $m$  multiplied by the binary decision variable that determines if the mould is used at period  $t$ .

**Mould Constraint** The second constraint ensures that the model is consistent and only one mould is used in one period:

$$\sum_{m=0}^M S_{mt} = 1 \forall t$$

**Demand** We need to ensure that the demand is satisfied:

$$Y_{pt-1} + X_{pt} - Y_{pt} = R_{pt} + D_{pt-1} - D_{pt} \forall p, t \in [1, T]$$

That is, the inventory level of product  $p$  at period  $t$  (amount left for next period)  $Y_{pt}$  is equal to the inventory level of product  $p$  at period  $t-1$  (amount left from previous period)  $Y_{pt-1}$  plus the production of product  $p$  at period  $t$ , minus the inventory level of product  $p$  at period  $t$  (amount left for next period), minus the demand of product  $p$  at period  $t$ , minus the delayed production from last period  $t-1$ , plus the delayed production for the next period.

**Initial Inventory** Now, we need to take into account the initial inventory level:  $X_{p0} - Y_{p0} = R_{p0} - D_{p0} \forall p, t = 0$

That is, since we do not have any inventory left nor delayed production from period -1 (the problem starts in  $t=0$ ), we need to modify slightly the demand constraint for  $t = 0$ .

### Inventory levels

We need to ensure that the inventory levels are within the minimum and maximum levels:

$$Y_{pt} \geq Y_{minp} \forall p, t$$

$$Y_{pt} \leq Y_{maxp} \forall p, t$$

You have successfully validated your model, and it is already providing valuable information for the company. Now the company would like to extend the model to take into account as well the set-up costs:

**Set up costs** As explained above, the injection machine uses different moulds to manufacture different parts. Each time that a mould is changed to make a different component, the operators need to set up the machine. The company estimates that this setup cost only depends on the mould that is used and is only applied when a product is changed.

**d.** Modify the model to take into account the set-up costs.

We introduce a new decision variable to model the sequence of mould changes and simplify the model:

- $Su_{mt}$  Mould setup sequence (Binary) 1 if mould  $m$  is changed in period  $t$ , 0 otherwise.
- $C_m$  Setup costs of mould  $m$  (Eur).

Now, the objective function becomes:

$$\min z = \sum_{t=0}^T \sum_{m=0}^M C_m \cdot Su_{mt} + \sum_{t=0}^T \sum_{p=0}^P C_s \cdot D_{pt} + \sum_{t=0}^T \sum_{p=0}^P C_d \cdot Y_{pt} \text{ (Eur)}$$

We also need to introduce the following constraints for the new decision variable:

$$Su_{m0} = S_{m0} \forall m$$

Meaning that we take into account the costs of the initial setup of the machine.

$$Su_{mt} \geq S_{mt} - S_{mt-1}$$

Meaning that we only take into account in the costs mould changes (that is, when the mould used is different for two consecutive periods).

## 3.3.9 Machinery Transport Planning

Try me



### Problem Definition

A firm must transport machines from production plants A, B and C to warehouses X, Y and Z. Five machines are required in X, four in Y and three in Z and there are eight machines available in A, five in B and three in C. **a)** Contemplate and solve a Graph Theory model to determine the maximal flow of the machines that can be transported and the run of this flow. **b)** By assuming a mean cost of €45 per transported machine, what would the total cost be of transporting the machines obtained in the former section?



## Requirements

Run the following script to install the packages used in the exercise:

```
[ ]: !pip install networkx
      !pip install matplotlib
```

## Solution

We are going to use [networkx] (<https://networkx.github.io/>) to draw the network and find the maximal flow at a minimum cost between two nodes. We are going to create a graph to model the transport process, the network will comprise the Source node (node S), production plant nodes (nodes A, B, and C), warehouse nodes (nodes X, Y, and Z), and a sink node (node T). Edges are going to have two attributes:

- $b_i$ : maximum capacity (number of machines) of each production plant or warehouse
- $c_i$ : cost, cost for the company to move one machine (€).

The flow of edges going from source node S to each production plant represent the available machines at each production plant. These edges have a capacity equal to machines available at each production plant, and a cost of 0; The flow of edges going from each production plant to each warehouse represents the machines transported by the company from each production plant to each warehouse. The edges will have a maximum capacity equal to the machines available at each production plan, and a cost of 45€. Finally, flow in edges from the warehouse nodes to the sink node represent the total number of machines in each warehouse. These edges will have a maximum capacity equal to the demand of each warehouse, a company cost of 0.

```
[3]: # Import the libraries that we are going to use, Network x to import the
import networkx as nx
import matplotlib.pyplot as plt

# Create an empty directed graph structure:
G = nx.DiGraph()

# Add edges and define two attributes, production and costs:
G.add_edges_from([("S", "A", {"capacity": 8, "cost": 0}),
                  ("S", "B", {"capacity": 5, "cost": 0}),
                  ("S", "C", {"capacity": 3, "cost": 0}),
                  ("A", "X", {"capacity": 8, "cost": 45}),
                  ("A", "Y", {"capacity": 8, "cost": 45}),
                  ("A", "Z", {"capacity": 8, "cost": 45}),
                  ("B", "X", {"capacity": 5, "cost": 45}),
                  ("B", "Y", {"capacity": 5, "cost": 45}),
                  ("B", "Z", {"capacity": 5, "cost": 45}),
                  ("C", "X", {"capacity": 3, "cost": 45}),
                  ("C", "Y", {"capacity": 3, "cost": 45}),
                  ("C", "Z", {"capacity": 3, "cost": 45}),
                  ("X", "T", {"capacity": 5, "cost": 0}),
                  ("Y", "T", {"capacity": 4, "cost": 0}),
                  ("Z", "T", {"capacity": 3, "cost": 0})])
```

(continues on next page)

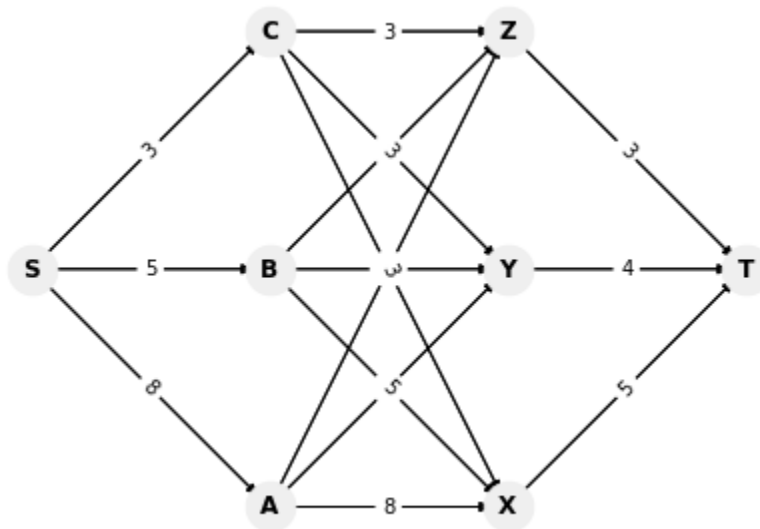
(continued from previous page)

```

# Draw the directed graph
pos = {"S": (0, 1),
       "A": (1, 0),
       "B": (1, 1),
       "C": (1, 2),
       "X": (2, 0),
       "Y": (2, 1),
       "Z": (2, 2),
       "T": (3, 1)
      }

nx.draw(G, pos)
nx.draw_networkx_edges(G, pos, weight=2)
nx.draw_networkx_nodes(G, pos, node_size=600, node_color='#efefef')
nx.draw_networkx_labels(G, pos, font_weight='bold' )

cap = nx.get_edge_attributes(G, 'capacity')
nx.draw_networkx_edge_labels(G, pos, edge_labels=cap)
plt.show()
    
```



```

[4]: max_flow, flow= nx.maximum_flow(G, "S", "T", capacity='capacity')
costs = nx.get_edge_attributes(G, 'cost')
cost=0
print("maximum flow value:", max_flow)
for k,v in flow.items():
    for k2 in v.keys():
        cost+=v[k2]*costs[(k,k2)]
        print("flow from ", k, " to ", k2, ": ", v[k2], "; Cost: ", v[k2]*costs[(k,
↪k2)], "total cost: ", cost)

print ("Total cost is: ", cost)

maximum flow value: 12
    
```

(continues on next page)

(continued from previous page)

```

flow from S to A : 4 ; Cost: 0 total cost: 0
flow from S to B : 5 ; Cost: 0 total cost: 0
flow from S to C : 3 ; Cost: 0 total cost: 0
flow from A to X : 0 ; Cost: 0 total cost: 0
flow from A to Y : 1 ; Cost: 45 total cost: 45
flow from A to Z : 3 ; Cost: 135 total cost: 180
flow from B to X : 2 ; Cost: 90 total cost: 270
flow from B to Y : 3 ; Cost: 135 total cost: 405
flow from B to Z : 0 ; Cost: 0 total cost: 405
flow from C to X : 3 ; Cost: 135 total cost: 540
flow from C to Y : 0 ; Cost: 0 total cost: 540
flow from C to Z : 0 ; Cost: 0 total cost: 540
flow from X to T : 5 ; Cost: 0 total cost: 540
flow from Y to T : 4 ; Cost: 0 total cost: 540
flow from Z to T : 3 ; Cost: 0 total cost: 540
Total cost is: 540

```

```

[ ]: import pandas as pd
      from IPython.display import display, Markdown

      flow_df = pd.DataFrame.from_dict(flow, orient='index')
      display(flow_df)

```

```

[ ]: optimal_flow = {}
      for k,v in flow.items():
          for k2 in v.keys():
              optimal_flow[k,k2] = flow[k][k2]

      nx.draw(G, pos)
      nx.draw_networkx_edges(G, pos, weight=2)
      nx.draw_networkx_nodes(G, pos, node_size=600, node_color='#efefef')
      nx.draw_networkx_labels(G, pos, font_weight='bold' )
      nx.draw_networkx_edge_labels(G, pos, edge_labels=optimal_flow)

```

```
[ ]:
```

### 3.3.10 Overbooking on Airlines

#### Problem Definition

A low-cost airline, ACP (Flying High), operates four daily flights from Valencia to London at 10:00, 12:00, 14:00 and 16:00 hours. The first two flights hold 100 passengers, and the last two can fly up to 150 passengers each. If overbooking occurs, which implies having sold more seats than the plane actually has, the airline can place a passenger on a later flight. Evidently each delayed traveller is compensated by being paid €200 plus €20 for each hour delayed. The firm places delayed travellers on their own operating flights, or on one of the flights from other airlines departing at 20:00 hours, which always have available seats (a capacity of 999 passengers is always considered) at no extra charge. Let us assume that at the beginning of the day we know that ACP has sold 110, 160, 100 and 100 seats on its four daily flights, respectively. Model this problem as a maximal flow model at a minimum cost in order to minimise the airline's total overbooking cost.

## Solution

We are going to use `networkx` to draw the network and find the maximal flow at a minimum cost between two nodes. We are going to create a graph to model the selling process as a system with different states: source state (containing node S), flights sold state (comprised of nodes 10s, 12s, 14s, 16s), flights used state (comprised of nodes 10u, 12u, 14u, 16u, and 20u), and a sink state (containing node T). Edges are going to have two attributes:

- $a_i$ : maximum capacity in number of flight tickets sold or used
- $c_i$ : cost, cost for the company to move a passenger from one adjacent node to the other

The flow of edges going from source node S to each flight sold state node flight represent the seats sold (edge S,10s represents the flights sold for flight at 10:00h, edge S,12s for the flight at 12:00s, etc.). These flights have a capacity equal to the number of tickets sold (110, 160, 100 and 140 respectively), and a cost of 0; The flow of edges going from each flight sold state to each flight used state represents how the company arranges passengers in the different flights. The edges will have a maximum capacity equal to the maximum number of passengers in each flight (100, 100, 150, 150 and 999) respectively, and a cost equal to the overbooking cost for each delayed flight (that is, zero for no delay, 240 for two hours late, 280 for hours late, and so on).

Finally, flow in edges from the flights used state nodes to the sink node represent the total number of seats used in each plane. These edges will have a maximum capacity equal to the maximum capacity of each plane and a company cost of 0.

```
[8]: import pandas as pd
import networkx as nx
from IPython.display import display, Markdown

# Create an empty directed graph structure:
G = nx.DiGraph()

# Add edges and define two attributes, production and costs:
G.add_edges_from([("S", "10s", {"capacity": 110, "cost": 0}),
                  ("S", "12s", {"capacity": 160, "cost": 0}),
                  ("S", "14s", {"capacity": 100, "cost": 0}),
                  ("S", "16s", {"capacity": 140, "cost": 0}),
                  ("10s", "10u", {"capacity": 100, "cost": 0}),
                  ("10s", "12u", {"capacity": 100, "cost": 240}),
                  ("10s", "14u", {"capacity": 150, "cost": 280}),
                  ("10s", "16u", {"capacity": 150, "cost": 320}),
                  ("10s", "20u", {"capacity": 999, "cost": 400}),
                  ("12s", "12u", {"capacity": 100, "cost": 0}),
                  ("12s", "14u", {"capacity": 150, "cost": 240}),
                  ("12s", "16u", {"capacity": 150, "cost": 280}),
                  ("12s", "20u", {"capacity": 999, "cost": 360}),
                  ("14s", "14u", {"capacity": 150, "cost": 0}),
                  ("14s", "16u", {"capacity": 150, "cost": 240}),
                  ("14s", "20u", {"capacity": 999, "cost": 320}),
                  ("16s", "16u", {"capacity": 150, "cost": 0}),
                  ("16s", "20u", {"capacity": 999, "cost": 280}),
                  ("10u", "T", {"capacity": 100, "cost": 0}),
                  ("12u", "T", {"capacity": 100, "cost": 0}),
                  ("14u", "T", {"capacity": 150, "cost": 0}),
                  ("16u", "T", {"capacity": 150, "cost": 0}),
                  ("20u", "T", {"capacity": 999, "cost": 0})])

# Draw the directed graph
pos = {"S": (0, 1),
       "10s": (1, 0),
```

(continues on next page)

(continued from previous page)

```

    "12s": (1, 0.67),
    "14s": (1, 1.33),
    "16s": (1, 2),
    "10u": (2, 0),
    "12u": (2, 0.5),
    "14u": (2, 1),
    "16u": (2, 1.5),
    "20u": (2, 2),
    "T": (3, 1)
}

```

```

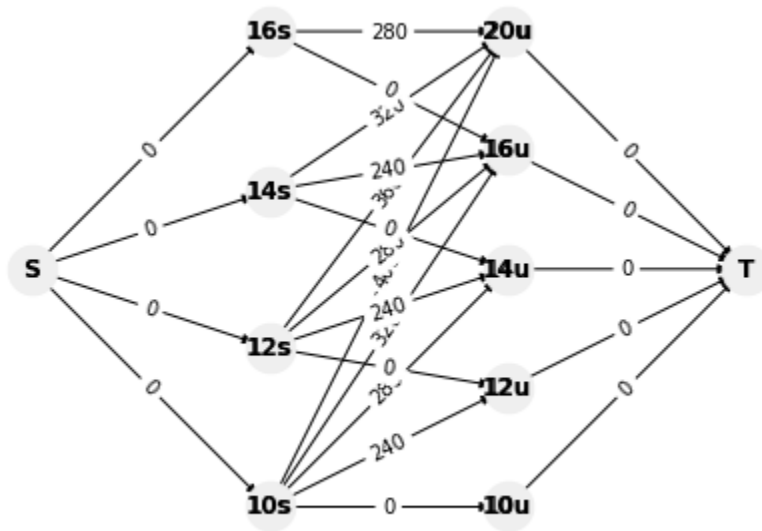
nx.draw(G, pos)
nx.draw_networkx_labels(G, pos)
nx.draw_networkx_nodes(G, pos, node_size=600, node_color='#efefef')
nx.draw_networkx_labels(G, pos, font_weight='bold' )
c_label = nx.get_edge_attributes(G, 'cost')
nx.draw_networkx_edge_labels(G, pos, edge_labels=c_label)

```

```

[8]: {('S', '10s'): Text(0.5, 0.5, '0'),
      ('S', '12s'): Text(0.5, 0.835, '0'),
      ('S', '14s'): Text(0.5, 1.165, '0'),
      ('S', '16s'): Text(0.5, 1.5, '0'),
      ('10s', '10u'): Text(1.5, 0.0, '0'),
      ('10s', '12u'): Text(1.5, 0.25, '240'),
      ('10s', '14u'): Text(1.5, 0.5, '280'),
      ('10s', '16u'): Text(1.5, 0.75, '320'),
      ('10s', '20u'): Text(1.5, 1.0, '400'),
      ('12s', '12u'): Text(1.5, 0.585, '0'),
      ('12s', '14u'): Text(1.5, 0.835, '240'),
      ('12s', '16u'): Text(1.5, 1.085, '280'),
      ('12s', '20u'): Text(1.5, 1.335, '360'),
      ('14s', '14u'): Text(1.5, 1.165, '0'),
      ('14s', '16u'): Text(1.5, 1.415, '240'),
      ('14s', '20u'): Text(1.5, 1.665, '320'),
      ('16s', '16u'): Text(1.5, 1.75, '0'),
      ('16s', '20u'): Text(1.5, 2.0, '280'),
      ('10u', 'T'): Text(2.5, 0.5, '0'),
      ('12u', 'T'): Text(2.5, 0.75, '0'),
      ('14u', 'T'): Text(2.5, 1.0, '0'),
      ('16u', 'T'): Text(2.5, 1.25, '0'),
      ('20u', 'T'): Text(2.5, 1.5, '0')}

```



```
[4]: edges_df = nx.to_pandas_edgelist(G);
display(edges_df)
```

	source	target	capacity	cost
0	S	10s	110	0
1	S	12s	160	0
2	S	14s	100	0
3	S	16s	140	0
4	10s	10u	100	0
5	10s	12u	100	240
6	10s	14u	150	280
7	10s	16u	150	320
8	10s	20u	999	400
9	12s	12u	100	0
10	12s	14u	150	240
11	12s	16u	150	280
12	12s	20u	999	360
13	14s	14u	150	0
14	14s	16u	150	240
15	14s	20u	999	320
16	16s	16u	150	0
17	16s	20u	999	280
18	10u	T	100	0
19	12u	T	100	0
20	14u	T	150	0
21	16u	T	150	0
22	20u	T	999	0

```
[13]: max_flow, flow= nx.maximum_flow(G, "S", "T", capacity='capacity')
costs = nx.get_edge_attributes(G, 'cost')
print("maximum flow value:", max_flow)
for k,v in flow.items():
    for k2 in v.keys():
        cost+=v[k2]*costs[(k,k2)]
```

(continues on next page)

(continued from previous page)

```

        print("flow from ", k, " to ", k2, ": ", v[k2], "; Cost: ", v[k2]*costs[(k,
        ↪k2)])

print ("Total cost is: ", cost)

```

```

maximum flow value: 510
flow from S to 10s : 110 ; Cost: 0
flow from S to 12s : 160 ; Cost: 0
flow from S to 14s : 100 ; Cost: 0
flow from S to 16s : 140 ; Cost: 0
flow from 10s to 10u : 100 ; Cost: 0
flow from 10s to 12u : 0 ; Cost: 0
flow from 10s to 14u : 0 ; Cost: 0
flow from 10s to 16u : 0 ; Cost: 0
flow from 10s to 20u : 10 ; Cost: 4000
flow from 12s to 12u : 100 ; Cost: 0
flow from 12s to 14u : 50 ; Cost: 12000
flow from 12s to 16u : 10 ; Cost: 2800
flow from 12s to 20u : 0 ; Cost: 0
flow from 14s to 14u : 100 ; Cost: 0
flow from 14s to 16u : 0 ; Cost: 0
flow from 14s to 20u : 0 ; Cost: 0
flow from 16s to 16u : 140 ; Cost: 0
flow from 16s to 20u : 0 ; Cost: 0
flow from 10u to T : 100 ; Cost: 0
flow from 12u to T : 100 ; Cost: 0
flow from 14u to T : 150 ; Cost: 0
flow from 16u to T : 150 ; Cost: 0
flow from 20u to T : 10 ; Cost: 0
Total cost is: 56400

```

```

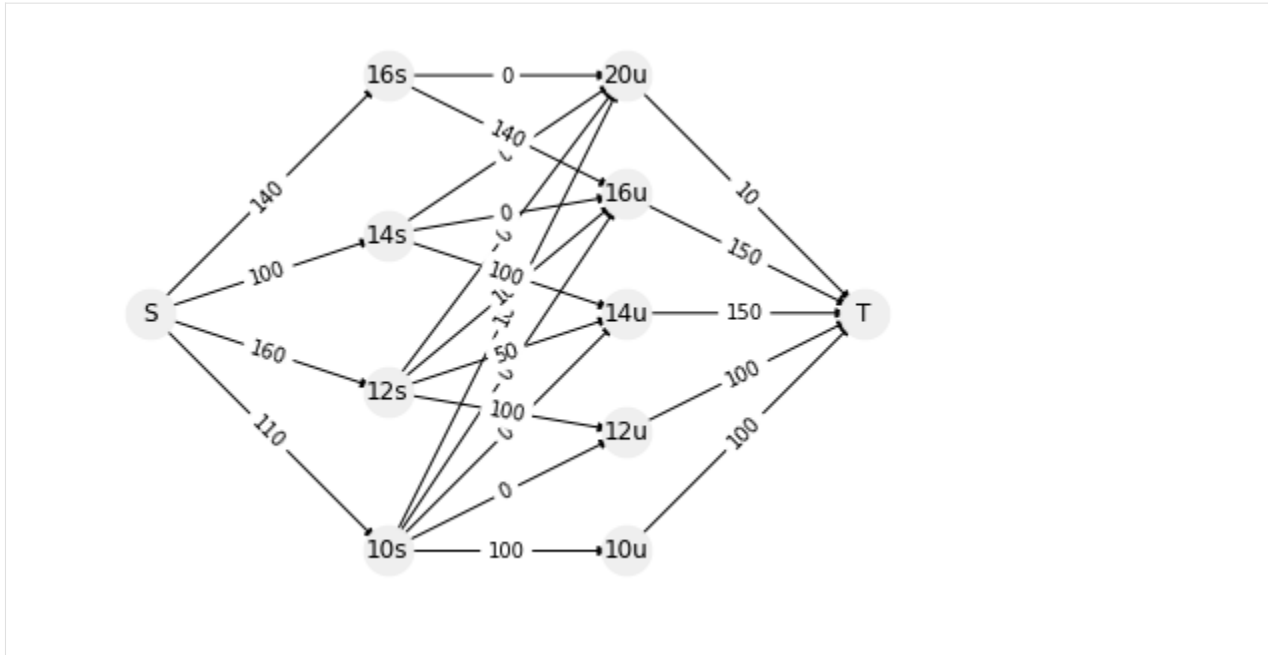
[16]: optimal_flow = {}
      for i in G.nodes():
          for j in flow[i].keys():
              optimal_flow[i,j] = flow[i][j]

      nx.draw(G, pos)
      nx.draw_networkx_labels(G, pos)
      nx.draw_networkx_nodes(G, pos, node_size=600, node_color='#efefef')
      nx.draw_networkx_edge_labels(G, pos, edge_labels=optimal_flow)

      print(flow)

      {'S': {'10s': 110, '12s': 160, '14s': 100, '16s': 140}, '10s': {'10u': 100, '12u': 0,
      ↪ '14u': 0, '16u': 0, '20u': 10}, '12s': {'12u': 100, '14u': 50, '16u': 10, '20u': 0},
      ↪ '14s': {'14u': 100, '16u': 0, '20u': 0}, '16s': {'16u': 140, '20u': 0}, '10u': {'T':
      ↪ 100}, '12u': {'T': 100}, '14u': {'T': 150}, '16u': {'T': 150}, '20u': {'T': 10},
      ↪ 'T': {}}

```



```
[20]: flow_df = pd.DataFrame.from_dict(flow, orient='index')
display(flow_df)
```

	10s	12s	14s	16s	10u	12u	14u	16u	20u	T
10s	NaN	NaN	NaN	NaN	100.0	0.0	0.0	0.0	10.0	NaN
10u	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	100.0
12s	NaN	NaN	NaN	NaN	NaN	100.0	50.0	10.0	0.0	NaN
12u	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	100.0
14s	NaN	NaN	NaN	NaN	NaN	NaN	100.0	0.0	0.0	NaN
14u	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	150.0
16s	NaN	NaN	NaN	NaN	NaN	NaN	NaN	140.0	0.0	NaN
16u	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	150.0
20u	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	10.0
S	110.0	160.0	100.0	140.0	NaN	NaN	NaN	NaN	NaN	NaN

[ ]:

### 3.3.11 Production Planning

#### Problem Definition

PLACASA, a supplier of plastic injection parts for the automobile sector, foresees the demand of an article for the following 4 months of 100, 140, 210 and 180 units, respectively. The firm can maintain only sufficient stock to meet the demand of each month, or it can have excess stock to meet the demand of two successive months or more with a storage cost of 1.2€ per month and unit of excessive stock. PLACASA calculates that the production cost for the following months, is €15, €12, €10 and €14 per unit. A preparation cost of €200 is incurred every time a production order is placed. The firm wishes to develop a production plan that minimises the total costs of production orders, of production and of keeping an article in stock. Formulate and solve the shortest path model to find an optimum production plan.



## Solution

We are going to use [networkx] (<https://networkx.github.io/>) to draw the network and find the shortest path between two nodes. Each node in the graph is going to represent a production period (one month). Thus, there will be 5 nodes, for month 0 (current month) and the four following months. Edges (i,j) represent the decision to produce in month i all the demand up to month j. For instance, edge 01 means that we decide to manufacture in month 0 100 units, which is the demand in month 1, and 04 means that we decide to manufacture in month 0  $100+140+210+180=650$  units, which is the demand forecast for the forthcoming 4 months. To calculate the costs, we need to apply the production cost  $cp_i$ , storage cost  $cs$ , and preparation costs  $c_P$ .

$$c_{ij} = c_P + \sum_{k=i+1}^j d_k \cdot (cp_i + (k-1) \cdot cs)$$

$$c_{01} = 200 + \sum_{k=1}^1 d_k \cdot (cp_0 + (k-1) \cdot cs) = 200 + 100 \cdot (15 + 0 \cdot 1.2) = 1500$$

$$c_{02} = 200 + \sum_{k=1}^2 d_k \cdot (cp_0 + (k-1) \cdot cs) = 200 + 100 \cdot (15 + 0 \cdot 1.2) + 140 \cdot (15 + 1 \cdot 1.2) = 3968$$

...

$$c_{24} = 200 + \sum_{k=3}^4 d_k \cdot (cp_2 + (k-1) \cdot cs) = 200 + 210 \cdot (10 + 0 \cdot 1.2) + 180 \cdot (10 + 1 \cdot 1.2) = 4316$$

$$c_{34} = 200 + \sum_{k=4}^4 d_k \cdot (cp_2 + (k-1) \cdot cs) = 200 + 180 \cdot (14 + 0 \cdot 1.2) = 2720$$

The following code shows the solution in Python.

```
[1]: import pandas as pd
import networkx as nx
from IPython.display import display, Markdown

# Create an empty directed graph structure:
G = nx.DiGraph()

# Add edges and define two attributes, production and costs:
G.add_edges_from([(0,1, {'cost': 1700, 'production': 100}),
                  (0,2, {'cost': 3968, 'production': 240}),
                  (0,3, {'cost': 7622, 'production': 450}),
                  (0,4, {'cost': 10970, 'production': 630}),
                  (1,2, {'cost': 1880, 'production': 140}),
                  (1,3, {'cost': 4652, 'production': 350}),
                  (1,4, {'cost': 7028, 'production': 530}),
                  (2,3, {'cost': 2300, 'production': 210}),
                  (2,4, {'cost': 4316, 'production': 390}),
                  (3,4, {'cost': 2720, 'production': 180}),
                  ])

# Draw the directed graph
nx.draw_shell(G, with_labels=True)

# Calculate the shortest path from 0 to 4, using the attribute cost as edge weight
shortest_path = nx.shortest_path(G, 0, 4, weight="cost")

# Calculate the production per month, and the total cost using the shortest path.
↪ solution and the cost attribute
total_cost = 0
production = {}
for i in range(len(shortest_path)-1):
    total_cost+=G[shortest_path[i]][shortest_path[i+1]]['cost']
    production[i]={'production': G[shortest_path[i]][shortest_path[i+1]]['production']
    ↪}]
```

(continues on next page)

(continued from previous page)

```
#Print the solution
print("Total cost is: ", total_cost)
print("Shortest path is:", shortest_path)
print("Monthly production is: ", production)

edges_df = nx.to_pandas_edgelist(G);

production = pd.DataFrame.from_dict(production, orient='index')
display(edges_df)
display(production)
```

Total cost is: 7896  
 Shortest path is: [0, 1, 2, 4]  
 Monthly production is: {0: {'production': 100}, 1: {'production': 140}, 2: {  
 ↪ 'production': 390}}

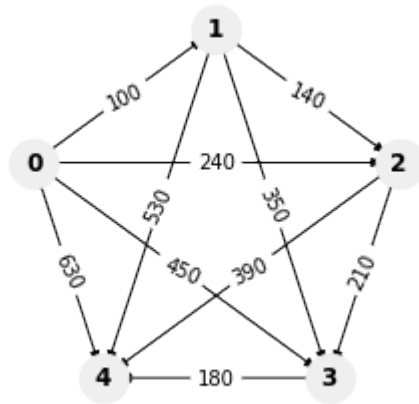
	source	target	cost	production
0	0	1	1700	100
1	0	2	3968	240
2	0	3	7622	450
3	0	4	10970	630
4	1	2	1880	140
5	1	3	4652	350
6	1	4	7028	530
7	2	3	2300	210
8	2	4	4316	390
9	3	4	2720	180

```
production
0      100
1      140
2      390
```

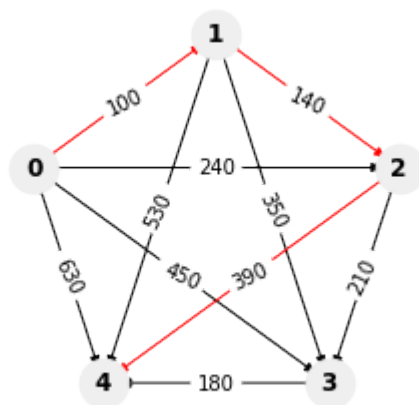
```
[3]: import matplotlib.pyplot as plt

# Draw the directed graph
pos = {0: (28.09, 0.62),
       1: (30, 2.01),
       2: (31.91, 0.62),
       3: (31.18, -1.63),
       4: (28.82, -1.63)
       }

nx.draw_networkx_edges(G, pos)
nx.draw_networkx_nodes(G, pos, node_size=600, node_color='#efefef')
nx.draw_networkx_labels(G, pos, font_weight='bold' )
c_label = nx.get_edge_attributes(G, 'production')
nx.draw_networkx_edge_labels(G, pos, edge_labels=c_label)
plt.axis('equal')
plt.axis('off')
plt.show()
```



```
[12]: # Draw the directed graph with colored edges
pos = {0: (28.09, 0.62),
      1: (30, 2.01),
      2: (31.91, 0.62),
      3: (31.18, -1.63),
      4: (28.82, -1.63)}
}
red_edges = list(zip(shortest_path,shortest_path[1:]))
edge_col = ['red' if edge in red_edges else 'black' for edge in G.edges()]
nx.draw_networkx_edges(G, pos, edge_color= edge_col)
nx.draw_networkx_nodes(G, pos, node_size=600, node_color='#efefef')
nx.draw_networkx_labels(G, pos, font_weight='bold')
c_label = nx.get_edge_attributes(G, 'production')
nx.draw_networkx_edge_labels(G, pos, edge_labels=c_label)
plt.axis('equal')
plt.axis('off')
plt.show()
```



```
[ ]:
```

### 3.3.12 Production Sequencing

#### Problem Definition

A manufacturer of the metal sector in Stafford manufactures four types of products in sequence on two machines. The Table below provides the necessary technical production details. Taking into account only the data on the production times per product and the maximum daily production capacity of each machine, and taking a maximum daily production of 100 units of each product type, consider and solve a maximal flow model in a graph to determine the maximum number of minutes during which machines 1 and 2 can operate on a production-in-sequence basis given the existing capacity constraints.

Machine	Product 1	Product 2	Product 3	Product 4	Daily Production Capacity (minutes)
1	2	3	4	2	500
2	3	2	1	2	380

#### Solution

This is an example of how graph theory can be used to solve production sequencing problems. In this case, the resource that we want to optimise is machine time. The flow in our graph is going to represent machine time and the unit is going to be minutes. We will have a source node, one node representing the first step of the sequence (machine 1), four nodes representing the different products to be manufactured, another node for the second step of the process (machine 2), and a sink node. The capacity of the edge between the source node and the Machine 1 node is equal to the machine production capacity (500), since it is the total number of minutes that we can use in that machine. The capacities of edges between the machine 1 node and the products nodes are the production times in that machine times the number of units (200, 300, 400, 200), since it is the maximum time that we can invest in each time of product in machine 1. Likewise, the edges between the product nodes and the machine 2 node are 300, 200, 100, and 200, and the capacity of the edge between the machine 2 node and the sink node is 380.

```
[14]: import pandas as pd
import networkx as nx
from IPython.display import display, Markdown

# Create an empty directed graph structure:
G = nx.DiGraph()

# Add edges and define two attributes, production and costs:
G.add_edges_from([("S", "M1", {"capacity": 500, "production": 0}),
                  ("M1", "P1", {"capacity": 200, "production": 2}),
                  ("M1", "P2", {"capacity": 300, "production": 3}),
                  ("M1", "P3", {"capacity": 400, "production": 4}),
                  ("M1", "P4", {"capacity": 200, "production": 2}),
                  ("P1", "M2", {"capacity": 300, "production": 3}),
                  ("P2", "M2", {"capacity": 200, "production": 2}),
                  ("P3", "M2", {"capacity": 100, "production": 2}),
                  ("P4", "M2", {"capacity": 200, "production": 2}),
                  ("M2", "T", {"capacity": 380, "production": 0})])

# Draw the directed graph
pos = {"S": (0, 1),
       "M1": (1, 1),
       "P1": (2, 0),
       "P2": (2, 0.67),
       "P3": (2, 1.33),
       "P4": (2, 2),
```

(continues on next page)

(continued from previous page)

```

    "M2": (3, 1),
    "T": (4, 1)
}

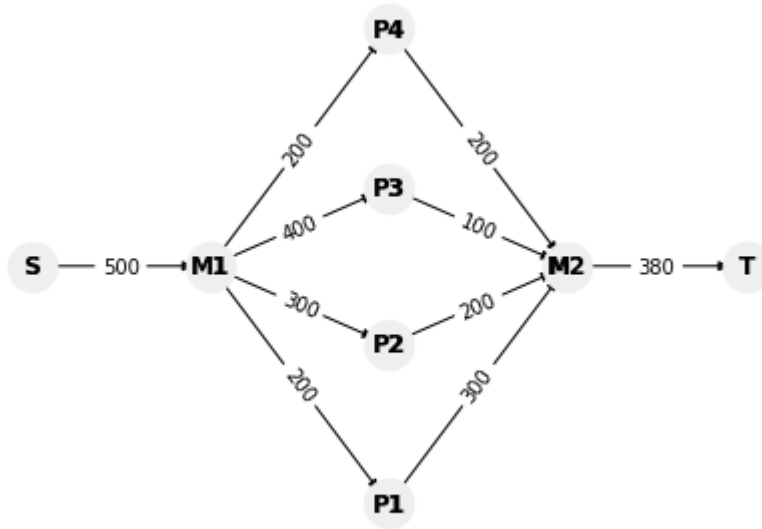
nx.draw(G, pos)
nx.draw_networkx_labels(G, pos)
nx.draw_networkx_nodes(G, pos, node_size=600, node_color='#efefef')
nx.draw_networkx_labels(G, pos, font_weight='bold' )
c_label = nx.get_edge_attributes(G, 'capacity')
nx.draw_networkx_edge_labels(G, pos, edge_labels=c_label)

```

```

[14]: {'S', 'M1'): Text(0.5,1,'500'),
('M1', 'P1'): Text(1.5,0.5,'200'),
('M1', 'P2'): Text(1.5,0.835,'300'),
('M1', 'P3'): Text(1.5,1.165,'400'),
('M1', 'P4'): Text(1.5,1.5,'200'),
('P1', 'M2'): Text(2.5,0.5,'300'),
('P2', 'M2'): Text(2.5,0.835,'200'),
('P3', 'M2'): Text(2.5,1.165,'100'),
('P4', 'M2'): Text(2.5,1.5,'200'),
('M2', 'T'): Text(3.5,1,'380')}

```



```

[15]: value, flow = nx.maximum_flow(G, "S", "T", capacity='capacity' )
print(value)
display(flow)

```

380

```

{'S': {'M1': 380},
'M1': {'P1': 80, 'P2': 200, 'P3': 100, 'P4': 0},
'P1': {'M2': 80},
'P2': {'M2': 200},
'P3': {'M2': 100},
'P4': {'M2': 0},

```

(continues on next page)

(continued from previous page)

```
'M2': {'T': 380},
'T': {}
```

[ ]:

### 3.3.13 Planet Express Deliveries

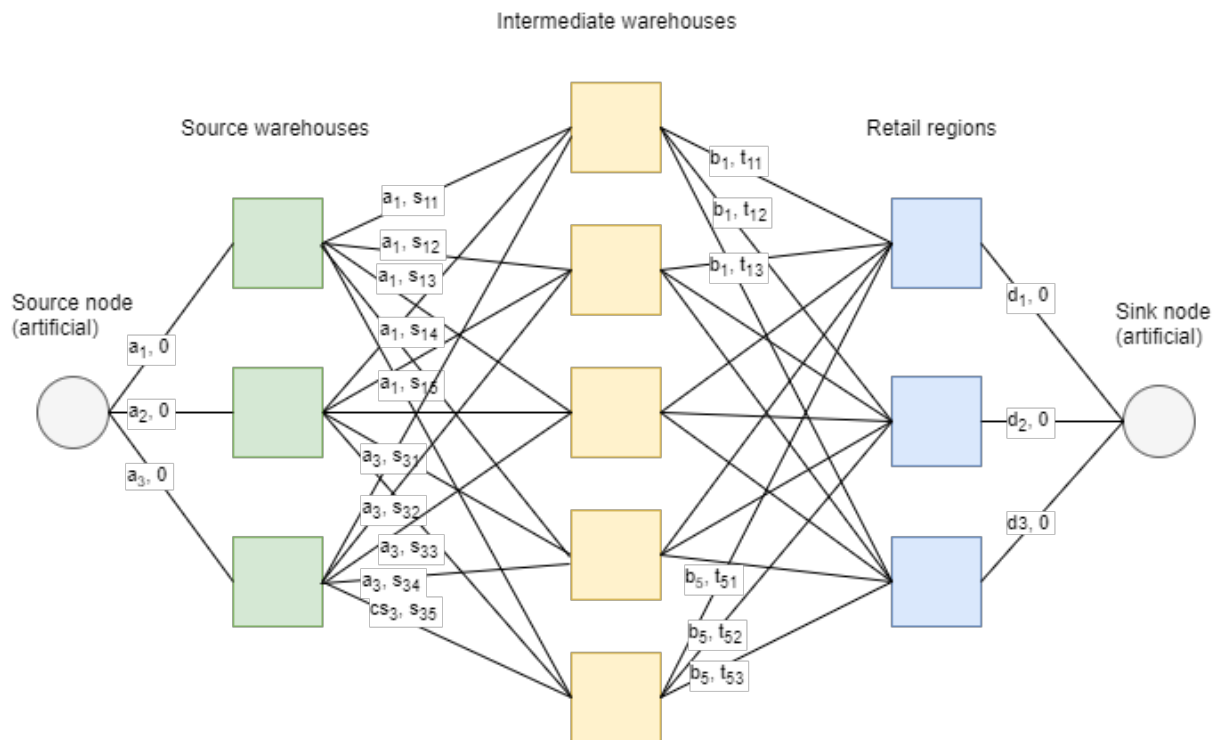
#### Problem definition

Planet Express deliveries wants to optimize their transportation network from a set of source warehouses to a set of destinations. In particular, they want to minimize the transportation costs through a set of intermediate warehouses, so that items are transported from the source warehouses, through the intermediate warehouses, to the destination regions.

Let us first consider that transportation costs per item from the source warehouse  $i$  to intermediate warehouse  $j$  as  $s_{ij}$  and that the transportation costs per item from intermediate warehouse  $j$  to destination region  $k$  is  $t_{jk}$ . Let us note the capacity of source warehouses as  $a_i$  items, the capacity of intermediate warehouses  $j$  as  $b_j$  items, and the demand for region  $k$  as  $d_k$  items. Let us also consider that  $m$  is the number of source warehouses,  $n$  is the number of intermediate warehouses and  $p$  is the number of destination regions.

**a** Consider that  $m = p = 3$  and  $n = 5$  and draw a network to solve the transportation problem as a maximal flow at minimum cost problem.

The network consists of a Source Node, 3 nodes that represent the source warehouses, 5 nodes that represent the intermediate warehouses, and 3 nodes that represent the destination regions. The edges must be labeled with the capacities and the costs. In the figure below, capacities and costs of every edge are labeled as “capacity, cost”, e.g. an edge with capacity  $a_{11}$  and cost  $s_{11}$  is labeled as  $a_{11}, s_{11}$ . For the sake of clarity, only the first and last edges are labeled at every stage:



**b** Model the problem as a Mixed Integer Programming (MIP), not taking into account the values of  $m, p,$  and  $n$  defined in the previous section

**Decision variables**

- $x_{ij}$ : (Integer) Number of units to transport from source warehouse  $i$  to intermediate warehouse  $j$
- $y_{jk}$ : (Integer) Number of units to transport from intermediate warehouse  $j$  to destination region  $k$

The linear model is then:

$$\min z = \sum_i \sum_j s_{ij} * x_{ij} + \sum_j \sum_k t_{jk} * y_{jk}$$

*s.t.*

Source warehouse capacity constraints:

$$\sum_j x_{ij} \leq a_i \quad \forall i$$

Intermediate warehouse capacity constraints:

$$\sum_k y_{jk} \leq b_j \quad \forall j$$

Demand constraints:

$$\sum_j y_{jk} \geq d_k \quad \forall k$$

Flow constraints:

$$\sum_i x_{ij} = \sum_k y_{jk} \quad \forall j$$

**c** Planet Express is now considering the use of warehouses from an external provider. The provider of the service has a total of  $n_e$  external warehouses. They offer a fixed cost  $F_l$  when an external warehouse  $l$  is used, plus a cost per item received and processed in warehouse  $l$ ,  $r_l$ . External warehouse  $l$  has a capacity of  $c_l$  items. Modify your model to incorporate this offer into your distribution network.

The following decision variables are introduced in the model:

- $Y_l$ : (Binary) Indicates if external warehouse  $l$  is used
- $x'_{il}$ : (Integer) Number of units to transport from source warehouse  $i$  to external intermediate warehouse  $l$
- $y'_{lk}$ : (Integer) Number of units to transport from external intermediate warehouse  $l$  to destination region  $k$

The model is now:

$$\min z = \sum_i \sum_j s_{ij} * x_{ij} + \sum_j \sum_k t_{jk} * y_{jk} + \sum_l Y_l * F_l + \sum_i \sum_l r_l * x'_{il}$$

*s.t.*

Source warehouse capacity constraints:

$$\sum_j x_{ij} + \sum_l x'_{il} \leq a_i \quad \forall i$$

Intermediate warehouse capacity constraints:

$$\sum_k y_{jk} \leq b_j \quad \forall j$$

External warehouse capacity constraints:

$$\sum_i x'_{il} \leq c_l \quad \forall l$$

Demand constraints:

$$\sum_j t_{jk} + \sum_l t'_{lk} \geq d_k \quad \forall k$$

Flow constraints:

$$\sum_i x_{ij} = \sum_k y_{jk} \quad \forall j$$

$$\sum_i x'_{il} = \sum_k y'_{lk} \quad \forall l$$

Logical constraint:

$$\sum_i x'_{il} \leq M * \sum_l Y_l$$

### 3.3.14 3D Printing Precision Parts

#### Problem Definition

You work for a company that manufactures high quality polymer parts for OrgoCorp, a galactic bioengineering company. The quality requirements are so high that you need to use very sophisticated 3D printing machines. You need to develop a mathematical programming model to organize monthly production considering:

- **Fixed operational costs (euros):** Each 3D printing machine  $m$  has a different fixed operational costs  $F_m$  that needs to be considered when a printing machine is used to print any units.
- **Unitary production costs (euros):** Each 3D printing machine  $m$  produces parts of product type  $p$  at a different unitary costs  $C_{mp}$
- **Capacity (minutes):** Each 3D printing machine  $m$  has a different capacity  $S_m$  (minutes)
- **Speed (minutes/part):** Each 3D printing machine  $m$  prints a unit of product type  $p$  at a different unitary speed
- **Demand (units):** OrgoCorp has confirmed a demand for  $d_p$  units of every product type  $p$  Build a model that takes into account these requirements, identifying indices, decision variables, objective function and constraints.

The CEO of OrgoCorp, The High Evolutionary, would like to set up a new contract allowing more flexibility in the delivery of units, such that:

- **Demand per period (units):** Now you need to consider that the demand for every product type  $p$  is not constant and depends on the planning period  $t$ ,  $D_{pt}$
- **Delayed demand (units):** You may delay the production for one period. For every unit of product type  $p$  that is delayed, OrgoCorp will charge an extra delayed demand cost of  $B_{pt}$  Modify the model to take into account this new contract

#### Solution

##### Part I: No delayed demand

The following mathematical programming model is proposed to model the problem:

Indices

- $m \in M$ : Set of 3D printing machines
- $p \in P$ : Set of product types



## Decision Variables

- $x_{mp}$ : (Integer) Number of units of product type  $p$  printed by 3D printing machine  $m$
- $Y_m$ : (Binary) 1 if 3D printing machine  $m$  is used, 0 otherwise

## Objective Function

- **Minimize** the total cost of production

$$\min z = \sum_{m=1}^M F_m \cdot Y_m + \sum_{m=1}^M \sum_{p=1}^P C_{mp} \cdot x_{mp}$$

where the first term are the fixed operational costs and the second term are the unitary production costs.

## Constraints

- **Capacity constraint:** The total time used by all the 3D printing machines cannot exceed the total capacity of the machines

$$\sum_{p=1}^P x_{mp} \cdot T_{mp} \leq S_m \cdot Y_m \quad \forall m \in [1, 2, \dots, M]$$

Where  $T_{mp}$  is the time needed (in minutes) to print a unit of product type  $p$  in 3D printing machine  $m$ . Based on the data provided, this time is calculated as the inverse of the speed of the machine for that product type, which we can note as  $S_{mp}$ , that is  $T_{mp} = \frac{1}{S_{mp}}$

Note that we are using the binary variable  $Y_m$  to ensure that the capacity constraint is only considered if the machine is used.

- **Demand constraint:** The total number of units of product type  $p$  printed by all the 3D printing machines must be equal or greater than the demand for that product type

$$\sum_{m=1}^M x_{mp} = d_p \quad \forall p \in [1, 2, \dots, P]$$

That is, the total number of units of product type  $p$  printed by all the 3D printing machines must be equal to the demand for that product type. We could add some slack to this constraint, making the demand constraints of type greater or equal, but we are assuming that the demand is always met exactly (being a precision parts company, we cannot afford to deliver more or less units than the ones requested).

## Model

Finally, the complete model is:

$$\min z = \sum_{m=1}^M F_m \cdot Y_m + \sum_{m=1}^M \sum_{p=1}^P C_{mp} \cdot x_{mp}$$

s.t

$$\sum_{p=1}^P x_{mp} \cdot T_{mp} \leq S_m \cdot Y_m \quad \forall m \in [1, 2, \dots, M]$$

$$\sum_{m=1}^M x_{mp} = d_p \quad \forall p \in [1, 2, \dots, P]$$

## Part II: Delayed demand

Based on the model proposed in part I, we need to introduce a new index  $t$  to represent the planning period. We also need to modify the decision variables to introduce the new index and allow our solution to adapt production in each period to demand. Therefore, the decision variables  $x_{mp}$  now becomes  $x_{mpt}$ , and  $Y_m$  becomes  $Y_{mt}$ . We also need to introduce a new set of decision variables to model delayed demand,  $D_{mpt}$ , which represents the number of units of product type  $p$  that are delayed in period  $t$  and printed in period  $t + 1$  by 3D printing machine  $m$ , and modify the objective function and constraints accordingly.

### Indices

- $m \in M$ : Set of 3D printing machines
- $p \in P$ : Set of product types
- $t \in T$ : Set of planning periods

### Decision Variables

- $x_{mpt}$ : (Integer) Number of units of product type  $p$  printed by 3D printing machine  $m$  in period  $t$
- $Y_{mt}$ : (Binary) 1 if 3D printing machine  $m$  is used in period  $t$ , 0 otherwise
- $I_{pt}$ : (Integer) Number of units of product type  $p$  delayed in period  $t$  and printed in period  $t + 1$

### Objective Function

- **Minimize** the total cost of production

$$\min z = \sum_{t=1}^T \sum_{m=1}^M F_m \cdot Y_{mt} + \sum_{t=1}^T \sum_{m=1}^M \sum_{p=1}^P C_{mp} \cdot x_{mpt} + \sum_{t=1}^T \sum_{p=1}^P B_{pt} \cdot I_{pt}$$

where the first term are the fixed operational costs, the second term are the unitary production costs and the third term are the delayed demand costs.

### Constraints

- **Capacity constraint:** The total time used by all the 3D printing machines cannot exceed the total capacity of the machines. We need to consider the time period:

$$\sum_{p=1}^P x_{mpt} \cdot T_{mp} \leq S_m \cdot Y_{mt} \quad \forall m \in [1, 2, \dots, M], \forall t \in [1, 2, \dots, T]$$

- **Demand constraint:** The total number of units of product type  $p$  printed by all the 3D printing machines must be equal or greater than the demand for that product type. Now, we need to make sure that the units produced satisfy the demand and the delayed demand:

$$\sum_{m=1}^M x_{mpt} = I_{pt-1} + D_{pt} - I_{pt} \quad \forall p \in [1, 2, \dots, P], \forall t \in [1, 2, \dots, T]$$

$$I_{p0} = 0 \quad \forall p \in [1, 2, \dots, P]$$

That is, the number of units produced in period  $t$  must be equal to the units delayed in the previous period  $t - 1$  plus the demand in period  $t$  minus the units delayed for the next period  $t + 1$ . Note that we are considering that the units delayed in period  $t$  are printed in period  $t + 1$ , so we need to consider the units delayed in period  $t - 1$  to calculate the units produced in period  $t$ . Finally, we are considering that there are no units delayed in the previous period  $t = 0$ .

Model

Summarizing, the complete model is:

$$\begin{aligned} \min z = & \sum_{t=1}^T \sum_{m=1}^M F_m \cdot Y_{mt} + \sum_{t=1}^T \sum_{m=1}^M \sum_{p=1}^P C_{mp} \cdot x_{mpt} + \sum_{t=1}^T \sum_{p=1}^P B_{pt} \cdot I_{pt} \\ \text{s.t} & \\ & \sum_{p=1}^P x_{mpt} \cdot T_{mp} \leq S_m \cdot Y_{mt} \quad \forall m \in [1, 2, \dots, M], \forall t \in [1, 2, \dots, T] \\ & \sum_{m=1}^M x_{mpt} = I_{pt-1} + D_{pt} - I_{pt} \quad \forall p \in [1, 2, \dots, P], \forall t \in [1, 2, \dots, T] \\ & I_{p0} = 0 \quad \forall p \in [1, 2, \dots, P] \end{aligned}$$

### 3.3.15 Spellman's Distribution Network Optimization

Spellman's would like Sabrina to focus on the optimization of their distribution network, to support the distribution of the 2 types of drinks from 3 sourcing facilities to 4 destination warehouses, through 6 intermediate warehouses. Sabrina needs to take into account the transportation costs (both from sourcing warehouses to intermediate warehouses and from intermediate warehouses to final destinations). Additionally, Sabrina can decide how many of the 6 intermediate solutions to use, given that they have a fixed operational cost that needs to be paid when the warehouse is used.

The following tables contain the transportation problem data:

#### Capacities of Sourcing Facilities (in units)

Sourcing Warehouse	Product A Capacity (units)	Product B Capacity (units)
S1	300	400
S2	250	150
S3	350	250

#### Demands at Final Destinations (in units)

Destination Warehouse	Product A Demand (units)	Product B Demand (units)
D1	150	150
D2	200	250
D3	175	175
D4	225	175

**Sourcing to Intermediate Warehouses Transportation Costs (Euros)**

From/To	I1	I2	I3	I4	I5	I6
S1	2	4	5	7	3	6
S2	3	2	6	8	4	5
S3	1	3	4	6	5	7

**Intermediate to Destination Warehouses Transportation Costs (Euros)**

From/To	D1	D2	D3	D4
I1	5	2	3	4
I2	4	1	5	2
I3	3	4	2	5
I4	5	3	4	1
I5	2	5	3	4
I6	4	2	1	5

**Operational Costs of Intermediate Warehouses (Euros)**

Warehouse	Operational Cost
I1	1000
I2	800
I3	1200
I4	1100
I5	700
I6	900

Note that the distribution costs are independent of the product type.

1. Help Sabrina write an Integer Programming Problem to model the transportation problem .

**Index** -  $i$ : Index of the sourcing warehouse -  $j$ : Index of the intermediate warehouse -  $k$ : Index of the destination warehouse -  $l$ : Index of the product type

**Decision variables** -  $x_{ijl}$ : Number of units of product  $l$  transported from sourcing warehouse  $i$  to intermediate warehouse  $j$  (Integer) -  $y_{jkl}$ : Number of units of product  $l$  transported from intermediate warehouse  $j$  to destination warehouse  $k$  (Integer) -  $Z_j$ : Binary variable indicating if intermediate warehouse  $j$  is used (Binary)

**Objective function**  $\min z = \sum_j Z_j * F_j + \sum_i \sum_j \sum_l s_{ijl} * x_{ijl} + \sum_j \sum_k \sum_l t_{jkl} * y_{jkl}$

Where: -  $F_j$  is the operational cost of intermediate warehouse  $j$ .  $F = [F_1, F_2, F_3, F_4, F_5, F_6] = [1000, 800, 1200, 1100, 700, 900]$  -  $s_{ijl}$  is the transportation cost from sourcing warehouse  $i$  to intermediate warehouse  $j$  for product  $l$ . The transportation costs do not depend on the product type and are indicated in the table above. -  $t_{jkl}$  is the transportation cost from intermediate warehouse  $j$  to destination warehouse  $k$  for product  $l$ . The transportation costs do not depend on the product type and are indicated in the table above.

**Constraints** - Sourcing capacity constraints:  $\sum_j x_{ijl} \leq C_{il} \quad \forall i \quad \forall l$

Where  $C_{il}$  is the capacity of sourcing warehouse  $i$  for product  $l$ . The capacities are indicated in the table above.

- Destination demand constraints:  $\sum_j y_{jkl} = D_{kl} \quad \forall k \quad \forall l$

Where  $D_{kl}$  is the demand of destination warehouse  $k$  for product  $l$ . The demands are indicated in the table above.

- Flow conservation constraints:

$$\sum_i x_{ijl} = \sum_k y_{jkl} \quad \forall j \quad \forall l$$

- Logical constraints:

$$\sum_i x_{ijl} \leq M * Z_j \quad \forall j$$

2. How would a Greedy algorithm solve the problem? Motivate your response

A greedy algorithm would solve the problem by making a series of decisions that are locally optimal at each step with the hope of finding a global optimum. The algorithm would start by selecting the intermediate warehouse with the lowest operational cost and then select the sourcing warehouse that minimizes the transportation cost to that intermediate warehouse. The algorithm would then select the destination warehouse that minimizes the transportation cost from the intermediate warehouse. This process would be repeated for each product type until all constraints are satisfied.

The greedy algorithm is not guaranteed to find the optimal solution to the transportation problem. However, it is computationally efficient and can provide a good solution in a reasonable amount of time. The algorithm is particularly useful when the problem is large and complex, as it can quickly generate a feasible solution that can be used as a starting point for more sophisticated optimization techniques.

### 3.3.16 Professors Xavier's School for Gifted Youngsters

#### Problem Definition

Professor Xavier's School for Gifted Youngsters is organizing a special summer session focusing on enhancing the students' unique abilities. Students from across the country have been invited, and the school needs to arrange their transportation. The goal is to ensure that all students can arrive at the school in time for the session while minimizing transportation costs.

1. First, let us consider an abstract model where gifted students reside in  $n$  different locations, can be transported in  $m$  different modes of transport. Each mode of transport from each location to the school has a different associated cost. Additionally, each transportation mode has a different capacity, and each location has a different number of gifted students that need transportation to the school. Write down an Integer Programming (IP) problem that minimizes the overall transportation costs.

**Index** -  $i \in \{1, 2, \dots, n\}$ : Index for the locations -  $j \in \{1, 2, \dots, m\}$ : Index for the modes of transport

**Decision Variables:** -  $x_{ij}$ : Number of students transported from location  $i$  to the school using mode of transport  $j$ .  $x_{ij}$  is an integer variable.

**Objective Function:** Minimize the total transportation cost:

$$\min z = \sum_{i=1}^n \sum_{j=1}^m c_{ij} x_{ij}$$

Where  $c_{ij}$  is the cost of transporting a student from location  $i$  to the school using mode of transport  $j$ .

**Constraints:**

1. Ensure that all students are transported to the school:

$$\sum_{j=1}^m x_{ij} = s_i, \forall i \in \{1, 2, \dots, n\}$$

Where  $s_i$  is the number of students at location  $i$ .

2. Ensure that the capacity of each mode of transport is not exceeded:

$$\sum_{i=1}^n x_{ij} \leq t_j, \forall j \in \{1, 2, \dots, m\}$$

Where  $t_j$  is the capacity of mode of transport  $j$ .

2. Given the following number of students:

Location	Number of Students
L1	20
L2	30
L3	25
L4	15
L5	40
L6	35

And the following capacities and costs of transport per student from the different locations for three different modes of transport:

Mode	Capacity	L1	L2	L3	L4	L5	L6
T1	50	10	8	12	11	9	7
T2	60	9	7	11	10	8	6
T3	40	8	9	10	12	7	11

Draw a graph that allows to model the transportation problem as a maximum flow at minimum cost transportation problem.

## 3.4 Python libraries

This section contains Python library tutorials:

### 3.4.1 Introduction to Graph Algorithms with Networkx

Try me



#### Introduction

This notebook provides an overview and tutorial of [Networkx](#), a Python package to create, manipulate, and analyse graphs with an extensive set of algorithms to solve common graph theory problems. Basically, we will use Networkx to build a network model of the network, and present some of the most important algorithms to solve the problems covered in the book, and finally we will see some nice add-ons and tools to draw networks and create networks from other data structures such as open maps!

## Requirements

You need to install Networkx in your runtime, so make sure you run this script if not already installed.

Note that the installation line includes the *extra* option to install additional packages which are useful to work with Networkx.

```
[ ]: !pip install networkx[default,extra]
!pip install ipython
!pip install pandas
```

## Installation

We are going to import the package as *nx* in this tutorial:

```
[ ]: import networkx as nx
```

## The basics

### Creating networks

We can create different types of graphs with the following constructor functions. Review the definitions in the [Graph theory tutorial](#):

- `nx.Graph`: Creates an object representing an undirected graph.
- `nx.DiGraph`: Creates an object representing a directed Graph
- `nx.MultiGraph`: Creates an object representing a with multiple edges between any pair of nodes
- `nx.MultiDiGraph`: Directed graph with multiple edges between any pair of nodes

By default, graphs are created without nodes or edges. The simplest way to add edges and nodes is to use the class methods `add_node` and `add_edge`. The former adds the object that is passed as argument as a node of the graph. We can add as a node a string variable, a numeric, or a tuple with several values which are relevant for our application. In fact, we can add any *hashable* object, any object that will not change during the execution of the program. Functions and objects of user-defined classes are also hashable by default. You can find the precise definition of hashable in the [glossary](#), but what is important is that Python can assign a unique identifier to each hashable object, and networkx uses these feature to identify each node of the network. Normally, we will use simple types like strings or integers, with different values for each node. The function `add_edge` creates an edge passing between two objects as arguments. If the objects are already in the network, it will create an edge between the two, as long as the type of network allows it (if not, it will raise an exception). If any of the objects passed as arguments are not in the network, the function will also add them as nodes. For instance:

```
[ ]: # Create an undirected graph
my_first_Graph = nx.Graph()

# Add an edge between nodes 1, 2. Automatically creates nodes 1 and 2
my_first_Graph.add_edge(1, 2)

# Add a third node
my_first_Graph.add_node(3)
```

We can add attributes to the edges. We can define as many attributes as we want and pass them as named arguments to the `add_edge` function, for instance:

```
[ ]: # add another edge with weight 3 between nodes 2 and 3.
my_first_Graph.add_edge(2, 3, weight=3)
```

Graph properties describe the structure of the graph, for instance, the list of nodes and edges. Some important properties are:

- `nodes`: Provides a list containing the nodes of the graph. Each element of the list is the object used to add a node.
- `edges`: Provides the list of the edges of the graph. Each edge is represented as a tuple where the first element is the object that represents the origin node and the second element is the object that represents the destination node.
- `adjacency`: The adjacency provides a dictionary containing the information in the adjacency matrix. Normally, the adjacency matrix will be sparse (containing lots of zeroes) and, for the sake of efficiency, networkx represents the adjacency as a dictionary where the node objects are keys and the values are dictionaries with the information of the edges with origin at the key node. The information of the edges are also dictionaries, where each key is the destination node object and the values are dictionaries with the attributes of the edge (see example below).
- `degree`: The degree of each node of the graph is represented as a tuple where the first element is the node object and the second element an integer with the degree of the graph.
- `size()`: The size of the graph (integer)
- `order()`: The order of the graph (integer)

Let's see these properties with the example above:

```
[ ]: # The nodes property contains a list of nodes
print(my_first_Graph.nodes)

# The edges:
print(my_first_Graph.edges)

# The adjacency:
print(my_first_Graph.adj)

# Or the order:
print(my_first_Graph.order())

#Or the size:
print(my_first_Graph.size())

#Or the degree
print(my_first_Graph.degree)
```



## Algorithms

Networkx has an extensive library of algorithms to solve graph theory problems, ranging from path algorithms to flow algorithms. The next sections provide practical examples and guidelines to solve common graph theory problems using Networkx.

### Path algorithms

It packages several algorithms to compute paths, the shortest paths and path lengths between nodes in a graph. The algorithms work both on directed and undirected graphs. Basically, they all take as arguments the graph, the source and target nodes, and optionally the edge attribute that represents the distance, cost or weight of each edge. If not provided, all edges have a weight equal to one.

#### shortest\_path

Shortest\_path computes the shortest paths in the graph. By default, it uses the Dijkstra algorithm, but it is possible to use Bellman-Ford's algorithm using the argument `method = 'bellman-ford'`. If only the source node is specified, shortest\_path returns a dictionary keyed by targets with the shortest path between the source node and every other node in the network. Conversely, if only the target is specified, it returns a dictionary keyed by sources with the shortest path from every node to the specified target node. If neither the source nor target are specified, it returns a dictionary of dictionaries with all shortest paths from every source and every target node.

```
[ ]: # Create a directed graph structure:
G = nx.DiGraph()

G.add_edge(0,1, cost=1700)
G.add_edge(0,2, cost=3968)
G.add_edge(0,3, cost=7622)
G.add_edge(0,4, cost=10970)
G.add_edge(1,2, cost=1880)
G.add_edge(1,3, cost=1880)
G.add_edge(1,4, cost=4652)
G.add_edge(2,3, cost=2300)
G.add_edge(2,4, cost=4316)
G.add_edge(3,4, cost=2720)

# Get the shortest path between nodes 0 and 4:
g_shortest_paths = nx.shortest_path(G, 0, 4, weight='cost')
print(g_shortest_paths)
```

#### minimum\_spanning\_tree

minimum\_spanning\_tree returns the minimum spanning tree of an undirected graph. Recall that the minimum spanning tree is a sub-graph that connects all the nodes in the graph with the minimum sum of edge weights. If the graph is not connected, the function returns a collection of the minimum spanning trees of each connected sub-graph. This is referred to as a spanning forest.

```
[ ]: G = nx.Graph()
G.add_node("a")
G.add_node("d")
G.add_edge("a", "b", weight=3)
G.add_edge("a", "c", weight=6)
```

(continues on next page)

(continued from previous page)

```
G.add_edge("b", "d", weight=1)
G.add_edge("c", "d", weight=2)
T = nx.minimum_spanning_tree(G)
print(T.edges)
```

## Flow algorithms

### maximum\_flow

`maximum_flow` finds a maximum flow of a commodity in a directed graph where each node has a 'capacity' attribute that specifies the maximum capacity that the flow can support. It returns the value of the flow and the path that provides the maximum flow in a dictionary keyed with the source and destination nodes. See the example below:

```
[ ]: G = nx.DiGraph()
G.add_edge("x", "a", capacity=3.0)
G.add_edge("x", "b", capacity=1.0)
G.add_edge("a", "c", capacity=3.0)
G.add_edge("b", "c", capacity=5.0)
G.add_edge("b", "d", capacity=4.0)
G.add_edge("d", "e", capacity=2.0)
G.add_edge("c", "y", capacity=2.0)
G.add_edge("e", "y", capacity=3.0)

flow_value, flow_dict = nx.maximum_flow(G, "x", "y")
print(flow_value)
print(flow_dict)
```

### min\_cost\_flow

`G` is a directed graph with edge capacity and in which nodes have demand, `maximum_flow_value` finds a maximum flow from a source node to a destination node, using the node `s` to node `t`, using edge attribute 'capacity' as edge capacity and node attribute 'demand' as the demand. negative demand means that the node wants to send flow, a positive demand means that the node want to receive flow. A flow on the digraph `G` satisfies all demand if the net flow into each node is equal to the demand of that node.

```
[ ]: G = nx.DiGraph()
G.add_node("a", demand=-5)
G.add_node("d", demand=5)
G.add_edge("a", "b", weight=3, capacity=4)
G.add_edge("a", "c", weight=6, capacity=10)
G.add_edge("b", "d", weight=1, capacity=9)
G.add_edge("c", "d", weight=2, capacity=5)
flowDict = nx.min_cost_flow(G)
```

## Displaying networks

You can draw the graph with the function `draw(G, pos)`. The function takes a graph as the first argument. The second argument is a tuple with the relative (x, y) coordinates of nodes in the graph.

Networkx provides convenient functions to customise the representation of the network:

- The function `draw_networkx_labels` is used to control the presentation of the node labels (reference [here](#))
- The function `draw_networkx_nodes` to control the presentation of the nodes (reference [here](#))
- The function `draw_networkx_edge_labels` to control the edge labels (reference [here](#)).

See a practical example below:

```
[ ]: pos = {1: (0, 0),
          2: (1, 0),
          3: (2, 0)}
nx.draw(my_first_Graph, pos)
nx.draw_networkx_labels(my_first_Graph, pos)
nx.draw_networkx_nodes(my_first_Graph, pos, node_size=600, node_color='#efefef')
nx.draw_networkx_edge_labels(my_first_Graph, pos, edge_labels={(1, 2): 0, (2, 3): 3})
```

## Creating networks from datasets

### Add edges from edge list

Networkx provides convenient functions and methods to create networks from large data structures. This is necessary to build networks in practical applications. This section covers functions to create graphs from iterables and Pandas data frames.

Specifically, The method `add_edges_from` adds edges from an array where each element is a tuple (*edge tuple*) with the following elements:

- origin node object
- destination node object
- (optional) dictionary with edge properties

```
[ ]: my_graph = nx.DiGraph()
my_edges = [(1, 2, {"weight": 3, "capacity": 5}), # adds an edge from node 1, to node_
           ↪2, with weight 3 and capacity 5
           (2, 1, {"weight": 2, "capacity": 4}),
           (2, 3, {"weight": 3, "capacity": 6}),
           (3, 1, {"weight": 3, "capacity": 6}),
           (3, 4, {"weight": 2, "capacity": 5}),
           (4, 3, {"weight": 5, "capacity": 8}),
           (4, 5, {"weight": 6, "capacity": 7})
           ]

my_graph.add_edges_from(my_edges)

# Add several nodes with same weight and capacity:
my_graph.add_edges_from([(5,4), (5, 2), (5, 3)], weight=5, capacity= 9)

# Draw with spring layout
nx.draw_spring(my_graph, with_labels=True)
```

### Adding paths from lists

It is also possible to add paths to a graph using an array that contains the list of nodes in the path. If the nodes do not exist in the graph, they are automatically added. It is possible to define properties for the edges added to the path. let us see it in action:

```
[ ]: my_graph = nx.DiGraph()

path_1 = [1, 2, 3, 4]
# Add a path 1 -> 2 -> 3 4 with weight 3
nx.add_path(my_graph, path_1, weight = 3)

# Add a reverse path with weight 4
path_2 = [4, 3, 2, 1]
nx.add_path(my_graph, path_2, weight = 4)

nx.draw_planar(my_graph)
```

### Adding cycles from list

The function `add_cycle` adds a cycle connecting the nodes provided in an array. It is similar to `add_path`, except that the first and the last node are connected:

```
[ ]: my_graph = nx.DiGraph()

path_1 = [1, 2, 3, 4]
# Add a path 1 -> 2 -> 3 4 with weight 3
nx.add_cycle(my_graph, path_1, weight = 3)

nx.draw_planar(my_graph)
```

### Creating graphs from Pandas dataframes

#### from\_pandas\_edgelist

Networkx has a good integration with [Pandas](#). The function `from_pandas_edgelist` creates a graph from a Pandas dataframe that contains the information of the edges. You only need to specify which column specifies the origin node, the destination node and which columns contain information about the edges we want to map as attributes. The arguments are:

- `edgelist`: The dataframe containing the edge information
- `source`: The column that contains the source nodes
- `target`: The column that contains the target nodes
- `edge_attr`: An array with the names of the columns that contain edge attributes

```
[ ]: import pandas as pd
edge_df = pd.DataFrame({'u': [1, 2, 2, 3, 3, 4], 'v': [2, 3, 4, 1, 4, 1], 'weight': [2, 1, 3, 4, 2, 1]})

# Create a graph from the edge list
graph = nx.from_pandas_edgelist(edge_df, 'u', 'v', ['weight'])

nx.draw_planar(graph)
```

`from_pandas_adjacency_matrix`

It is also possible to create a graph from a Pandas dataframe that contains the adjacency matrix. An entry  $i,j$  in the dataframe corresponds to the weight of an edge from node  $i$  to  $j$ . For directed graphs, it is necessary to use the `create_using` argument equal to `nx.DiGraph`:

```
[ ]: adjacency_matrix = pd.DataFrame([[0, 1, 0, 4, 0, 3],
                                     [1, 0, 1, 0, 0, 7],
                                     [0, 0, 0, 1, 0, 8],
                                     [8, 0, 2, 0, 2, 5],
                                     [9, 0, 7, 2, 0, 1],
                                     [0, 0, 0, 1, 0, 0]])

graph = nx.from_pandas_adjacency(adjacency_matrix, create_using=nx.DiGraph)

print(graph.nodes)
print(graph.adj)
```

### Export data to Pandas dataframe

The functions `to_pandas_adjacency` and `to_pandas_edgelist` export the information of the graph to dataframes, either as the adjacency matrix or as an edge list (make sure you run the cell above):

```
[ ]: df = nx.to_pandas_edgelist(graph)
df
```

### Creating networks from Open Street Maps data

`osmnx` is a really nice package that allows to create a map from Open Street Map data, which is a really nice resource in logistics applications.

Installation

Run the following script to install `osmnx` if it is not installed in your system:

```
[ ]: !apt-get -qq install -y libspatialindex-dev && pip install -q -U osmnx
```

Creating a map

You can create a network from a map using three functions:

- `graph_from_place`: Creates a network map from a search query to Open Street Maps. For instance, the query 'Valencia, VC, Spain' returns a network of the street map of the city of Valencia. There are some requirements to build these queries, and if it does not work in your app, you can use any of the other two methods described in this section.
- `graph_from_address`: Creates a network map centered providing an address. The argument `address` specifies the address to geocode and use as the central point around which the graph is constructed. The argument `dist` specifies the maximum distance from the center address to any node in the graph.

- `graph_from_bbox`: Creates a graph within four points of a bounding box, specifying the north northern latitude of the bounding box, the south southern latitude of bounding box, the east eastern longitude of the bounding box and the west western longitude of the bounding box.

```
[ ]: import networkx as nx
import osmnx as ox
# get two markers from Google Maps around EDEM
# Latitude, Longitude
#North-west marker
# 39.468370277652426, -0.33528989341188187
# South east marker
# 39.45963084307057, -0.3184406656568226
G_nx = ox.graph_from_bbox(north=39.468370277652426,
                          south=39.45963084307057,
                          east=-0.3184406656568226,
                          west=-0.33528989341188187)
```

### Rendering the map

The function `plot_graph_folium` uses `Folium` to render the map in an interactive map:

```
[ ]: m1 = ox.plot_graph_folium(G_nx)
m1
```

### `get_nearest_node`

When you work on a graph, you can possibly click on a point in the map that is not necessarily an edge in the graph, so to be able to process it, you first need to find the node in the graph that is closest to the point. You can use the `get_nearest_point` function for this purpose, it returns the node that is closest to the point you pass as an argument so that you can later use the algorithms:

```
[ ]: edem = (39.46211739713285, -0.3288013882525529)
nearest_node = ox.get_nearest_node(G_nx, edem)
nearest_node
```

### Networkx algorithms

You can now run any algorithm on your graph. The following example shows the spanning tree in green. Since `spanning_tree` returns a graph, we can use again the function `plot_graph_folium` to render the result. Since we want to show it over the previous generated graph, we pass the `Folium` map object `m1` created below as the value of the named parameter `graph_map`, and we use the `edge_color` `red` to represent the edges of the spanning tree in a different color.

```
[ ]: spanning_tree = nx.minimum_spanning_tree(G_nx.to_undirected())
ox.plot_graph_folium(spanning_tree, graph_map=m1, edge_color='#ff0000')
```

## plot route

You can plot a route in a map using the function `plot_route()`. Optionally, you can pass as argument a previously generated map so that the route is printed as an overlay:

```
[ ]: edem = (39.46211739713285, -0.3288013882525529)
      museum = (39.46329739674903, -0.33194815644130443)
      edem_nearest = ox.get_nearest_node(G_nx, edem)
      museum_nearest = ox.get_nearest_node(G_nx, museum)
      route = nx.shortest_path(G_nx, edem_nearest, museum_nearest)
      ox.plot_route_folium(G_nx, route, route_map=m1, weight=7)
```

```
[ ]:
```





## NON-LINEAR PROGRAMMING

This chapter provides an introduction to Non-Linear Programming (NLP), the branch of optimisation that deals with problem models where the functions that define the relationship between the unknowns (either objective function or constraints) are not linear. The fact that the functions are not linear makes NLP a lot harder to deal with from a mathematical point of view. The chapter will focus on specific cases that have many applications in real life problems, particularly quadratic functions and their relationship to machine learning.

### 4.1 Tutorials

This section contains some basic tutorials:

#### 4.1.1 Non-Linear Programming

Try me



#### Introduction

A non-linear programming (NLP) problem is similar to a linear programming problem in the sense that it is also composed of an objective function and constraints, but a NLP includes **at least** a non-linear function. In general, this chapter will deal with continuous decision variables and problems where either the objective function, or any of the constraints are not linear. NLP problems are in many cases more difficult to solve than linear programming problems. The mathematical foundations of NLP optimisation date back to the enlightenment era and the work of Joseph Louis Lagrange, a great mathematician and astronomer. He developed a method to take into account non-linear constraints into optimisation problems, which is called Lagrange multipliers, and that will be used in constrained NLP. In this introduction, we will use some Python code snippets to show why NLP are more difficult to solve.

### Hard to distinguish local and global optimal values

In linear programming, methods like the simplex need only to know the value of the unknowns and the value of the objective function to determine if the solution is optimal, because the maximum is always at a vertex of the feasibility region. Non-linear functions have **critical points** (we will provide a more formal definition of a critical point later on in the book), which can be either local or global maxima or minima:

- **Local maximum:** The value of the function at the critical point is higher than the values of the function around the critical point.
- **Global maximum:** The value of the function at the critical point is the highest value of the function.
- **Local minimum:** The value of the function at the critical point is lower than the values of the function around the critical point.
- **Global minimum:** The value of the function at the critical point is the lowest value of the function.

Let us look at one example in Python.

```
[27]: import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

x = np.linspace(-3, 3, 100)
y = (np.sin(x**3) - 5*np.sin(x**2) + 4*np.sin(x) + 12)**2 - 4
plt.plot(x,y)

plt.title('$f(x)=(\sin(x^3) - 5\sin(x^2) + 4\sin(x) + 12)^2 - 4$')

# sets limits
plt.xlim((-3,3))
plt.xlabel(r'$x$')

plt.ylim((0,459))
plt.ylabel(r'$f(x)$')

global_min = np.min(y)
global_min_x = x[np.argmin(y)]

global_max = np.max(y)
global_max_x = x[np.argmax(y)]

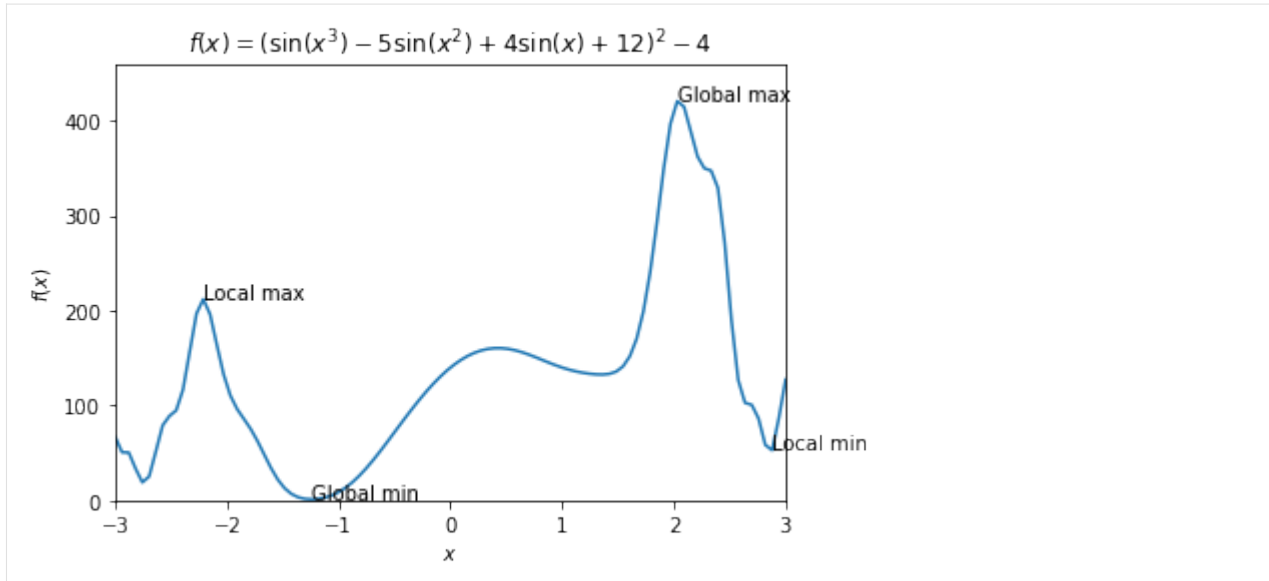
plt.annotate('Global min', (global_min_x, global_min))
plt.annotate('Global max', (global_max_x, global_max))

local_min = np.min(y[x > 1])
local_min_x = x[y == local_min]

plt.annotate('Local min', (local_min_x, local_min))
local_max = np.max(y[x < -1])
local_max_x = x[y==local_max]

plt.annotate('Local max', (local_max_x, local_max))

[27]: Text([-2.21212121], 212.0460765561599, 'Local max')
```

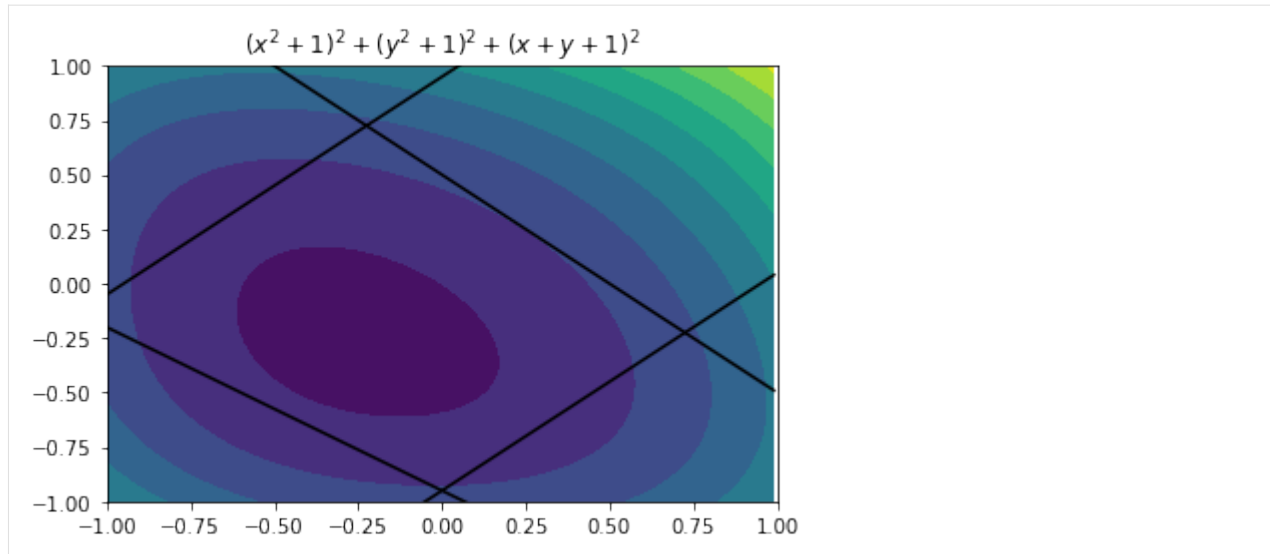


As it can be noted in the graph, the function has many critical points. Any critical point can be a local maximum or a global maximum depending on the range of values that  $x$  can take. For instance, if  $x$  could only take values in the range  $x \in [-3, 0]$ , then the local maximum would become a global maximum. This makes it somewhat more difficult to find the optimal solution in an optimisation problem.

### Optima not restricted to vertex

With non-linear functions, the optimal (maximum or minimum) value might no longer be at one of the vertices. For instance, let us take us an example the function in the following Python script:

```
[44]: x = np.arange(-1, 1, 0.01)
y = np.arange(-1, 1, 0.01)
xx, yy = np.meshgrid(x, y, sparse=True)
z = (xx**2+ 1)**2+ (yy**2+ 1)**2+ (xx+yy+ 1)**2
h = plt.contourf(x, y, z, levels=10)
plt.plot(x, y+0.95, color='black')
plt.plot(x, -y+0.5, color='black')
plt.plot(x, y-0.95, color='black')
plt.plot(x, -0.75*y-0.95, color='black')
plt.xlim((-1,1))
plt.ylim((-1,1))
plt.ylabel('$y$')
plt.xlabel('$x$')
plt.title('$ (x^2+ 1)^2+ (y^2+ 1)^2+ (x+y+ 1)^2 $')
plt.show()
```



The example shows a non-linear function on two decision variables,  $x$  and  $y$ . The black lines represent some constraints limiting the feasibility region. The contour graph shows that the maximum value of the objective function is within the interior of the feasibility region and not in one of the vertices.

The following situations are examples where the optimal value might not be at one vertex of the feasibility region:

- Multimodal function: As in the example above, the function might be **multimodal** and depend on several decision variables, increasing its complexity.
- Interior optima: As in the example above, the optimal value might be found in an interior point of the feasibility region.
- Discontinuous: The objective function might be discontinuous and again, the optimal value might not be surely located in one vertex.

### Complex mathematical theory

Given that the objective functions and the constraints can be of any family of functions, potentially, there is a huge body of mathematical theory that can be applied to NLP, since non-linear functions have a wide range of characteristics. Definitively, this makes NLP somewhat more difficult. The theoretical framework is more sophisticated, and modeling is also more difficult, since the researcher needs to define the types of function to use in the model.

### Solver selection

Due to the complex mathematical theory behind, selecting the right solver to solve a specific problem can be complex. In fact, different algorithms and solvers can provide different results. Normally, algorithms deal with local optimisation, rather than global optimisation, and for that reason, the selection of the initial point used to start the algorithm might have an impact in the solution. Hence, different algorithms might provide different solutions, but when a specific algorithm is selected, its configuration (particularly the initial point) might yield different results.

## Types of problems

In this chapter, NLP is divided into two different classes of problems with increasing complexity:

- **Unconstrained NLP Problems** This class of NLP problems have no constraints, we look for a maximum or minimum of the objective function within some bounded decision variables, but not subject to any constraint.
- **Constrained NLP Problems** This class of problems have constraints, i.e. there are restrictions to the objective function solution determined by a set of functions of the decision variables.

Additionally, to better present the mathematical framework behind NLP, we will consider **One-Dimensional** unconstrained problems, where there is only one decision variable, and **Multi-Dimensional** or multi-modal, where there are several decision variables

### 4.1.2 Unconstrained Non-Linear Programming

#### One dimensional problems set-up

Let us first consider just one decision variable. The objective is to search a maximum or minimum of a non-linear function. Without loss of generalisation, let us consider that the objective is to minimize the objective function:

$$\min f(x) \quad x \in \mathbb{R}$$

Any (local or global) minimum of  $f(x)$ ,  $x^*$  will have the following properties:

1. **First derivative test:** The first derivative of a function at a point  $x^*$  represents the slope of a line tangent to  $f(x)$  at  $x^*$ . In a minimum, the first derivative is equal to zero,  $\frac{df(x)}{dx} = 0$ , meaning that the tangent is horizontal.
2. **Second derivative test:** The second derivative is positive  $\frac{d^2 f(x)}{dx^2}$ , meaning that the slope of the tangent is increasing and therefore, it must be less than zero for  $x < x^*$ , meaning that the function decreases before  $x^*$ , and greater than zero for  $x > x^*$ , meaning that the function increases after  $x^*$ . Therefore  $f(x)$  takes its minimum value at  $x^*$ , at least in its proximity.

By definition,  $x^*$  is a critical point. For a maximum, the second derivative is negative and using the same logic,  $f(x)$  decreases in the proximity of  $x^*$ .

#### Bi-section algorithm

Since the sign of the first derivative changes at a critical point, it is possible to search a critical point if we find two initial points such that the sign of the first derivative is different. The simplest algorithm that uses this property is known as the bi-section method, and it can be formulated as this:

##### Initialisation

1. Choose an acceptable error  $\epsilon$
2. Find an initial left point  $x_L$  such that  $\frac{df(x_L)}{dx} < 0$
3. Find an initial right point  $x_R$  such that  $\frac{df(x_R)}{dx} > 0$

**Iteration 1.** Bisect the range  $[x_L, x_R]$   $x' = \frac{x_L + x_R}{2}$  2. If  $\frac{df(x')}{dx} < 0$  then make  $x_L = x'$ , else if  $\frac{df(x')}{dx} > 0$  then make  $x_R = x'$   
 3. If  $|x_R - x_L| \leq 2\epsilon$  then the solution is  $\frac{x_L + x_R}{2}$ , else repeat

That is, we start with two initial values for which the first derivative has different sign. If the function is continuously differentiable, then there must be a value somewhere in between these values where the first derivative is zero. Then, we make a bi-section and continue looking in the part of the range for which there is a change in the sign of the first derivative. The process stops either when we find exactly a critical point or when the distance between both points is less than the acceptable error.

### Multidimensional problems set-up

For multidimensional problems the logic applied is the same, except that in this case it is necessary to consider partial derivatives in the different decision variables or dimensions of the problem. Thus, given an n-dimensional function:

$$f(x) \quad x = [x_1, x_2, \dots, x_n]$$

The **gradient vector** is defined as a vector containing the first partial derivatives:

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix}$$

The gradient represents the slopes of the tangent planes to  $f(x)$  in every dimension. For a minimum, the gradient must be zero, as an extension of the first derivative test of one dimensional problems.

1. **Gradient test** The gradient must be zero at a critical point  $x^* \nabla f(x^*) = [0, \dots, 0]^T$ .

Now, to know if the function decreases **in every direction** around  $x^*$ , we need to define the **Hessian**:

$$\text{Hess}f(x) = \begin{bmatrix} \frac{d^2 f(x)}{dx_1^2} & \frac{d^2 f(x)}{dx_1 dx_2} & \dots & \frac{d^2 f(x)}{dx_1 dx_n} \\ \frac{d^2 f(x)}{dx_2 dx_1} & \frac{d^2 f(x)}{dx_2^2} & \dots & \frac{d^2 f(x)}{dx_2 dx_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{d^2 f(x)}{dx_n dx_1} & \frac{d^2 f(x)}{dx_n dx_2} & \dots & \frac{d^2 f(x)}{dx_n^2} \end{bmatrix}$$

$$\text{Hess}f(x) = \begin{bmatrix} h_{11} & h_{12} & \dots & h_{1n} \\ h_{21} & h_{22} & \dots & h_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ h_{n1} & h_{n2} & \dots & h_{nn} \end{bmatrix}$$

That is, the Hessian is a nxn matrix that contains the second order derivatives.

To determine how the function increases or decreases in any direction around a given point, we can check **Sylvester's criterion**. This criterion evaluates the **leading principal minors** of the Hessian matrix. A **leading principal minor**  $k$  of a Hessian matrix is the determinant of the square sub-matrix that includes columns and rows from 1 to  $k$ , that is, if note,  $H_k$  as the leading minor of the Hessian matrix A, we can compute the leading principal minors as:

$$H_1 = |h_{11}|$$

$$H_2 = \begin{vmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{vmatrix}$$

...

Note that an n-dimensional function has n Hessian leading minors. Now, the Hessian determinants can help us decide whether a function is a local or global minimum, according to the following definitions:

- **Definitive positive Hessian:** all leading principal minors are greater than zero  $> 0 \forall i \in [1, \dots, n]$ .
- **Semi-definitive positive Hessian:** all leading principal minors are greater or equal than zero  $\geq 0 \forall i \in [1, \dots, n]$ .
- **Definitive negative Hessian:** all leading principal minors are lower than zero  $< 0 \forall i \in [1, \dots, n]$ .
- **Semi-definitive negative Hessian:** all leading principal minors are less or equal than zero  $\leq 0 \forall i \in [1, \dots, n]$ .

Now, the second derivative test becomes:

2. **Second partial derivative test:**

- In a global maximum, the Hessian is definitive negative
- In a global minimum, the Hessian is definitive positive
- In a local maximum, the Hessian is semi-definitive negative
- In a local minimum, the Hessian is semi-definitive positive

### 4.1.3 Gradient Descent Methods

#### Introduction

Gradient Descent Methods constitute a large family of heuristic methods based on the gradient definitions presented in the set-up of unconstrained NLP. Gradient descent methods are useful to find critical points and test whether they are global minimum (for minimisation problems) or global maximum (for maximisation problems).

Basically, they consist of two types of functions:

- **search functions:** to find points where the gradient is zero or close to zero, i.e. functions to find critical points.
- **test functions:** to verify whether the critical point is a global or local maximum or minimum.

The different implementations have different performance trade-offs depending on the characteristics of the optimisation problem. In this notebook, we are going to cover two different methods, the simple gradient descent method and the stochastic gradient descent, with applications in machine learning.

#### Simple Gradient descent

In the Simple gradient descent method, the algorithm starts at an initial point, calculates the gradient, and if it's different from zero, it looks for the next point to test *in the direction opposite to the gradient*. Recall that, at any given point  $x^*$  the gradient of the objective function  $\nabla f(x^*)$  represents a vector tangent to  $f(x^*)$ . This tangent vector represents the direction of the maximum change of  $x$  towards infinity in the  $n$ -dimensional space. Therefore, the algorithm continues the search in the opposite direction to this vector, assuming that this will provide the maximum change of the function towards its minimum value. With this, the Simple Gradient descent method can be defined as:

#### Start

- Choose an acceptable error  $\epsilon$
- Choose a starting point  $x'$

#### Iterate

- Calculate  $\nabla f(x')$
- If  $|\nabla f(x')| \leq \epsilon$  then exit with  $x'$  as the solution, else
- Set  $x'' = x' - t \cdot \nabla f(x')$   $t \geq 0, t \in \mathbb{R}$  and iterate again.

Note that since  $t$  is a positive value and  $|\nabla f(x')|$  is a vector, we are changing the value of  $x$  in the direction of the vector. The change is proportional to  $t$ , which can be a fixed parameter or a calculated parameter depending on the implementation. One implementation alternative is to use a one-dimensional search (e.g. using a bisection algorithm) to find  $t^*$  such that  $f(x'')$  is a local minimum. Note that since both  $x'$  and  $|\nabla f(x')|$  are vectors, the only unknown is  $t$ , and this bisection is a reduction of the multidimensional minimisation problem to a one-dimensional minimisation problem.

## Stochastic Gradient Descent (SGD)

The stochastic gradient descent method can be applied in problems where the gradient of the objective function can be estimated from a subset of the parameters needed to compute the closed form of the gradient. The stochastic gradient descent is one of the fundamental optimization methods used in machine learning and given its importance, we will illustrate some outstanding applications of this method in the field of machine learning the of this method using in the quadratic optimization examples. Let us consider the following objective function:

$$\min R_n(\theta) = \frac{1}{n} \sum_{t=1}^n L_i(\theta)$$

The stochastic gradient descent method is defined as:

### Start

- Choose an acceptable error  $\epsilon$
- Choose a starting point  $\theta$  (e.g  $\theta = 0$ )
- Choose a learning factor for each iteration  $k$   $\zeta_k$

### Iterate

- select a random value  $t^*$
- Calculate the gradient of  $\nabla L_{t^*}(\theta_k)$
- Update  $\theta_k$  in the direction contrary to the gradient at  $t^*$ :  $\theta_{k+1} = \theta_k - \zeta_k \cdot \nabla L_{t^*}(\theta_k)$
- Repeat until the difference between two consecutive estimations is sufficiently small, or until the maximum number of iterations is reached.

## 4.1.4 Interactive Gradient-Descent in Python

Try me



### Introduction

Gradient-Descent (GD) methods are in the backbone of many machine learning methods. in this tutorial, we will use a practical example to illustrate how the simplest algorithm in the family (Simple Gradient Descent) works.

### Import libraries

Before we dive into GD methods, let's import the libraries that we will use in this tutorial:

```
[ ]: import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
```



## Problem definition

Gradient-Descent methods allow us to find the maximum or minimum of a function that depends on an array of variables:

$$x = [x_1, x_2, \dots, x_n]$$

For minimization, we want to find the minimum value of a function of  $x$ :

$$\min z = f(x)$$

And similarly, for maximization, we want to find the maximum value of the function in  $x$ :

$$\max z = f(x)$$

## Tutorial Example

In this tutorial, as an example, let us find the maximum of the following function:

$$\max z = f(x) = (8 - x_1) * x_1 + (16 - x_2) * x_2$$

Note that it is a quadratic expression, probably the simplest form of non-linear function. We have chosen this function because it is relatively easy to analyze, so we will use it to get insights on Gradient descent methods.

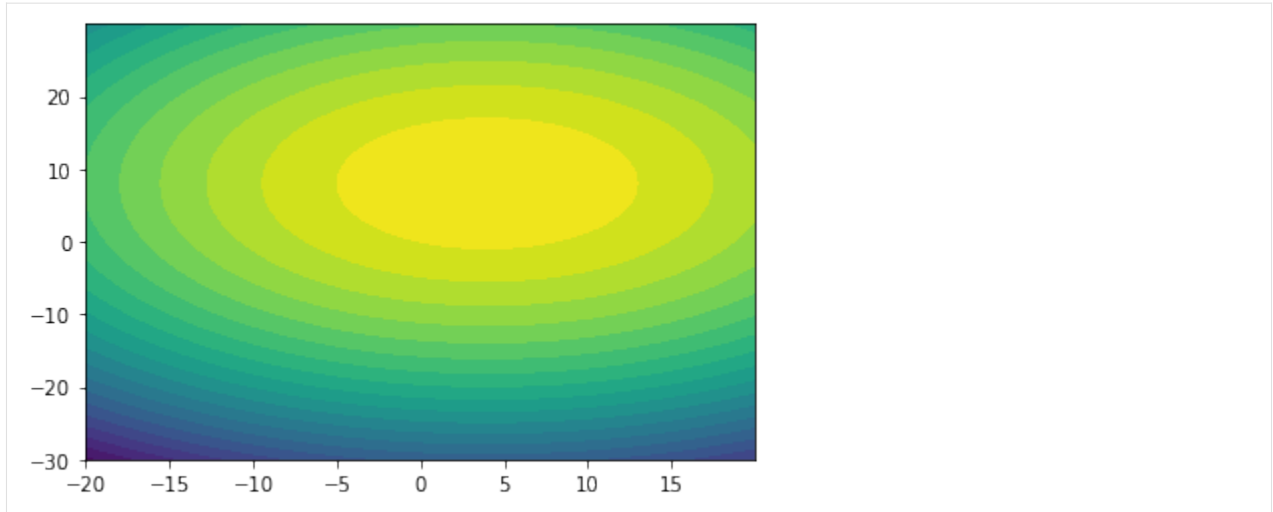
### Implementing the function in Python

The following Python function returns this value, given that you pass  $x$  as a list or array:

```
[ ]: def my_func(x):
    # This is the function to optimize.
    # x is an array containing the dependent variables to optimize (x = [x_1, x_2, ...,
    ↪x_n])
    # By default this is the function used in the example,
    # but you can change it if you want to analyze other examples
    return (8-x[0])*x[0] + (16-x[1])*x[1]
```

Now, let's plot it to get a general idea of how the function looks like. The following script represents a contour plot of the function:

```
[ ]: x_1 = np.arange(-20, 20, 0.01) # The horizontal axis of the contour plot, it will
    ↪represent x_1
    x_2 = np.arange(-30, 30, 0.01) # The vertical axis of the contour plot, it will
    ↪represent x_1
    xx, yy = np.meshgrid(x_1, x_2, sparse=True) # This is a 2D grid of points where the
    ↪function will be evaluated
    z = my_func([xx, yy]) # We then calculate function z
    h = plt.contourf(x_1, x_2, z, levels=20)
```



The yellow area shows the values that are greater. From calculus, we know that the maximum is found at  $x^*$  such that:

$$\nabla f(x^*) = [0, \dots, 0]^T$$

In this example the gradient is a vector of length 2:

$$\nabla f(x) = [8 - 2 * x_1, 16 - 2 * x_2]^T$$

We can easily find the solution analytically with a linear system of equations, one equation for each component of the gradient:

$$8 - 2 * x_1 = 0$$

$$16 - 2 * x_2 = 0$$

So, the maximum is found at:

$$x_1 = 4$$

$$x_2 = 8$$

However, in many applications, particularly in machine learning, it is computationally expensive (we would need a lot of computational resources) to find the solution of  $\nabla f(x^*) = [0, \dots, 0]^T$  analytically. In those cases, gradient-descent methods are a very efficient solution to find an approximate solution.

The strategy of GD methods is based on the following, it is expensive to find the value of  $x$  where the gradient is zero, but it is easy (in computational terms) to compute the value of the gradient for any value of  $x$  (e.g.  $x'$ ). Now, if the value of the gradient at  $x'$  is sufficiently small (for instance, smaller than an arbitrary error  $\epsilon$  in absolute value,  $|\nabla f(x')| \leq [\epsilon, \dots, \epsilon]^T$ , then we can consider that it is a good estimate of the maximum!

For instance, the steepest method starts at an initial  $x'$ , and iteratively, updates this value as:

$$x'' = x' + t * \nabla f(x')$$

where  $t$  is a parameter known as the **learning rate**. The gradient is a vector that tells us the direction of **maximum change** of the function, so the algorithm takes a step in this direction at every iteration (a step of size  $t$ ). When the gradient is “reasonably close to zero”, the algorithm has found an approximation, and if not, it tries updating the value again, making  $x' = x''$  and repeating the calculation above. Intuitively, think that you are in a mountain, and you want to get to the top, but the mountain is really high, and there are trees around you, so you do not really know how to get there. However, at every step, you can move in the direction where the slope is steepest. This is what the simple gradient descent algorithm does. Note that since it is a maximization problem (and we want to get to the top of the mountain), we update  $x$  with a positive increment in the direction of the gradient. For **minimization**, we would update with a negative increment ( $x'' = x' - t * \nabla f(x')$ ).

Define the gradient

Ok, to make things simple and easy to update, let us define another Python function to calculate the gradient. This function will take an array as  $(x = [x_1, x_2, \dots, x_3])$  and return an array with the gradient of the function. By default, it will compute  $\nabla(x) = [\frac{\delta f(x)}{\delta x_1}, \frac{\delta f(x)}{\delta x_2}] = [-2 * x_1 + 8, -2 * x_2 + 16]$ , but you can update it if you would like to test some other function.

```
[ ]: def gradient_func(x):
    # This function returns the gradient as a Numpy array.
    # y will contain the gradient, we initialize it as a zero array.
    # since x has two components, x_1 and x_2, we initialize it as an array
    # with two members:
    y = [0, 0]
    # This is the gradient of the function (8-x_1)*x_1+(16-x_2)*x_2
    # Here we have the first component of the gradient (-2*x_1 +8)
    y[0] = -2*x[0]+8
    # And here the second component (-2*x_2 + 16):
    y[1] = -2*x[1]+16
    return np.array(y)
```

Gradient descent function

Ok, now, let us program another function in Python to run the Simple Gradient Descent algorithm. This function takes the following parameters:

- **start:** The starting value of  $x'$ . This is where our hike to the top (or bottom) of the mountain starts.
- **learn\_rate:** The learning rate  $t$ , or the size of the steps that we are going to take.
- **max\_iter:** Maximum number of iterations, to control that we do not enter and endless loop (or an endless hike..)
- **tol:** This is the tolerance, or the maximum error in the approximation of the gradient  $\epsilon$ . In other words, how close we want to get to the actual top of the mountain. This parameter is optional and by default takes the value 0.01
- **sense:** This is the sense or type of optimization problem. By default, the sense is “max” for maximization problem (to hike uphill to the top of the mountain. You can use `sense="min"` for a minimization problem.
- **gradient:** This is a function that computes the gradient, by default is the function `gradient_func` we have defined before.

The algorithm will return: - **iterations:** An array that contains the value of  $x$  and the error at every iteration.

```
[ ]: def gradient_descent(start, learn_rate, max_iter, tol =0.01, sense="max",
    ↪gradient=gradient_func):
    iterations = [] #Let's keep track of all iterations in a dict
    x = start
    for i in range(max_iter):
        g = gradient(x)
        error = np.absolute(np.array(g))
        iterations.append([np.copy(x), np.copy(error)]) #Append to the iteration, a
    ↪copy of the value and the error obtained

    tolerance = tol*np.ones(error.shape)
    if np.all(error <= tol): #If we are below tolerance, exit the for loop
        break
    step = learn_rate*g
    if sense == "max":
```

(continues on next page)

(continued from previous page)

```

        x += learn_rate*g
    else:
        x -= learn_rate*g
    return iterations

```

### Testing the gradient descent

Ok, Let's test the algorithm starting at  $x = [x_1, x_2] = 0$ , and use  $t = 0.147$  and  $\epsilon = 0.1$ . We will then show the result iterations in a Pandas dataframe so that we can visualize the results in a table.

```

[ ]: ## These are the parameters for the Simple Gradient Descent
start_point = np.array([0.0, 0.0])
t = 0.25
tolerance = 0.1

# Let us compute the iterations
iter = gradient_descent(start_point, t, max_iter=100, tol=tolerance)

# Now, Let's plot the info in a dataframe
pd.DataFrame(iter, columns= ["X", "error"])

```

	X	error
0	[0.0, 0.0]	[8.0, 16.0]
1	[2.0, 4.0]	[4.0, 8.0]
2	[3.0, 6.0]	[2.0, 4.0]
3	[3.5, 7.0]	[1.0, 2.0]
4	[3.75, 7.5]	[0.5, 1.0]
5	[3.875, 7.75]	[0.25, 0.5]
6	[3.9375, 7.875]	[0.125, 0.25]
7	[3.96875, 7.9375]	[0.0625, 0.125]
8	[3.984375, 7.96875]	[0.03125, 0.0625]

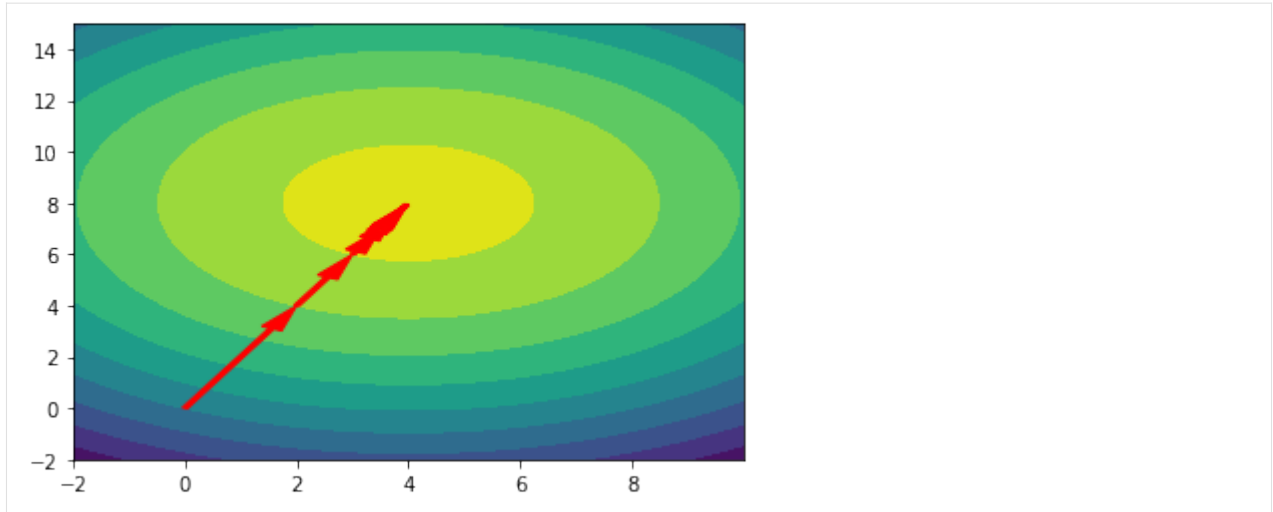
Ok, let us visualize the result in a contour graph. We are going to use red arrows to represent how the different iterations converge to the maximum. Each red arrow starts at  $x'$  and stops at  $x''$  ( $x' \Rightarrow x''$ ).

```

[ ]: x_1 = np.arange(-2, 10, 0.01)
x_2 = np.arange(-2, 15, 0.01)
xx, yy = np.meshgrid(x_1, x_2, sparse=True)
z = my_func([xx, yy])
h = plt.contourf(x_1, x_2, z, levels=10)
for i in range(1, len(iter)):
    start_point = iter[i-1][0]
    end_point = iter[i][0]
    plt.arrow(start_point[0], start_point[1], end_point[0] - start_point[0], end_
    ↪point[1] - start_point[1],
    color='red', width=0.1, head_width=0.4, head_length=1, length_includes_
    ↪head=True)

plt.show()

```



### 4.1.5 Questions

1. Repeat the experimentation with lower tolerances values and note how the number of iterations increases. For instance, try to modify the script to achieve a tolerance of 0.01. How many iterations did the algorithm need, and what is the difference in the estimated  $x$  with respect to the previous experiment. What if you decrease it to 0.001?
2. What happens with the distances between iterations as we get closer to the maximum? Can you reason why this happens?
3. Let us experiment now with the learning rate  $t$ . Let us set the tolerance back to 0.1 and test the script above with a learning rate of 0.25. Is convergence faster (in less iterations)? Try it now with a value of 1. Did the algorithm find an optimal solution? What about 0.5? What can we say about the sensitivity of the algorithm to changes in the learning rate?
4. We know where the maximum solution is. Can we use the second partial derivative test generalization to know if it is a local maximum or minimum?
5. Can you repeat the experiment for another function, for instance, minimize  $z = f(x) = 2x_1^2 + x_2^2 - 10(x_1 + x_2)$ ?

### 4.1.6 Constrained NLP

#### Introduction

Recall that in constrained NLP, the objective is to find the optimal value (maximum or minimum) of an objective function, subject to a set of constraints, where at least one function is not linear. In this section, we will present the canonical form for non-linear problems, and use this form to introduce the Kuhn-Tucker conditions, which are a set of conditions that the optimal value must hold.

### Canonical Form

The canonical form of a **minimisation** problem is defined as:

$$\min f(x)$$

s.t.

$$g_i(x) \geq 0 \quad \forall i = 1, \dots, m$$

where  $x = [x_1, x_2, \dots, x_n]$  is an array with the  $n$  different decision variables of the problem,  $f(x)$  is the (non-linear) objective function and  $g_i(x)$  are the (non-linear) functions of the left hand sides of the  $m$  constraints. Note that in this canonical form, all the constraints are of type *less or equal* and all the right hand sides are 0. It is not required that all the constraints are of the same type, but for now, we will use this definition without loss of generality to present the Kuhn-Tucker conditions.

Similarly, the canonical form of a **maximisation** problem is defined as:

$$\max f(x)$$

s.t.

$$g_i(x) \leq 0 \quad \forall i = 1, \dots, m$$

Note that, besides that and the fact that the type of optimisation function (which is obviously of type maximise) is different, the canonical form is the same as for minimisation problems.

### Kuhn Tucker Conditions

The Kuhn-Tucker (KT) conditions provide a set of differential equations that can be used to find the optimal value of a constrained NLP, based on the **Lagrangian** defined below:

#### Lagrangian

Given an optimisation problem with  $m$  constraints in the canonical form, let us define the **Lagrangian** function as:

$$L(x, \lambda) = f(x) + \sum_{i=1}^m \lambda_i * g_i(x)$$

That is, to take into account the constraints in the objective function, we add the corresponding functions on the right hand side multiplied by a set of coefficients noted as  $\lambda_i$  which are known as Lagrangian multipliers.

#### Gradient condition

Now, for any candidate solution to be an optimal solution, we know that, as for unconstrained NLP, it must be a critical point and thus satisfy:

$$\nabla_x L(x, \lambda) = 0$$

That is, since each component of the gradient is the first order derivative of the corresponding decision variable, the Lagrangian must satisfy:

$$\frac{\delta L}{\delta x_j} = 0 \quad \forall j = [1, \dots, n]$$

**Feasibility condition**

Additionally, we must ensure that the solution is **feasible**, i.e. that meets all the constraints. Therefore, the feasibility condition yields:

$$g_i(x) \leq 0 \quad \forall i = [1, \dots, m]$$

Note that, given the expression of the Lagrangian, this is equivalent to:

$$\frac{\partial L}{\partial \lambda_i} g_i(x) \leq 0 \quad \forall i = [1, \dots, m]$$

**Orthogonality condition**

Now, note that for the Lagrangian and the objective function to represent exactly the point of the function, at the optimal value, the following conditions must be met:

$$\lambda_i * g_i(x) = 0 \quad \forall i = [1, \dots, m]$$

That is, either the Lagrangian multiplier is equal to zero or the corresponding right hand side is equal to zero.

**Non-positive, non-negativity conditions**

Once that the problem is in the canonical form, we also need to check that the Lagrangian multipliers have the correct size, that is, that they are non-negative or non-positive depending on the type of optimisation function and the type of constraint. The following table summarizes the conditions for **minimization problems**:

i-th constraint type	Minimization	Maximization
$\leq$	$\lambda_i \geq 0$	$\lambda_i \leq 0$
$=$	unrestricted	unrestricted
$\geq$	$\lambda_i \leq 0$	$\lambda_i \geq 0$

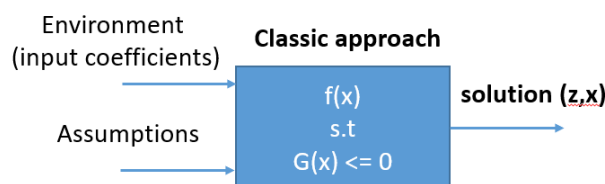
**4.1.7 Introduction to Machine Learning**

**Introduction**

Machine learning is a field of mathematics focused on developing software that solve problems by learning from experience. The main idea behind machine learning is to develop algorithms that mimic the reasoning capability of learning through examples. As explained, in the introduction to operations research, artificial intelligence gathers a set of mathematical tools that allow to solve problems that intelligent beings are able to solve intuitively, but that are a priori complex to solve computationally. One of the most relevant fields of artificial intelligence is machine learning, which focuses on how to solve problems that cannot be precisely specified, or whose solution method cannot be described by symbolic reasoning rules, by using available data.

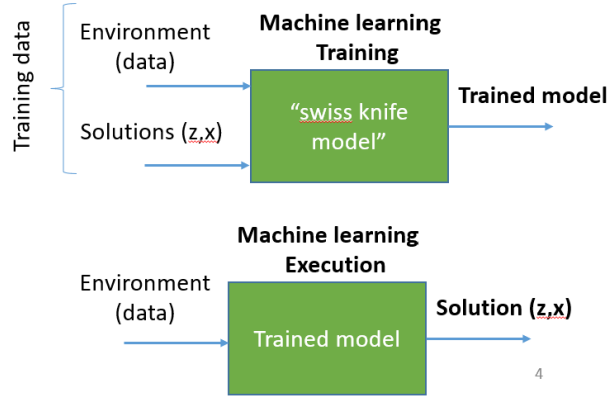
As shown in this unit, machine learning provides tools to reduce problems for which there is no formal mathematical model to an optimization problem through data. Note that this is an important change of paradigm to previous problem-solving approaches, as the starting point is available data, not a mathematical model of the problem.

Let us look illustrate how we have been approaching problem-solving until now with the following figure:



Based on the knowledge of the problem at hand and based on a set of assumptions, so far we have developed a mathematical model of a decision problem as an optimization problem subject to a set of constraints, and use data to set up the coefficients of the model based on the environment framing the decision-making problem. The solution to the problem provided the values of the objective variable and the decision variables.

However, the approach in machine learning is radically different:



Now, what we do is use a *flexible* mathematical model, or *swiss-knife* type of mathematical model, which is in a nutshell any mathematical model which has enough degrees of freedom to model a wide set of problems. For now, we can just assume that we will use models with coefficients that are not defined a priori, and that there are enough of such coefficients to allow it to model many problems. Now, **Training** consists in finding the parameters of the model that provide the best mapping between input data and output data, given a set of examples, or **training data**. With these parameters, the machine learning model **agrees** with the experienced results described in the training data. Out of all possible values for the parameters, the optimal parameters minimize an **error function** between the solutions obtained with the model and the solutions available in the training data.

This way, **learning is reduced to a minimization problem**. There are many machine learning approaches, but in most cases, learning is reduced to a minimization problem and gradient descent methods are behind most machine learning algorithms.

### Core learning tasks

This section describes the core learning tasks, which is how we refer to the types of problems that are normally solved with machine learning. Before we go into the details of each individual learning task, bear in mind that the common denominator is that these are tasks that humans and other type of intelligence beings perform naturally and intuitively, but that do not have a closed mathematical form.

#### Prediction

Learn the behaviour of a variable with time, to make predictions about future trends or events. The variable can be quantitative (e.g. predict the value of a market stock based on a historic dataset) or categorical (e.g. predict the time to failure of a machine)



## Classification and clustering

Learn to classify input data into known categories (classification) or unknown categories (clustering) based on the features available in the training data. For instance, classify images of cells into tumorous or healthy (classification), or learn the different types of customers from collected data.

## Generation

Learn to generate data with the same characteristics of a set of examples. For instance, learn to translate text into a language, or generate videos from images.

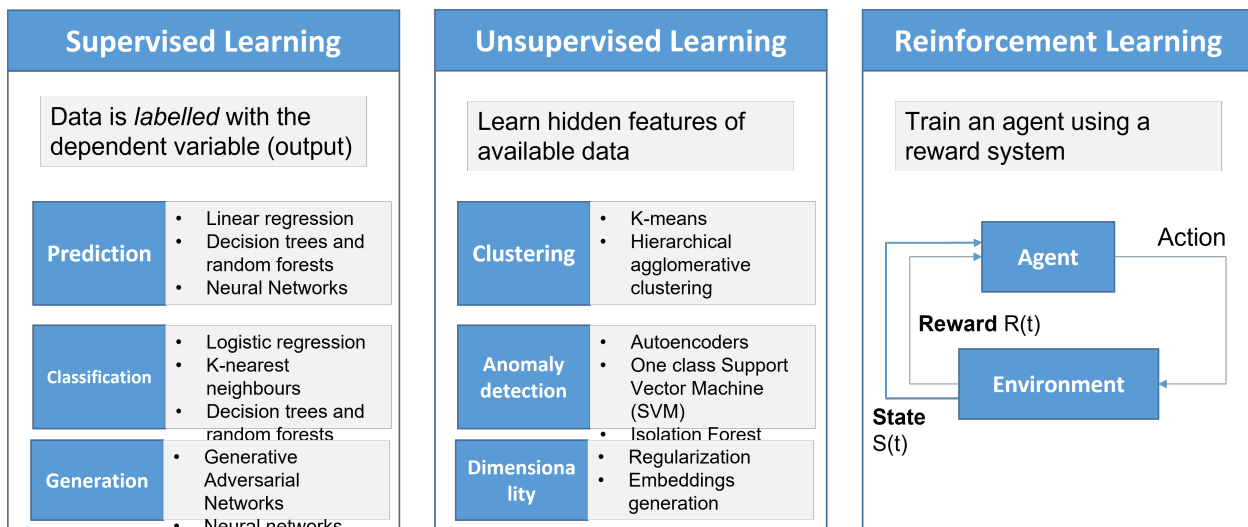
## Optimization

Learn to solve complex optimization problems, either combinatorial optimization problems (e.g. play chess or Go) or convex optimization (drone navigation control) which are very difficult to model and solve with other techniques.

## Types of machine learning techniques

There are three main categories of machine learning techniques:

- **Supervised learning:** In supervised learning, training data contains the outputs of the model, that is, it contains examples of the expected output of the model once it is trained. We refer to the outputs as **dependent variables** and the inputs as the **independent variables**. The common terminology establishes that training data is **labelled** with the dependent variables. That is, each training data sample is associated with the corresponding output obtained when collecting the training data. Supervised learning is applied to prediction, classification, and generation problems.
- **Unsupervised learning:** In unsupervised learning on the other hand, the objective is to learn hidden features of the available data. That is, the output is not explicitly found in the training data, but rather, it is implied and the objective is to learn it from the training data set. Unsupervised learning is used to solve clustering problems, as well as other types of problems like anomaly detection, or subtasks (tasks needed to solve a learning task) like dimensionality (determine how to use the input data).
- **Reinforcement learning:** Reinforcement learning is a different type of machine learning technique based on a reward system. A reinforcement learning agent is a mathematical model of a real world agent performing different actions. Similarly, the environment is modeled as a state machine with a numerable number of possible states. Every action of the agent results in a change of the environment to a given state. For each state, there is an associated reward, and during training, the data is configured to maximize the overall reward obtained.



## 4.1.8 Quadratic Optimization

### Introduction

Experience will soon tell you that amid the wide range of solvers that support NLP, most provide support specifically for quadratic functions. This is because there is a wide range of problems that deal specifically with quadratic functions, rather than with generic non-linear functions. Recall that a quadratic function is a polynomial function of the decision variables with terms of at most second order degree. For instance, an outstanding example of quadratic function is the quadratic error function:

$$Loss(z) = \frac{z^2}{2}$$

Any optimization problem that aims to minimize a given error can use this empirical error function as the objective function. The next subsections present some examples of quadratic optimization problems that use exclusively this type of functions.

### Machine Learning Linear Regression as a Quadratic Unconstrained NLP

#### Supervised Learning Regression Problem Set-Up

The objective of a linear regression problem is to *explain* a response variable, or dependent variable, based on the values of a set of explanatory variables, or independent variables, assuming that the relationship between the response variable and the explanatory variables is linear.

That is, given:

- $x^{(t)}$ : feature vector t. Sample vector t of explanatory variables (e.g. different characteristics of people like gender, age, region, device, ...).  $x^{(t)}$  is a d-dimensional vector (i.e.  $x^{(t)} \in \mathbb{R}$ )
- $y^{(t)}$ : Label t. Response variable t that we want to explain (e.g. for instance the probability that a person is a potential customer or not). Let us assume that every  $y^{(t)}$  is a real value (i.e.  $y^{(t)} \in \mathbb{R}$ )
- $S_n$ : Training set containing all labeled feature vectors, i.e.  $S_n = \{x^{(t)}, y^{(t)} | t = 1, \dots, n\}$

The objective is to learn the linear mapping between the feature vector space x and the response variable y:

$$\hat{y} = f(x, \theta, \theta_0) = \sum_{i=1}^d x * \theta + \theta_0 = x \cdot \theta + \theta_0$$

So that the value of y can be estimated for any point in the vector space x based on the training data set. The parameters to learn are:

- $\theta$ : Linear regression function coefficients
- $\theta_0$ : Linear regression independent term (Let us assume it is zero for now, without loss of generality)

In general, the objective is to minimise the **empirical risk**, noted as  $R_n$ , which is the average of the estimation error at every training point t. Let us for now note Loss the loss function that represents the error at every training point. Then, the objective function is:

$$\min R_n(\theta) = \frac{1}{n} \sum_{t=1}^n Loss(y^{(t)} - x^{(t)} \cdot \theta)$$

Now, if we consider the quadratic error function, the empirical risk becomes:

$$\min R_n(\theta) = \frac{1}{n} \sum_{t=1}^n \frac{(y^{(t)} - x^{(t)} \cdot \theta)^2}{2}$$

This minimisation problem is a quadratic optimization problem where the decision variables are the components of  $\theta$ .

Thus, we can compute the closed form of the gradient as:

$$\begin{aligned} \nabla_{\theta} R_n(\theta) &= \frac{1}{n} \sum_{t=1}^n \nabla_{\theta} \frac{(y^{(t)} - x^{(t)} \cdot \theta)^2}{2} = \frac{1}{n} \sum_{t=1}^n [(y^{(t)} - x^{(t)} \cdot \theta) * (-x^{(t)})] = -\frac{1}{n} \sum_{t=1}^n y^{(t)} * x^{(t)} - \\ &\frac{1}{n} \sum_{t=1}^n (x^{(t)} \cdot \theta) * -x^{(t)} \end{aligned}$$

Now, note that the first term is the sum-product of scalars ( $y^{(t)}$ ) and vectors of size  $d$  ( $x^{(t)}$ ) and therefore, it is a vector. Let us note this first term as  $b$ :

$$b = \frac{1}{n} \sum_{t=1}^n y^{(t)} * x^{(t)}$$

As for the second term, note that is the sum of the scalar product of two vectors of size  $d$  (which is a scalar) times another vector of size  $d$ . We can apply the following transformation:

$$-\frac{1}{n} \sum_{t=1}^n (x^{(t)} \cdot \theta) * -x^{(t)} = \frac{1}{n} \sum_{t=1}^n x^{(t)} * (x^{(t)})^T * \theta = A * \theta$$

That is, first we use the vector product properties to place together the two  $x^{(t)}$  and then we move  $\theta$  outside the summation since it does not depend on  $t$ .

With this transformation, the closed form of the optimisation problem now becomes:

$$\nabla_{\theta} R_n(\theta) = -b + A\theta = 0 \rightarrow A\theta = b$$

### 4.1.9 Machine learning classification as a constrained quadratic optimization problem

Try me



#### Problem definition

In this notebook we are to provide a closed form to reduce a machine learning linear classification problem to a constrained quadratic optimization problem.

In this example, let us imagine that we want to predict the status of an engine by reading sensor data. We do not know what is the mathematical expression that models the status of the machine as a function of the sensor data reading, but we have collected data in different conditions, so we have a **dataset** with the data readings that have been generated by sensors measuring the engine in different states: normal operation, and failure state.

In linear classification problems, we find the following definitions:

- $x^{(t)}$ : feature vector of length  $t$  of independent variables (e.g. in our example this is going to be the different sensor data readings like temperature, rotation speed, torque)  $x$  is a vector of length  $d$  i.e.  $x^{(t)} \in \mathbb{R}^d$ , and each element  $x_1, x_2, \dots$  is going to represent the readings of the different sensors.
- $y^{(t)}$ : Label  $t$  representation the variable that we want to explain from the feature vector (e.g. in our case, whether the engine is in a failure state or not)

The objective is to learn the parameters of a **maximum margin linear separator** from a training dataset. A maximum margin linear separator is a **hyperplane** in the vector space  $X$ . Think of a hyperplane as a plane that divides the vector space into two regions. For instance, a line, which is a 1 dimensional plane, divides a 2-dimensional space into two regions, the area above the line and the area below the line. Consider that a hyperplane in the vector space of size  $d$  can be represented by a vector of size  $d$ . Let us note as  $\hat{\theta} \in \mathbb{R}^d$  the vector that represents our linear separator.

To clearly visualize the results, we are going to work with a linear separator of size  $d = 2$ , so that we can visualize the results in a coordinate system. Therefore, we are only going to have data from two sensors, and our data points are noted as:

$$x = [x_1, x_2]$$

$x_1$ : temperature sensor data readings

$x_2$ : Encoder measuring the engine speed in rpms

### Dataset generation

Now, since we do not really have a dataset, we are going to generate a synthetic one. The following script generates random points and represents them in the bi-dimensional containing all data points.

```
[ ]: import numpy as np
      from matplotlib import pyplot as plt

      ## This parameter determines (half) the number of samples used
      no_samples = 30

      ## This array represents our temperature readings in failure state
      temp_readings_failure = 7*np.random.random_sample((no_samples,1))+29

      ## This array represents our temperature readings in normal state
      temp_readings_normal = 7*np.random.random_sample((no_samples,1))+22

      #This array represents our rpms in failure state
      rpms_failure = 400*np.random.random_sample((no_samples,1))+2900

      #This array represents our rpms in normal state
      rpms_normal = 400*np.random.random_sample((no_samples,1))+2500
```

### Problem representation

Now, let us plot our dataset, together with a hyper-plane which is not necessarily a separator, to illustrate how a line (1-dimensional hyper-plane) divides the 2-dimensional space into two regions:

```
[ ]: #Prepare the figure
      fig, ax = plt.subplots()

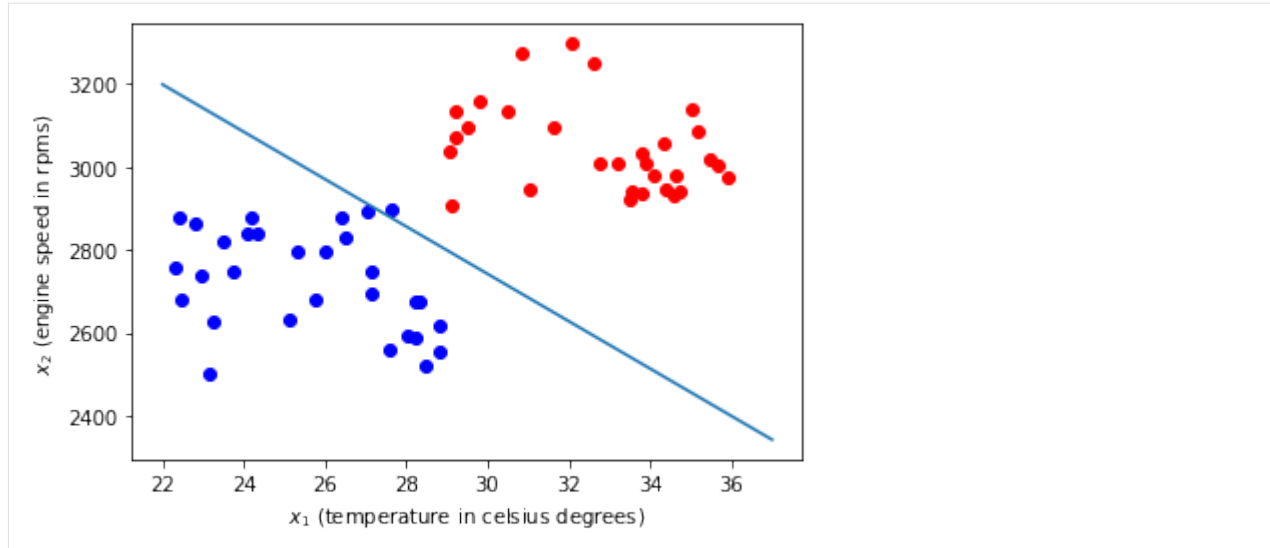
      #Plot the data in failure state in red
      ax.scatter(temp_readings_failure, rpms_failure, color='red')
      #Plot the data in normal operation in blue
      ax.scatter(temp_readings_normal, rpms_normal, color='blue')

      # Make a linear space to plot the temperature
      t = np.linspace(22,37)

      # Initial separator
      thetahat_1 = -57
      thetahat_0 = 4452

      #
      r = thetahat_1*t + thetahat_0
      ax.plot(t,r)
      plt.xlabel('$x_1$ (temperature in celsius degrees)')
      plt.ylabel('$x_2$ (engine speed in rpms)')

      Text(0, 0.5, '$x_2$ (engine speed in rpms)')
```



### How linear separators work (Support Vector Machines)

The line represents a linear separator with parameters  $\hat{\theta} = [57, 1]$ ,  $\hat{\theta}_0 = 4452$ , that is, all points in the line satisfy the following equation:

$$\hat{\theta} \cdot x + \hat{\theta}_0 = 0$$

conform a line (1 dimensional hyperplane) that *classifies* all points of the 2 dimensional space into two categories. The objective is to configure the parameters of this line so that all data points corresponding to the failure state fall into one category, and that all data points corresponding to the normal operation state fall into the other category. This way, we can use the expression  $\hat{\theta} \cdot x + \hat{\theta}_0$  to explain the status of the engine from a reading  $x$ :

$$\hat{\theta} \cdot x + \hat{\theta}_0 \geq 0 \rightarrow \text{the engine is in failure state}$$

$$\hat{\theta} \cdot x + \hat{\theta}_0 \leq 0 \rightarrow \text{the engine is in normal state}$$

We can use the sign operator to provide a numeric value to the labels that represent both status:

$$h(x) = \text{sign}(\hat{\theta} \cdot x + \hat{\theta}_0)$$

If we read new sensor data and multiply it by the coefficients of our linear separator, we can predict the status of the machine as a failure if the sign is positive (the result is greater than zero) and we can predict the status of the machine as normal state if the result is negative. This type of classifiers are also known as **Support Vector Machines**.

### Learning the parameters of an SVM

#### The training dataset

Now, the objective is to learn the parameters of our separator from the dataset, a set of historic readings from which we know the status of the machine. Let a sample  $t$  of our data set be noted as:

- $x^{(t)} = [x_1^{(t)}, x_2^{(t)}]$  temperature sensor and rpm readings at sample  $t$
- $y^{(t)}$  status at sample  $t$  {-1 if machine is in normal state, +1 if machine is in failure state}

We are going to have many data readings, some of them in normal state, and some in failure state, and we are going to use these data to find the parameters of a separator that divides the space into two regions based on the information that we have in the database we have collected.

Let us note our training data set as:

$$S_n = [x^{(t)}, y^{(t)}] \forall t \in [1, \dots, n]$$

### Reducing the learning problem to an optimization problem

Now, given that we are going to use the sign of the result to determine the status of the machine, every separator must ensure that:

$$\begin{aligned} \hat{\theta} \cdot x^{(t)} + \hat{\theta}_0 &\geq 0 \quad \forall y^{(t)} = 1 \\ \hat{\theta} \cdot x^{(t)} + \hat{\theta}_0 &\leq 0 \quad \forall y^{(t)} = -1 \end{aligned}$$

There might be many lines that satisfy these constraints, or there might be actually none, that is, the space might not be separable by a line. If the space can be separated by a line, the objective is to find a hard margin separator. Note that it is beneficial to maximise the **margin** of our separator, that is, the distance between the line and the closest points above and below it. Given that there is uncertainty on the information we have collected (for instance, we might not have gathered data in every possible situation), it is better to find a margin that maximises the distance between the data clusters in the two regions. In fact, this is what a human will solve the problem. This is known as the maximum margin SVM, i.e., to find the linear hyperplane with the maximum margin, such that all points are classified correctly. Furthermore, the parameters  $\hat{\theta}$  and  $\hat{\theta}_0$  of the maximum margin separator satisfy the following constraint:

$$y^{(t)} * (\hat{\theta} \cdot x^{(t)} + \hat{\theta}_0) \geq 1 \quad \forall t \in [1, \dots, n]$$

For convenience, we have set up the minimum distance to the separator to 1. This yields that the margin of our hyperplane is given by  $\frac{2}{\|\hat{\theta}\|}$ . For instance, in our example, imagine that point  $x_A$  is the closest point above the line ( $y^A = 1$ ) and point  $x_B$  is the closest point below the line ( $y^B = -1$ ). Let us now define a line that is parallel to the separator and passes through  $x^A$ :

$$\begin{aligned} \hat{\theta} \cdot x^A + \hat{\theta}_0 &= 1 \\ \hat{\theta}_0 \cdot x_0^A + \hat{\theta}_1 \cdot x_1^A + \hat{\theta}_0 - 1 &= 0 \end{aligned}$$

and another one that is also parallel to the separator and passes through  $x^B = -1$ :

$$\begin{aligned} -\hat{\theta} \cdot x^B - \hat{\theta}_0 &= 1 \\ \hat{\theta}_1 \cdot x_1^B + \hat{\theta}_1 \cdot x_2^B + \hat{\theta}_0 + 1 &= 0 \end{aligned}$$

The distance between both parallel lines is noted as  $\omega$  and is given by:

$$\omega = \frac{|(\hat{\theta}_0+1) - (\hat{\theta}_0-1)|}{\sqrt{\hat{\theta}_1^2 + \hat{\theta}_2^2}} = \frac{2}{\|\hat{\theta}\|}$$

Therefore, our objective function is to maximise this expression, or equivalently, to minimise  $\frac{\|\hat{\theta}\|}{2}$ , which is a function of the components of the vector  $\hat{\theta}$  and  $\hat{\theta}_0$ .

With this, our non-linear problem becomes:

$$\min \frac{\sqrt{\hat{\theta}_1^2 + \hat{\theta}_2^2}}{2}$$

subject to:

$$y^{(t)} * (\hat{\theta} \cdot x^{(t)} + \hat{\theta}_0) \geq 1 \quad \forall t \in [1, \dots, n]$$

Or equivalently

$$\min \frac{\sqrt{\hat{\theta}_1^2 + \hat{\theta}_2^2}}{2}$$

subject to:

$$\begin{aligned} \hat{\theta} \cdot x^{(t)} + \hat{\theta}_0 &\geq 1 \quad \forall y^{(t)} = 1 \\ \hat{\theta} \cdot x^{(t)} + \hat{\theta}_0 &\leq -1 \quad \forall y^{(t)} = -1 \end{aligned}$$

## SVMs in Python

Now, we are going to use two different approaches in Python to find the maximum margin separator for our dataset, using the package for data science [Scipy](#).

### Using non-linear programming

In the first approach we are going to use the package **optimize** for optimization. More specifically, we are going to use the [LinearConstraint](#), the [Bounds](#) modules, and the [minimize](#) function. The script below imports the dependencies and packs the features into a matrix.

```
[ ]: from scipy.optimize import LinearConstraint, Bounds, minimize

# Pack the 2 sample arrays into 1 array using np.append
failure_features = np.append(temp_readings_failure, rpms_failure, 1)
normal_features = np.append(temp_readings_normal, rpms_normal, 1)
```

### Minimize

The `scipy.optimize.minimize` function takes the following parameters:

- `fun`: A callable function that provides the objective function as a function of a one dimensional array, where each element of the array is a decision variable.
- `x0`: An initial guess, or initial values for our decision variables
- `method`: (Optional) The algorithm used for optimization
- `constraints`: List or dictionary containing the constraints for the optimization problem.

Given that our problem has constraints, the function will select the [L-BFGS-M](#) algorithm.

Let us first define the objective function to minimize:

```
[ ]: def objective_func(theta):
      return 1/2*(theta[0]**2 + theta[1]**2)**(1/2)
```

### Constraints

Now, for the constraints, we can either introduce linear constraints or non-linear constraints. The linear constraints are instances of `scipy.optimize.LinearConstraint`. This function has the following parameters:

- `A`: Matrix defining the coefficients in the left-hand-sides of the constraints
- `ub`: A vector defining the upper bound of the constraints or a scalar defining the upper bound for all the constraints
- `lb`: A vector defining the lower bound of the constraints or a scalar defining the lower bound for all the constraints

Note that, playing with the values provided to `ub` and `lb` it is possible to defines constraints of type less or equal, greater or equal, or equal. With this function, the following code snippet defines the set of constraints corresponding to the failure state data set samples

```
[ ]: # The right hand sides are x_1^(t), x_2^(t), 1, so we append 1s to the failure_
      ↪ features to obtain all right hand sides:
failure_rhs_coefs = np.append(failure_features, np.array(np.ones((no_samples, 1))), 1)
# For the label +1, we do not need to make any change to the RHS, the lower bound is_
```

(continues on next page)

(continued from previous page)

```

↪1, i.e.  $1 \leq x_1(i) * \theta_1 + x_2(i) * \theta_2 + \theta_0 \leq \text{inf}$ 
failure_linear_constraints = LinearConstraint(failure_rhs_coefs, lb=1, ub=np.inf)
    
```

And the following code snippet to define the set of constraints for the normal state condition

```

[ ]: normal_rhs_coefs = np.append(normal_features, np.array(np.ones((no_samples,1))),1)
normal_rhs_coefs = normal_rhs_coefs
# For the label +1, we do not need to make any change to the RHS, the lower bound is
↪1, i.e.  $-\text{np.inf} \leq x_1(i) * \theta_1 + x_2(i) * \theta_2 + \theta_0 \leq -1$ 
normal_linear_constraints = LinearConstraint(normal_rhs_coefs, lb=-np.inf, ub=-1)
    
```

### Example

Now, we have everything that we need to build our separator! let us use the initial separator at the beginning of the function as the initial guess:

```

[ ]: theta_hat_0 = np.array([57, 1, -4452])
linear_constraint = [failure_linear_constraints, normal_linear_constraints]
print(len(linear_constraint))
# x_0 = [ 1.13792022e+00,  5.42842896e-03, -4.83177976e+01]
res = minimize(objective_func, theta_hat_0, constraints=linear_constraint,
               options={"maxiter":1000, "disp": True})
print(res)

theta = res.x

#Prepare the figure
fig, ax = plt.subplots()

#Plot the data in failure state in red
ax.scatter(temp_readings_failure, rpms_failure, color='red')
#Plot the data in normal operation in blue
ax.scatter(temp_readings_normal, rpms_normal, color='blue')

# Make a linear space to plot the temperature
t = np.linspace(22,37)

plt.xlabel('$x_1$ (temperature in celsius degrees)')
plt.ylabel('$x_2$ (engine speed in rpms)')

r_mms = (-theta[0] /theta[1])*t - (theta[2] / theta[1])
ax.plot(t, r_mms)

mod = (abs(theta[0]**2 + theta[1]**2))**-1/2
print(mod)
r_mms_max = (-theta[0] /theta[1])*t + (-theta[2] + 1)/ theta[1]

r_mms_min = (-theta[0] /theta[1])*t + (-theta[2] - 1)/ theta[1]

ax.plot(t, r_mms_min, color='red')
ax.plot(t, r_mms_max, color='black')
    
```

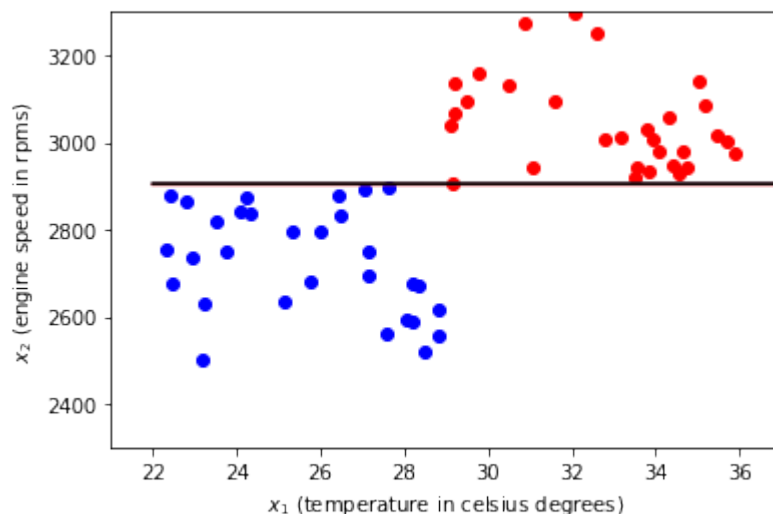
(continues on next page)



(continued from previous page)

```
ax.set_xlim(21, 37)
ax.set_ylim(2300, 3300)
print(np.min(np.sum(failure_rhs_coefs*theta,axis=1)))
```

```
2
Optimization terminated successfully.      (Exit mode 0)
      Current function value: 0.7663190555426902
      Iterations: 10
      Function evaluations: 51
      Gradient evaluations: 10
     fun: 0.7663190555426902
     jac: array([0.00503157, 0.49997468, 0.          ])
message: 'Optimization terminated successfully.'
     nfev: 51
      nit: 10
     njev: 10
    status: 0
   success: True
         x: array([ 1.54231359e-02,  1.53256051e+00, -4.45199915e+03])
0.2128583851271692
1.0
```



Alternative solution: Adding a regulation parameter

Note that the separator might not be the same as a human would have selected when solving this task. This is because the algorithm is quite sensitive to the initial guess passed as argument, and it can converge to a local minimum. To make the algorithm more resilient, we can modify slightly the objective function to factor in  $\theta_0$ :

```
[ ]: def objective_func(theta):
      return 1/2*(theta[0]**2 + theta[1]**2 + theta[2]**2)**(1/2)
```

With this regularization, we assume by looking at the data that the separator we are looking for crosses the y-axis at a high value, and therefore, we include it in the objective function. Now, if we try again with this objective function, we get a support vector machine which is closer to the solution that a human would provide:

```
[ ]: theta_hat_0 = np.array([57, 1, -4452])
linear_constraint = [failure_linear_constraints, normal_linear_constraints]
print(len(linear_constraint))
# x_0 = [ 1.13792022e+00,  5.42842896e-03, -4.83177976e+01]
res = minimize(objective_func, theta_hat_0, constraints=linear_constraint,
               options={"maxiter":1000, "disp": True})
print(res)

theta_prime = res.x

#Prepare the figure
fig, ax = plt.subplots()

#Plot the data in failure state in red
ax.scatter(temp_readings_failure, rpms_failure, color='red')
#Plot the data in normal operation in blue
ax.scatter(temp_readings_normal, rpms_normal, color='blue')

# Make a linear space to plot the temperature
t = np.linspace(22,37)

plt.xlabel('$x_1$ (temperature in celsius degrees)')
plt.ylabel('$x_2$ (engine speed in rpms)')

r_mms = (-theta_prime[0] /theta_prime[1])*t - (theta_prime[2] / theta_prime[1])
ax.plot(t, r_mms)

mod = (abs(theta[0]**2 + theta[1]**2))**-1/2
print(mod)
r_mms_max = (-theta_prime[0] /theta_prime[1])*t + (-theta_prime[2] + 1)/ theta_
↪prime[1]

r_mms_min = (-theta_prime[0] /theta_prime[1])*t + (-theta_prime[2] - 1)/ theta_
↪prime[1]

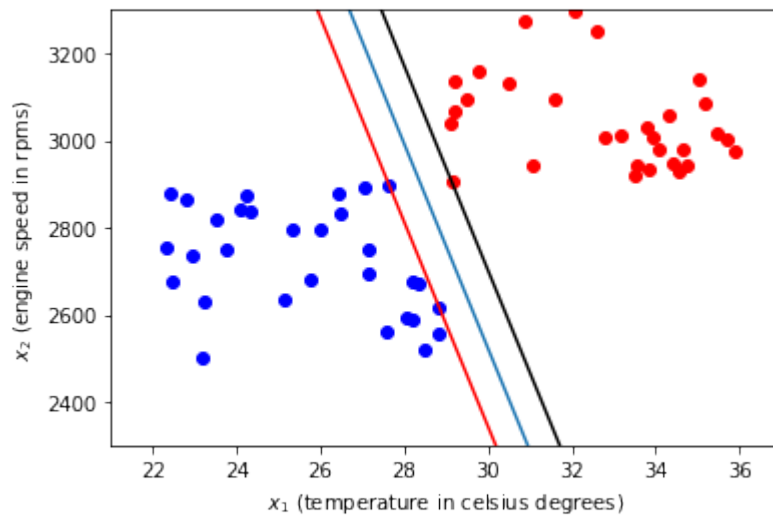
ax.plot(t, r_mms_min, color='red')
ax.plot(t, r_mms_max, color='black')
ax.set_xlim(21, 37)
ax.set_ylim(2300, 3300)
print(np.min(np.sum(failure_rhs_coefs*theta,axis=1)))

2
Optimization terminated successfully.      (Exit mode 0)
      Current function value: 26.689281097128372
      Iterations: 8
      Function evaluations: 40
      Gradient evaluations: 8
     fun: 26.689281097128372
     jac: array([ 1.22678280e-02,  5.22136688e-05, -4.99849558e-01])
  message: 'Optimization terminated successfully.'
     nfev: 40
      nit: 8
     njev: 8
    status: 0
   success: True
         x: array([ 1.30967158e+00,  5.57996292e-03, -5.33624927e+01])
```

(continues on next page)

(continued from previous page)

```
0.2128583851271692
1.0
```



## Using Scipy

### SVM

Luckily for us, we do not need to worry about in practice because Scipy packs an SVM model

```
[ ]: from matplotlib.axis import YTick
      from sklearn import svm

      clf_svm = svm.SVC(kernel="linear")
      X = np.append(failure_features, normal_features, axis=0)
      #y is a vector of ones
      y = np.ones((2*no_samples,1))
      # make the normal features negative:
      y[no_samples:] = -1

      clf_svm.fit(X,y)

      xx = np.linspace(21, 37, no_samples)
      yy = np.linspace(2300, 3300, no_samples)
      YY, XX = np.meshgrid(yy,xx)
      xy = np.vstack([XX.ravel(), YY.ravel()]).T
      Z = clf_svm.decision_function(xy).reshape(XX.shape)

      #Prepare the figure
      fig, ax = plt.subplots()

      #Plot the data in failure state in red
      ax.scatter(temp_readings_failure, rpms_failure, color='red')
      #Plot the data in normal operation in blue
      ax.scatter(temp_readings_normal, rpms_normal, color='blue')

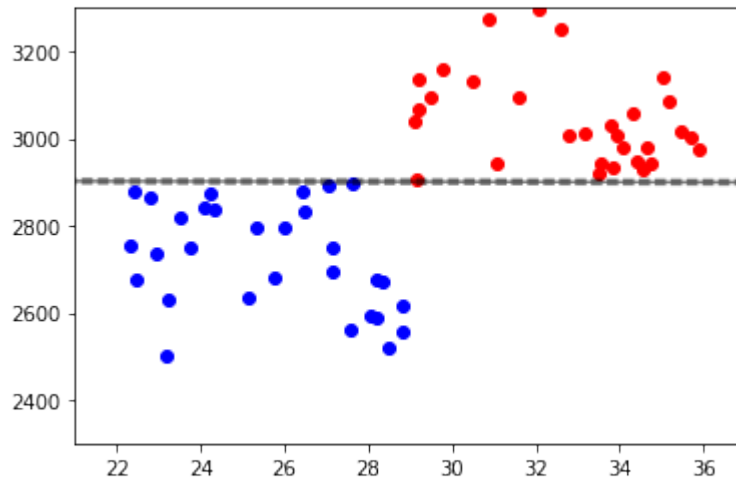
      ax.contour( XX, YY, Z, colors="k", levels=[-1, 0, 1], alpha=0.5, linestyles=["--", "-
      ↪", "--"])
```

(continues on next page)

(continued from previous page)

```
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993:
↳DataConversionWarning: A column-vector y was passed when a 1d array was expected.
↳Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```



### Stochastic Gradient Descent

Also, the library implements a Stochastic Gradient Descent model:

```
[ ]: from sklearn.linear_model import SGDClassifier
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

X = np.append(failure_features, normal_features, axis=0)
#y is a vector of ones
y = np.ones((2*no_samples,1))
# make the normal features negative:
y[no_samples:] = -1

clf_sgd = make_pipeline(StandardScaler(), SGDClassifier())

clf_sgd.fit(X, y)

xx = np.linspace(21, 37, no_samples)
yy = np.linspace(2300, 3300, no_samples)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = clf_sgd.decision_function(xy).reshape(XX.shape)
#Prepare the figure
fig, ax = plt.subplots()

#Plot the data in failure state in red
ax.scatter(temp_readings_failure, rpms_failure, color='red')
#Plot the data in normal operation in blue
```

(continues on next page)

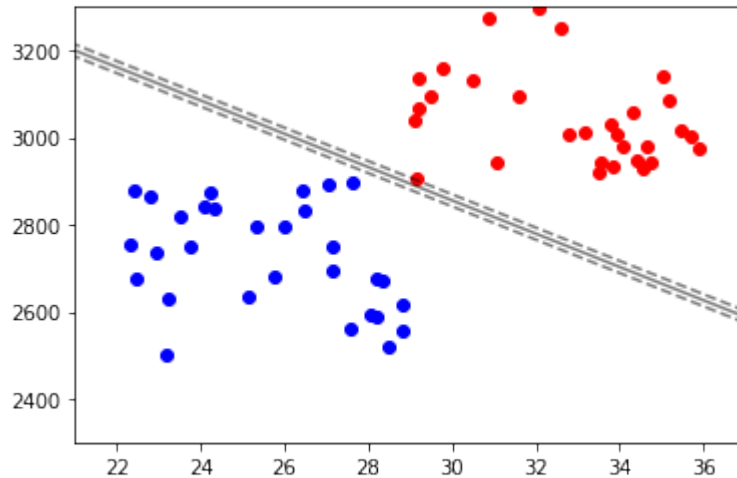
(continued from previous page)

```
ax.scatter(temp_readings_normal, rpms_normal, color='blue')

ax.contour( XX, YY, Z, colors="k", levels=[-1, 0, 1], alpha=0.5, linestyles=["--", "->", "--"])

plt.show()

/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993:
↳DataConversionWarning: A column-vector y was passed when a 1d array was expected.
↳Please change the shape of y to (n_samples, ), for example using ravel().
    y = column_or_1d(y, warn=True)
```



1. Looking at the shapes of the different separators, which one do you think performed best given the training data? Why?
2. Note that the variable `no_samples` defined in the first cell defines the number of samples. Increase the value of samples to 3000 and run all the script again to see what happens. What are your main conclusions of this experiment?

Set again the number of samples to 30 to answer the rest of the questions. 3. Note that in the **Using Linear Programming Section**, the variable `theta` has the parameters of the first linear separator, and the variable `theta_prime` the parameter of the second linear separator. Recall that the margin is defined as  $\omega = \frac{|\hat{\theta}_0+1)-(\hat{\theta}_0-1)|}{\sqrt{\hat{\theta}_1^2+\hat{\theta}_2^2}} = \frac{2}{\|\hat{\theta}\|}$ . Calculate the margin of the two linear separators. Which one has the largest margin? Why?

```
[ ]: print("Parameters of NLP estimator:")
print(theta)
print("Parameters of NLP estimator with regularization:")
print(theta_prime)
```

[ ]:

4. Now, imagine we want to predict the status of the machine for the following points, given as pairs of temperature, rpms (temperature, rpm):
  - (32, 2600)
  - (34, 2600)

- (36, 2600)

Note that to get the results with the two NLP linear separators, you need to calculate if the point is over or below the separators as, as:

- $\theta_0 \cdot \text{temp} + \theta_1 \cdot \text{rpm} + \theta_2 \geq 1 \rightarrow \text{predict failure}$
- $\theta_0 \cdot \text{temp} + \theta_1 \cdot \text{rpm} + \theta_2 \leq -1 \rightarrow \text{predict normal state}$

Test both linear separators in these provided points. Looking at the data, which one do you think provides more reliable results?

[ ]:

5. Now, to predict values with the Scipy separators, you need to use the method `predict()` of the predictor. For instance, the following script uses the SVM predictor to predict the values at 24 degrees and 2200 rpms and 28 degrees and 3300 rpms:

```
[ ]: res = clf_svm.predict([[24, 2200], [28, 3300]])
print(res)
[-1.  1.]
```

Write down a script to predict the values at the same points as in exercise 4, but using the Scipy models you have trained. What are the main conclusions?

## 4.2 Exercises

In this section you have a collection of CLP problems sorted by difficulty:

- **Easy problems:** You can tackle these problems after the first lesson on NLP.
- **Normal problems:** These problems require that you are familiar with concepts like Kuhn Tucker conditions.
- **Hard problems:** This is the type of problems that definitively get you ready for the exam.

### 4.2.1 Planning the Construction of homes

On a housing estate on the Alicante coastline, two types of homes are being built: apartments and penthouses, whose prices are  $p_1$  and  $p_2$  M€ respectively. The curve of demand for apartments is  $d_1 = 40 - 2p_1$  and  $d_2 = 150 - 3p_2$  for penthouses. The builder calculated that, owing to the orders that he has already sent to his raw materials suppliers, it is worth building 15 times more apartments than penthouses. He has also calculated that building an apartment costs him 5M€, while a penthouse costs him 3M€. Knowing that the builder has a budget of 350 M€, work out the following:

- a) Write an NLP to calculate the optimal prices for apartments and penthouses.
- b) Write the Kuhn Tucker conditions of the NLP problem

### 4.2.2 Car Production Planning

The Operations Department of DINOCO automobile manufacturing and assembly firm must establish the daily production quantities of two car models (SUV and Compact) which maximise profits according to the production capacity constraints. The firm also wishes to respect the environment, and in the future to benefit its sales, given its commitment to increase the FFM (Fleet Fuel Mileage) indicator; that is, the mileage efficiency per litre of fuel of produced fleet.

To simplify this study case, a linear relation between the sale price and the quantity to be produced is assumed, given by the following function:  $q_i = a_i - b_i \cdot p_i$ , where  $a_i$  and  $b_i$  are constants provided in the table below, and  $p_i$  is the mean unit price per family of cars in K€. Parameter  $C_i$  is also provided, which represents the cost per unit for the cars of family  $i$ .

The only productive resource that restricts the system is the assembly line. It is estimated that the firm spends 1 h every day on assembling SUV and Compact cars. The time required ( $R_i$ ) to produce one unit of each family of cars and the remaining former data are shown in the table.

i	Car model	a	b	C (K€)	R(seconds)	Kmpl
1	SUV	440	20	17	54	12
2	Compact	248	12	13	56	20

The FFM indicator, which must be at least 15 Kmpl, is calculated by using the harmonic mean:

$$FFM = \frac{q_1 + q_2}{\frac{q_1}{kmpl_1} + \frac{q_2}{kmpl_2}}$$

- a) Formulate a non-linear programming model to help the Operations Department establish an optimum production plan which meets the existing constraints
- b) Find the Kuhn-Tucker conditions
- c) A solver provides the following solution:  $q_1 = 31.14$ ,  $q_2 = 34.26$ ,  $\lambda_1 = -0.034$ ,  $\lambda_2 = 0$ , where  $\lambda_1$  is the Lagrangian multiplier corresponding to the capacity constraint and  $\lambda_2$  is the Lagrangian multiplier of the FFM constraint. Interpret the results.

### 4.2.3 Nova Case

#### Problem Definition

You work for Nova case, a start-up company that manufactures protective cases for mobile phones made of a novel material. The marketing department of your company has estimated a formula for the optimal price elasticity of demand for the two different models you are currently selling online. Recall that the price elasticity of demand represents the relationship between the quantity and the price of a good or service. They have estimated that, for both products, the price elasticity follows the following expression:

$$LQ = K + E \cdot LP$$

Where LQ is the logarithm of the quantity, LP is the logarithm of the price, K is the demand constant and E the demand elasticity. You are in charge of the production department and need to calculate the optimal production mix one planning period (1 day), taking into account this relationship between price and quantity. You have local production, and the most restrictive resource is the moulding machine which has different cycle time (in seconds) for each case. You also have a limited supply of the novel material and each model uses a different quantity of this material. In summary, this is the data of your first product:

K1 = 4 E1 = 2 cycle time = 30 seconds key material quantity = 45 grams

And for your second product: K2 = 8 E2 = 2 cycle time = 25 seconds key material quantity = 40 grams

The total operational time of the moulding machine in the planning period is 6.5 hours and the supplier can provide you up to 40 kilograms per day of the novel material.

- a. Write down a Non-Linear Programming problem to calculate the optimal production mix.
- b. Write down the Kuhn Tucker conditions.
- c. Assume that you need to take into account in your model the initial stock level of each product at the beginning of the planning period. How does this affect your model? Motivate your response
- d. What would it mean if, in a Kuh Tucker point, the Lagrangian multiplier of the material quantity constraint was equal to zero?

### 4.2.4 Maximising investments of a startup

#### Problem Definition

Your startup company has just raised 100M€ in a round of funding to reinvest in the company during the next period. Using regression and historical data, your team has found the following formulas for the Return of Investments (ROI) in different departments:

#### Research and Development:

$$10 \cdot (\sqrt[3]{2X} - 2)$$

#### Advertisement:

$$10 \cdot (\ln(X) - 2)$$

#### Technical Equipment:

$$10 \cdot (e^{X/50} - 2)$$

#### Human Resources:

$$10 \cdot \frac{X}{50}$$

According to the collective agreement, the minimum investment in human resources is 20M€. Additionally, analysing the formulas obtained, it is clear that although significant investments in technical equipment yield a high ROI, for small investments the ROI of R&D or advertisement is actually larger. However, due to the pressure imposed by competitors, the company has decided that investments in technical equipment must be at least high enough to ensure that this concept does not have a negative impact in the ROI of the next period investments.

- a. Write a Non-Linear Programming Model to maximise the ROI
- b. Write down the Kuhn-Tucker conditions
- c. Solving the problem yields a value of -0.29 for the lagrangian multiplier of the minimum investment in human resources constraint. Interpret this result. How should the company use this value in the negotiation of the next collective agreement?

### 4.2.5 Planning refreshment production

A firm that packs refreshments employs the same syrup to produce its 1.5 l COLI and PEPSA products on its S1 production line. Once processed, each hectolitre of syrup produces 40 units of the 1.5 l COLI product or 20 units of the 1.5 l PEPSA product. If  $x_1$  units of the 1.5 l COLI product and  $x_2$  units of the 1.5 l PEPSA product are produced, the firm estimates that the daily income obtained in dollars would be given by the following function:

$$f(x) = 49000 \cdot x_1 - x_1^2 + 30 \cdot x_2 - 2 \cdot x_2^2$$

It costs 150 dollars to buy and process each hectolitre of syrup. The S1 packaging line has a net capacity of producing 56,800 1.5 l product units every day. The firm is committed to produce at least half the amount of PEPSA than COLI. Although priority orders tend to amend its production planning, the firm wishes to have a basic product planning that optimises its daily profits.



- a) Formulate a non-linear programming model that helps the firm create its basic daily product planning for its S1 packaging line.
- b) Write down the Kuhn Tucker conditions that would help the company determine the basic daily production plan.
- c) The following critical point is found  $x_1 = 13838.33; x_2 = 6919.16; \lambda_1 = 0; \lambda_2 = 15323.34; z = 287249204.2$ , where  $\lambda_1$  is the Lagrangian multiplier corresponding to the S1 packaging line capacity constraint. Can you conclude whether it is a local/global maximum? With this solution, What is the maximum quantity that the firm would be willing to pay for 1 h of overtime production? How much would the daily profit increase per additional COLI unit produced?

## 4.2.6 Production Mix of Pharma

### Problem definition

Cheston Pharmaceuticals manufactures two different products. The company has conducted different studies to estimate the demand curve of their products, to learn the relationship between price and demand. The company has build a linear model that has proven to be accurate enough in the price range of interest for the company. Let us note the linear model as:

$$d_j = -\theta_j * p_j + \hat{\theta}_j \quad p_j = [m_j, M_j], \forall j = [1, 2]$$

Where  $d_j$  is the total units of product  $j$  manufactured,  $p_j$  is the price of product  $j$  and  $\theta_j$  and  $\hat{\theta}_j$  are the parameters of the linear models for the curve of demand of product  $j$ . The price of product  $j$  must be in the range of interest, that is, greater or equal than the minimum price  $m_j$  and lower or equal than the maximum  $M_j$ .

Both products incorporate a critical compound for which the company has limited resources to produce. The total production capacity of the company for the compound in the planning period is C (milligrams). A unit of product  $j$  contains  $r_j$  milligrams of the critical compound.

Let us assume that the production costs of product  $j$  can be neglected.

- a. Write a Non-Linear Programming problem to calculate the optimal production mix (number of units to produce of each product) that maximises the revenues in the planning period.
- b Write down the Lagrangian
- c Write down the Kuhn-Tucker conditions:
- d. Write down the Hessian matrix and the Hessian determinants. In case that you find a critical point, can you determine if it is a local or a global maximum? Motivate your response.

## 4.2.7 Optimising the design of headphones

### Problem definition

You work on the R&D department of Banshee Ltd., a company that manufactures headphones. Through a customer survey you have determined that the level of satisfaction of your headphones is given by the following expression:

$$z = 5 + \ln(x_1) + x_2^2 + x_1 * x_2$$

where  $x_1$  is the thickness of the diaphragm in microns and  $x_2$  is the radius of the diaphragm in millimeters. Due to design constraints, the volume of the driver, which can be estimated as  $0.2 * \pi * x_2 * x_1$  cannot be larger than 0.172 cubic millimeters. The properties of the material you use makes it unfeasible to manufacture a diaphragm with a thickness lower than 0.3 microns or higher than 0.9 microns.

- a Write a Non-Linear Programming (NLP) problem to find the optimal design specifications (thickness and radius of diaphragm) for Banshee Ltd.
- b Write down the Lagrangian and the Kuhn Tucker conditions

c A solver gives you the following solution ( $x_1 = 0.9, x_2 = 0.304$ ). Determine if it is a local or global maximum

## 4.2.8 Mathematical Nirvana

### Problem Definition

Nathaniel Richards is a young, brilliant scientist that has developed the following Non-Linear Programming (NLP) problem to find the optimal balance between studying and meditating to maximize the overall satisfaction and achieve a state of enlightenment (Mathematical Nirvana).

**Decision Variables:** Let:

$x_1$  Time spent studying (in hours)

$x_2$  = Time spent meditating (in hours)

$x = [x_1, x_2]$  The set of decision variables

$x_1, x_2 \geq 0$

**Objective Function:** Maximize the overall satisfaction obtained from studying and meditating:

$$\max z = f(x) = 2 * x_1 + 0.5 * \ln(1 + x_1) + 0.7 * x_2 + 0.3 * \sqrt{x_2}$$

**Constraints:** Subject to: Maximum amount of time available:

$$x_1 + x_2 \leq 10$$

Minimum amount of time studying to ensure academic performance:

$$x_1 \geq 2$$

Unfortunately Nathaniel mysteriously disappeared before he could completely analyse the problem, so you need to complete his work according to the following instructions:

1. Obtain the Kuhn-Tucker conditions
2. Obtain the Hessian and determine if this solution ( $x_1 = 3.9, x_2 = 6.1$ ) is a global or local maximum
3. Use the Kuhn-Tucker conditions to calculate the Lagrangian multipliers for this solution, can you explain what they mean? Discuss if this can be an optimal solution to the problem

## 4.3 Solved Exercises

In this section you have the solution to the different exercises.

### 4.3.1 Planning the Construction of homes

On a housing estate on the Alicante coastline, two types of homes are being built: apartments and penthouses, whose prices are  $p_1$  and  $p_2$  M€ respectively. The curve of demand for apartments is  $d_1 = 40 - 2p_1$  and  $d_2 = 150 - 3p_2$  for penthouses. The builder calculated that, owing to the orders that he has already sent to his raw materials suppliers, it is worth building 15 times more apartments than penthouses. He has also calculated that building an apartment costs him 5M€, while a penthouse costs him 3M€. Knowing that the builder has a budget of 350 M€, work out the following:

a) Write an NLP to calculate the optimal prices for apartments and penthouses.

$$\max z = (p_1 - 5)(40 - 2p_1) + (p_2 - 3)(150 - 3p_2) = -2p_1^2 - 3p_2^2 + 50p_1 + 159p_2 - 650$$

s.t

$$5(40 - 2p_1) + 3(150 - 3p_2) \leq 350 \rightarrow -10p_1 - 9p_2 + 300 \leq 0$$

$$15(150 - 3p_2) \leq (40 - 2p_1) \rightarrow 2p_1 - 45p_2 + 2210 \leq 0$$

$$p_1, p_2 \geq 0$$

**b)** Write the Kuhn Tucker conditions of the NLP problem:

The Lagrangian is:

$$L = -2p_1^2 - 3p_2^2 + 50p_1 + 159p_2 - 650 + \lambda_1(-10p_1 - 9p_2 + 300) + \lambda_2(2p_1 - 45p_2 + 2210)$$

**Gradient condition:**

$$\nabla(L) = 0$$

$$\frac{dL}{dq_1} = -4p_1 + 50 - 10\lambda_1 + 2\lambda_2 = 0$$

$$\frac{dL}{dq_2} = -6p_2 + 159 - 9\lambda_1 - 45\lambda_2 = 0$$

**Feasibility Condition:**

$$-10p_1 - 9p_2 + 300 \leq 0$$

$$2p_1 - 45p_2 + 2210 \leq 0$$

**Orthogonality Condition:**

$$\lambda_1 \cdot (-10p_1 - 9p_2 + 300) = 0$$

$$\lambda_2 \cdot (2p_1 - 45p_2 + 2210) = 0$$

**Non-negativity condition:**

$$p_1, p_2 \geq 0$$

$$\lambda_1, \lambda_2 \leq 0$$

```
[1]: import numpy as np
from scipy.optimize import LinearConstraint, minimize

def objective_func(p):
    return 2*p[0]**2 + 3*p[1]**2 - 50*p[0] - 159*p[1]+650

rhs_coefs = np.array([[10, 9], [2, -45]])

constraints = LinearConstraint(rhs_coefs, lb=[300, 2210], ub=[np.inf, np.inf])

p0 = [5, 40]

res = minimize(objective_func, p0, constraints=constraints, bounds=[(0, np.inf), (0,
->np.inf)],
               options={"maxiter":1000, "disp": True})

print(res.x)

Optimization terminated successfully      (Exit mode 0)
      Current function value: 2387450.0025050757
      Iterations: 3
      Function evaluations: 11
      Gradient evaluations: 3
[1.10500000e+03 2.08014536e-08]
```

### 4.3.2 Car Production Planning

The Operations Department of DINOCO automobile manufacturing and assembly firm must establish the daily production quantities of two car models (SUV and Compact) which maximise profits according to the production capacity constraints. The firm also wishes to respect the environment, and in the future to benefit its sales, given its commitment to increase the FFM (Fleet Fuel Mileage) indicator; that is, the mileage efficiency per litre of fuel of produced fleet.

To simplify this study case, a linear relation between the sale price and the quantity to be produced is assumed, given by the following function:  $q_i = a_i - b_i \cdot p_i$ , where  $a_i$  and  $b_i$  are constants provided in the table below, and  $p_i$  is the mean unit price per family of cars in K€. Parameter  $C_i$  is also provided, which represents the cost per unit for the cars of family  $i$ .

The only productive resource that restricts the system is the assembly line. It is estimated that the firm spends 1 h every day on assembling SUV and Compact cars. The time required ( $R_i$ ) to produce one unit of each family of cars and the remaining former data are shown in the table.

i	Car model	a	b	C (K€)	R(seconds)	Kmpl
1	SUV	440	20	17	54	12
2	Compact	248	12	13	56	20

The FFM indicator, which must at least be 15 Kmpl, is calculated by using the harmonic mean:

$$FFM = \frac{q_1 + q_2}{\frac{q_1}{kmp1_1} + \frac{q_2}{kmp1_2}}$$

a) Formulate a non-linear programming model to help the Operations Department establish an optimum production plan which meets the existing constraints

The decision variables are the quantities to produce ( $q_i$ )

The objective of the production plan is to maximise profits:

$$\begin{aligned} \max z &= \sum_i q_i \cdot (p_i - C_i) \\ &= \sum_i (q_i) \cdot \left( \frac{a_i - q_i}{b_i} - C_i \right) \\ &= q_1 \cdot \left( \frac{440 - q_1}{20} - 17 \right) + q_2 \cdot \left( \frac{248 - q_2}{12} - 13 \right) \\ z &= -0.05q_1^2 - 0.083q_2^2 + 5q_1 + 7.6q_2 \end{aligned}$$

We have two constraints,

The first constraint takes into account the bottleneck in the assembly line (seconds):

$$\begin{aligned} \sum_i R_i \cdot q_i &\leq 3600 \\ 54 \cdot q_1 + 56 \cdot q_2 &\leq 3600 \\ 54 \cdot q_1 + 56 \cdot q_2 - 3600 &\leq 0 \end{aligned}$$

The second constraint is on the FFM (kmpl):

$$\begin{aligned} FFM &\geq 15 \rightarrow \\ q_1 + q_2 &\geq 15 \cdot \left( \frac{q_1}{kmp1_1} + \frac{q_2}{kmp1_2} \right) \rightarrow \\ q_1 \cdot \left( 1 - \frac{15}{kmp1_1} \right) + q_2 \cdot \left( 1 - \frac{15}{kmp1_2} \right) &\geq 0 \rightarrow \\ -q_1 \cdot \frac{3}{12} + q_2 \cdot \frac{5}{20} &\geq 0 \rightarrow \\ q_1 \cdot \frac{3}{12} - q_2 \cdot \frac{5}{20} &\leq 0 \rightarrow \\ 0.25 \cdot q_1 - 0.25 \cdot q_2 &\leq 0 \end{aligned}$$

We also have the non-negativity constraint:

$$q_i \geq 0 \forall i$$

**b) Find the Kuhn-Tucker conditions**

First, we obtain the Lagrangian:

$$L = -0.05q_1^2 - 0.083q_2^2 + 5q_1 + 7.7q_2 + \lambda_1 \cdot (54 \cdot q_1 + 56 \cdot q_2 - 3600) + \lambda_2 \cdot (0.25 \cdot q_1 - 0.25 \cdot q_2)$$

**Gradient condition:**

$$\nabla(L) = 0$$

$$\frac{dL}{dq_1} = -0.1q_1 + 5 + \lambda_1 \cdot 54 + \lambda_2 \cdot 0.25 = 0$$

$$\frac{dL}{dq_2} = -0.166q_2 + 7.7 + \lambda_1 \cdot 56 - \lambda_2 \cdot 0.25 = 0$$

**Feasibility Condition:**

$$54 \cdot q_1 + 56 \cdot q_2 - 3600 \leq 0$$

$$0.25 \cdot q_1 - 0.25 \cdot q_2 \leq 0$$

**Orthogonality Condition:**

$$\lambda_1 \cdot (54 \cdot q_1 + 56 \cdot q_2 - 3600) = 0$$

$$\lambda_2 \cdot (0.25 \cdot q_1 - 0.25 \cdot q_2) = 0$$

**Non-negativity condition:**

$$\lambda_1, \lambda_2 \leq 0$$

**c) A solver provides the following solution:**  $q_1 = 31.14$ ,  $q_2 = 34.26$ ,  $\lambda_1 = -0.034$ ,  $\lambda_2 = 0$ , where  $\lambda_1$  is the Lagrangian multiplier corresponding to the capacity constraint and  $\lambda_2$  is the Lagrangian multiplier of the FFM constraint. Interpret the results.

The Hessian is:

$$H = \begin{bmatrix} -0.1 & 0 \\ 0 & -0.166 \end{bmatrix}$$

The Hessian determinants are:

$$H_1 = -0.1$$

$$H_2 = 0.0166$$

Which is not definite negative, so the test is inconclusive. We can try with the equivalent minimisation problem  $\min z^* = -z$ . We know that for this problem the Hessian is:

$$H = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.166 \end{bmatrix}$$

And in this case, the Hessian is definitive positive, so we know that the (critical) Kuhn-Tucker point of the equivalent maximisation problem is global maximum.

The optimal production plan is to make 31.14 units of SUV cars and 34.26 units of compact cars. The assembly line has a shadow price of 0.034€, meaning that we could increase the profit by 0.034€ per every extra second of assembly line effective time.

### 4.3.3 Nova Case

#### Problem Definition

You work for Nova case, a start-up company that manufactures protective cases for mobile phones made of a novel material. The marketing department of your company has estimated a formula for the optimal price elasticity of demand for the two different models you are currently selling online. Recall that the price elasticity of demand represents the relationship between the quantity and the price of a good or service. They have estimated that, for both products, the price elasticity follows the following expression:

$$LQ = K + E \cdot LP$$

Where LQ is the logarithm of the quantity, LP is the logarithm of the price, K is the demand constant and E the demand elasticity. You are in charge of the production department and need to calculate the optimal production mix one planning period (1 day), taking into account this relationship between price and quantity. You have local production, and the most restrictive resource is the moulding machine which has different cycle time (in seconds) for each case. You also have a limited supply of the novel material and each model uses a different quantity of this material. In summary, this is the data of your first product:

- $K_1 = 4$
- $E_1 = 2$
- cycle time = 30 seconds
- key material quantity = 45 grams

And for your second product:

- $K_2 = 8$
- $E_2 = 2$
- cycle time = 25 seconds
- key material quantity = 40 grams

The total operational time of the moulding machine in the planning period is 6.5 hours and your supplier can provide you up to 40 kilograms per day of the novel material.

**a.** Write down a Non-Linear Programming problem to calculate the optimal production mix. (1 point) This is a non-linear production mix problem where the objective is to maximise profits. The objective function is of the type:

$$\max z = \sum_i Q_i \cdot P_i$$

We know that relationship between price and quantity is determined by the price elasticity expression:

$$LQ = K + E \cdot LP$$

Now, we can transform this into a linear expression as:

$$e^{\ln(Q)} = e^{k+E \cdot \ln(P)} = e^k * e^{E \cdot \ln(P)}$$

$$Q = e^k \cdot P^E$$

or

$$P = e^{-k/E} \cdot Q^{1/E}$$

We can now substitute this expression in the objective function.

$$\max z = \sum_i Q_i \cdot e^{-k_i/E_i} \cdot Q_i^{1/E_i}$$

$$\max z = \sum_i e^{-k_i/E_i} \cdot Q_i^{E_i+1/E_i}$$

Now, by substituting the values of k and E for both products, we get:

$$\max z = e^{-2} \cdot Q_1^{3/2} + e^{-4} \cdot Q_2^{3/2}$$

We have two constraints that depend on the decision variables, the operating time of the moulding machine and the availability of materials:

$$30 \cdot Q_1 + 25 \cdot Q_2 \leq 6.5 \cdot 60 \cdot 60$$

$$45 \cdot Q_1 + 40 \cdot Q_2 \leq 40000$$

**b.** Write down the Kuhn Tucker conditions. (2 points)

Let us start by writing the problem in the canonical form:

$$\max z = e^{-2} \cdot Q_1^{3/2} + e^{-4} \cdot Q_2^{3/2}$$

$$30 \cdot Q_1 + 25 \cdot Q_2 - 23400 \leq 0$$

$$45 \cdot Q_1 + 40 \cdot Q_2 - 40000 \leq 0$$

Now, the Lagrangian:

$$L = e^{-2} \cdot Q_1^{3/2} + e^{-4} \cdot Q_2^{3/2} + \lambda_1 \cdot (30 \cdot Q_1 + 25 \cdot Q_2 - 23400) + \lambda_2 \cdot (45 \cdot Q_1 + 40 \cdot Q_2 - 40000)$$

We are ready to proceed with the Kuhn Tucker conditions.

- Gradient condition

$$\frac{dL}{dQ_1} = 3/2 \cdot e^{-2} \cdot \sqrt{Q_1} + 30 \cdot \lambda_1 + 45 \cdot \lambda_2$$

$$\frac{dL}{dQ_2} = 3/2 \cdot e^{-4} \cdot \sqrt{Q_2} + 25 \cdot \lambda_1 + 40 \cdot \lambda_2$$

- Orthogonality condition

$$\lambda_1 \cdot (30 \cdot Q_1 + 25 \cdot Q_2 - 23400) = 0$$

$$\lambda_2 \cdot (45 \cdot Q_1 + 40 \cdot Q_2 - 40000) = 0$$

- Feasibility condition

$$30 \cdot Q_1 + 25 \cdot Q_2 - 23400 \leq 0$$

$$45 \cdot Q_1 + 40 \cdot Q_2 - 40000 \leq 0$$

- Non negativity condition

$$Q_1, Q_2 \geq 0$$

$$\lambda_1, \lambda_2 \leq 0$$

**c.** Assume that you need to take into account in your model the initial stock level of each product at the beginning of the planning period. How does this affect your model? Motivate your response (1 point)

If we need to take into account the initial stock level, we need to define new decision variables representing the amounts to stock for next periods, and a new index to represent the periods. The objective function would take into account the total available quantity, that is, the sum of the produce quantity plus the quantity manufactured in previous periods that we can use at one period. Similarly, in the constraint we could use the available resources to produce for the current period or for future periods.

**d.** What would it mean if, in a Kuhn Tucker point, the Lagrangian multiplier of the material quantity constraint was equal to zero? (1 point)

If the Lagrangian multiplier of the material quantity constraint is equal to zero, then by the orthogonality condition the right hand side of the constraint might be less than the daily quantity available, meaning that there might be some unused quantity.

### 4.3.4 Maximising investments of a startup

#### Problem definition

Your startup company has just raised 100M€ in a round of funding to reinvest in the company during the next period. Using regression and historical data, your team has found the following formulas for the Return of Investments (ROI) in different departments:

#### Research and Development:

$$10 \cdot (\sqrt[3]{2X} - 2)$$

#### Advertisement:

$$10 \cdot (\ln(X) - 2)$$

#### Technical Equipment:

$$10 \cdot (e^{X/50} - 2)$$

#### Human Resources:

$$10 \cdot \frac{X}{50}$$

According to the collective agreement, the minimum investment in human resources is 20M€. Additionally, analysing the formulas obtained, it is clear that although significant investments in technical equipment yield a high ROI, for small investments the ROI of R&D or advertisement is actually larger. However, due to the pressure imposed by competitors, the company has decided that investments in technical equipment must be at least high enough to ensure that this concept does not have a negative impact in the ROI of the next period investments.

**a.** Write a Non-Linear Programming Model to maximise the ROI (1.5 points)

Decision variables  $x_n, n = [1, 2, 3, 4]$ : (non-negative) Investment in M€ in 1: R&D, 2: Advertisement, 3: Technical Equipment, 4: Human Resources

Maximise ROI.

$$\max z = 10 \cdot (\sqrt[3]{2x_1} - 2) + 10 \cdot (\ln(x_2) - 2) + 10 \cdot (e^{x_3/50} - 2) + 10 \cdot \frac{x_4}{50}$$

s.t.

$$x_1 + x_2 + x_3 + x_4 \leq 100$$

$$x_4 \geq 20$$

$$10 \cdot (e^{x_3/50} - 2) \geq 0$$

$$x_n \geq 0 \quad \forall n$$

**b.** Write down the Kuhn-Tucker conditions (1.5 points)

First, we write down the problem in the **canonical form**:

$$\max z = 10 \cdot (\sqrt[3]{2x_1} - 2) + 10 \cdot (\ln(x_2) - 2) + 10 \cdot (e^{x_3/50} - 2) + 10 \cdot \frac{x_4}{50}$$

s.t.

$$x_1 + x_2 + x_3 + x_4 - 100 \leq 0$$

$$20 - x_4 \leq 0$$

$$-10 \cdot (e^{x_3/50} - 2) \leq 0$$

$$x_n \geq 0 \quad \forall n$$

From here, we can write down the **Lagrangian**:



$$L = 10 \cdot (\sqrt[3]{2x_1 - 2}) + 10 \cdot (\ln(x_2) - 2) + 10 \cdot (e^{x_3/50} - 2) + 10 \cdot \frac{x_4}{50}$$

$$\lambda_1 \cdot (x_1 + x_2 + x_3 + x_4 - 100)$$

$$\lambda_2 \cdot (20 - x_4)$$

$$\lambda_3 \cdot (-10 \cdot (e^{x_3/50} - 2))$$

And, once we have computed the lagrangian, we can obtain the Kuhn-Tucker conditions:

**Gradient condition**

$$\nabla L = 0$$

$$\frac{dL}{dx_1} = 20/3 \cdot (2 \cdot x_1 - 2)^{-2/3} + \lambda_1 = 0$$

$$\frac{dL}{dx_2} = 10 \cdot (1/x_2) + \lambda_1 = 0$$

$$\frac{dL}{dx_3} = 10/50 \cdot e^{x_3/50} + \lambda_1 - \lambda_3 \cdot 10/50 \cdot e^{x_3/50} = 0$$

$$\frac{dL}{dx_4} = 10/50 + \lambda_1 - \lambda_2 = 0$$

**Orthogonality condition**

$$\lambda_1 \cdot (x_1 + x_2 + x_3 + x_4 - 100) = 0$$

$$\lambda_2 \cdot (20 - x_4) = 0$$

$$\lambda_3 \cdot (-10 \cdot (e^{x_3/50} - 2)) = 0$$

**Feasibility condition**

$$x_1 + x_2 + x_3 + x_4 - 100 \leq 0$$

$$20 - x_4 \leq 0$$

$$-10 \cdot (e^{x_3/50} - 2) \leq 0$$

**non-negative condition:**

$$\lambda_n \leq 0 \forall n$$

$$x_n \geq 0 \quad \forall n$$

**c.** Solving the problem yields a value of -0.29 for the lagrangian multiplier of the minimum investment in human resources constraint. Interpret this result. How should the company use this value in the negotiation of the next collective agreement? (1 point)

The lagrangian multiplier represents the change in the objective function per unitary change in the right-hand side of the corresponding constraint. Note that, in the canonical form, if we increase the right hand side 1 unit (from 0), then we need to increase the left hand side also one unit for the equation to remain the same, for instance making the investments in human resources raise to 21M€. This yields a decrease of the ROI of -0.29M€, so the impact of increasing the investments in human resources is negative in the return for the company. In the next collective agreement negotiation, the company could use this to argue against proposals or make a proposal to reduce the investment.

### 4.3.5 Planning refreshment production

A firm that packs refreshments employs the same syrup to produce its 1.5 l COLI and PEPSA products on its S1 production line. Once processed, each hectolitre of syrup produces 40 units of the 1.5 l COLI product or 20 units of the 1.5 l PEPSA product. If  $x_1$  units of the 1.5 l COLI product and  $x_2$  units of the 1.5 l PEPSA product are produced, the firm estimates that the daily income obtained in dollars would be given by the following function:

$$f(x) = 49000 \cdot x_1 - x_1^2 + 30 \cdot x_2 - 2 \cdot x_2^2$$

It costs 150 dollars to buy and process each hectolitre of syrup. The S1 packaging line has a net capacity of producing 56,800 1.5 l product units every day. The firm is committed to produce at least half the amount of PEPSA than COLI. Although priority orders tend to amend its production planning, the firm wishes to have a basic product planning that optimises its daily profits.

a) Formulate a non-linear programming model that helps the firm create its basic daily product planning for its S1 packaging line. Our decision variables are:

- $x_1$ : Units of COLI
- $x_2$ : Units of PEPSA

The objective is to maximise the profits, so we need to calculate the difference between the revenues and the costs:

$$\max z = f(x) - c(x)$$

Let us first consider the costs as a function of the number of hectolitres of syrup used ( $h_l$ ):

$$c = 150 \cdot h_l$$

$$h_l = x_1/40 + x_2/20$$

If we define the cost as a function of  $x = [x_1, x_2]$ , then:

$$c(x) = 150 \cdot (x_1/40 + x_2/20)$$

Now,  $z$  becomes:

$$z = 49000 \cdot x_1 - x_1^2 + 30 \cdot x_2 - 2 \cdot x_2^2 - 3.75 \cdot x_1 - 7.5 \cdot x_2$$

As for the constraints, the capacity cannot exceed the total capacity of the line:

$$x_1 + x_2 \leq 56800$$

And the

$$z = 48996.25x_1 - x_1^2 + 22.5 \cdot x_2 - 2x_2^2$$

*s.t.*

$$x_1 + x_2 \leq 56800$$

$$x_2 \geq x_1/2$$

Or in canonical form:

$$z = 48996.25x_1 - x_1^2 - 22.5 \cdot x_2 - 2x_2^2$$

*s.t.*

$$x_1 + x_2 - 56800 \leq 0$$

$$x_1 - 2x_2 \leq 0$$

$$x_1, x_2 \geq 0$$

b) Write down the Kuhn Tucker conditions that would help the company determine the basic daily production plan.

The Lagrangian is:

$$L = 48996.25x_1 - x_1^2 - 22.5 \cdot x_2 - 2x_2^2 + \lambda_1 (x_1 + x_2 - 56800) + \lambda_2 (x_1/2 - x_2)$$

**Gradient condition:**

$$\nabla(L) = 0$$

$$\frac{dL}{dx_1} = 48996.25 - 2x_1 + \lambda_1 + \lambda_2 = 0$$

$$\frac{dL}{dx_2} = 22.5 - 4x_2 + \lambda_1 - \lambda_2 = 0$$

**Feasibility Condition:**

$$x_1 + x_2 - 56800 \leq 0$$

$$x_1 - 2x_2 \leq 0$$

**Orthogonality Condition:**

$$\lambda_1 \cdot (x_1 + x_2 - 56800) = 0$$

$$\lambda_2 \cdot (x_1 - 2x_2) = 0$$

**Non-negativity condition:**

$$x_1, x_2, \geq 0$$

$$\lambda_1, \lambda_2 \leq 0$$

c) The following critical point is found  $x_1 = 13838.33; x_2 = 6919.16; \lambda_1 = 0; \lambda_2 = -15323.34; z = 287249204.2$ , where  $\lambda_1$  is the Lagrangian multiplier corresponding to the S1 packaging line capacity constraint. Can you conclude whether it is a local/global maximum? With this solution, What is the maximum quantity that the firm would be willing to pay for 1 h of overtime production? How much would the daily profit increase per additional COLI unit produced?

The Hessian matrix is:

$$H = \begin{bmatrix} -2 & 0 \\ 0 & -4 \end{bmatrix}$$

The Hessian determinants are:

$$H_1 = -2$$

$$H_2 = 8$$

It is not possible to determine whether this Kuhn-Tucker point is a local maximum or minimum. However, we can do the test in the equivalent minimisation problem  $min z^* = -z$ . In this case, the Hessian matrix is:

$$H = \begin{bmatrix} 2 & 0 \\ 0 & 4 \end{bmatrix}$$

$$H_1 = 2$$

$$H_2 = 8$$

Which is definitive positive and therefore this Kuhn Tucker point is a global minimum in the equivalent minimisation problem and by extension a global maximum in the original maximization problem.

In this Kuhn-Tucker point, the  $\lambda_1$  is the shadow price of the constraint representing the net capacity and is equal to zero, meaning that the capacity is not binding the profit, and it is not profitable for the company to increase it with extra hours. On the other hand, the shadow price of the second constraint is  $\lambda_2$  is -15323.34, meaning that a unitary change of the function  $g_2$  increases the objective function in 15323.34. Since  $x_1$  (COLI) is directly proportional to  $g_2(x)$ , a positive change on the right hand side would increase the profit in 15323.34. Since changes in  $x_2$  are multiplied by a factor of -2 in the second constraint, a unitary change of  $x_2$  (PEPSA) would decrease the profit in 7661.67.

### 4.3.6 Production Mix of Pharma

#### Problem definition

Cheston Pharmaceuticals manufactures two different products. The company has conducted different studies to estimate the demand curve of their products, to learn the relationship between price and demand. The company has build a linear model that has proven to be accurate enough in the price range of interest for the company. Let us note the linear model as:

$$d_j = -\theta_j * p_j + \hat{\theta}_j \quad p_j = [m_j, M_j], \forall j = [1, 2]$$

Where  $d_j$  is the total units of product  $j$  manufactured,  $p_j$  is the price of product  $j$  and  $\theta_j$  and  $\hat{\theta}_j$  are the parameters of the linear models for the curve of demand of product  $j$ . The price of product  $j$  must be in the range of interest, that is, greater or equal than the minimum price  $m_j$  and lower or equal than the maximum  $M_j$ .

Both products incorporate a critical compound for which the company has limited resources to produce. The total production capacity of the company for the compound in the planning period is  $C$  (milligrams). A unit of product  $j$  contains  $r_j$  milligrams of the critical compound.

Let us assume that the production costs of product  $j$  can be neglected.

**a.** Write a Non-Linear Programming problem to calculate the optimal production mix (number of units to produce of each product) that maximises the revenues in the planning period.

The optimization problem can be defined as:

$$\max z = \sum_j d_j * p_j$$

s.t.

$$p_j \geq m_j \quad \forall j$$

$$p_j \leq M_j \quad \forall j$$

$$\sum_j d_j * r_j \leq C$$

Applying the demand curve:

$$p_j = \frac{d_j - \hat{\theta}_j}{-\theta_j} = -\frac{1}{\theta_j} * d_j + \frac{\hat{\theta}_j}{\theta_j}$$

yields:

$$\max z = \sum_j \left( -\frac{1}{\theta_j} * d_j^2 + \frac{\hat{\theta}_j}{\theta_j} * d_j \right)$$

s.t.

$$\frac{1}{\theta_j} * d_j - \frac{\hat{\theta}_j}{\theta_j} + m_j \leq 0 \quad \forall j$$

$$-\frac{1}{\theta_j} * d_j + \frac{\hat{\theta}_j}{\theta_j} - M_j \leq 0 \quad \forall j$$

$$\sum_j d_j * r_j - C \leq 0$$

**b** Write down the Lagrangian

The Lagrangian is:

$$\begin{aligned} L(d_j, \lambda_i) = & \sum_j \left( -\frac{1}{\theta_j} * d_j^2 + \frac{\hat{\theta}_j}{\theta_j} * d_j \right) + \lambda_1 * \left( \frac{1}{\theta_1} * d_1 - \frac{\hat{\theta}_1}{\theta_1} + m_1 \right) + \lambda_2 * \left( \frac{1}{\theta_2} * d_2 - \frac{\hat{\theta}_2}{\theta_2} + m_2 \right) + \dots \\ & \dots + \lambda_3 * \left( -\frac{1}{\theta_1} * d_1 + \frac{\hat{\theta}_1}{\theta_1} - M_1 \right) + \lambda_4 * \left( -\frac{1}{\theta_2} * d_2 + \frac{\hat{\theta}_2}{\theta_2} - M_2 \right) + \lambda_5 (\sum_j d_j * r_j - C) \end{aligned}$$

c Write down the Kuhn-Tucker conditions:

**Gradient condition**

$$\nabla L(d_j, \lambda_i) = 0$$

$$-\frac{2}{\theta_1} * d_1 + \frac{\hat{\theta}_1}{\theta_1} + \frac{\lambda_1}{\theta_1} - \frac{\lambda_3}{\theta_1} + \lambda_5 * r_1 = 0$$

$$-\frac{2}{\theta_2} * d_2 + \frac{\hat{\theta}_2}{\theta_2} + \frac{\lambda_2}{\theta_2} - \frac{\lambda_4}{\theta_2} + \lambda_5 * r_2 = 0$$

**Orthogonality condition**

$$\lambda_1 * \left( \frac{1}{\theta_1} * d_1 - \frac{\hat{\theta}_1}{\theta_1} + m_1 \right) = 0$$

$$\lambda_2 * \left( \frac{1}{\theta_2} * d_2 - \frac{\hat{\theta}_2}{\theta_2} + m_2 \right) = 0$$

$$\lambda_3 * \left( -\frac{1}{\theta_1} * d_1 + \frac{\hat{\theta}_1}{\theta_1} - M_1 \right) = 0$$

$$\lambda_4 * \left( -\frac{1}{\theta_2} * d_2 + \frac{\hat{\theta}_2}{\theta_2} - M_2 \right) = 0$$

$$\lambda_5 (\sum_j d_j * r_j - C) = 0$$

**Feasibility condition**

$$\frac{1}{\theta_j} * d_j - \frac{\hat{\theta}_j}{\theta_j} + m_j \leq 0 \forall j$$

$$-\frac{1}{\theta_j} * d_j + \frac{\hat{\theta}_j}{\theta_j} - M_j \leq 0 \forall j$$

$$\sum_j d_j * r_j - C \leq 0$$

**Non-negativity condition**

$$p_j \geq 0 \quad \forall j = [1, 2]$$

$$\lambda_i \leq 0 \forall i = [1, 5]$$

d. Write down the Hessian matrix and the Hessian determinants. In case that you find a critical point, can you determine if it is a local or a global maximum? Motivate your response.

$$H = \begin{bmatrix} -\frac{2}{\theta_1} & 0 \\ 0 & -\frac{2}{\theta_2} \end{bmatrix}$$

$$h_1 = -\frac{1}{\theta_1}$$

$$h_2 = \frac{1}{\theta_1 * \theta_2}$$

It is not possible to conclude that this is a critical point from the Hessian determinants, although it is possible to test the equivalent minimization problem  $z^* = -z$  for which the Hessian matrix is:

$$H = \begin{bmatrix} \frac{1}{\theta_1} & 0 \\ 0 & \frac{1}{\theta_2} \end{bmatrix}$$

And the Hessian determinants are:

$$h_1 = \frac{1}{\theta_1}$$

$$h_2 = \frac{1}{\theta_1 * \theta_2}$$

Meaning that the Hessian is definitive positive and therefore the critical point that has been found is a global maximum.

### 4.3.7 Optimising the design of headphones

#### Problem definition

You work on the R&D department of Banshee Ltd., a company that manufactures headphones. Through a customer survey you have determined that the level of satisfaction of your headphones is given by the following expression:

$$z = 5 + \ln(x_1) + x_2^2 + x_1 * x_2$$

where  $x_1$  is the thickness of the diaphragm in microns and  $x_2$  is the radius of the diaphragm in millimeters. Due to design constraints, the volume of the driver, which can be estimated as  $0.2 * \pi * x_2 * x_1$  cannot be larger than 0.172 cubic millimeters. The properties of the material you use makes it unfeasible to manufacture a diaphragm with a thickness lower than 0.3 microns or higher than 0.9 microns.

**a** Write a Non-Linear Programming (NLP) problem to find the optimal design specifications (thickness and radius of diaphragm) for Banshee Ltd.

The objective is to maximise the level of satisfaction given by the following expressions:

$$\max z = 5 + \ln(x_1) + x_2^2 + x_1 * x_2$$

s.t.

$$2 * \pi * x_2 * x_1 \leq 1.72$$

$$x_1 \geq 0.3$$

$$x_1 \leq 0.9$$

**b** Write down the Lagrangian and the Kuhn Tucker conditions

The Lagrangian is:

$$L(x_1, x_2, \lambda_1, \lambda_2, \lambda_3) = 5 + \ln(x_1) + x_2^2 + x_1 * x_2 + \lambda_1 * (2 * \pi * x_2 * x_1 - 1.72) + \lambda_2 * (0.3 - x_1) + \lambda_3 * (x_1 - 0.9)$$

#### Gradient condition

$$\nabla(L(x_1, x_2, \lambda_1, \lambda_2, \lambda_3)) = 0$$

$$\frac{\delta L}{\delta x_1} = \frac{1}{x_1} + x_2 + 2 * \pi * \lambda_1 * x_2 - \lambda_2 + \lambda_3 = 0$$

$$\frac{\delta L}{\delta x_2} = 2 * x_2 + x_1 + 2 * \pi * \lambda_1 * x_1 = 0$$

#### Feasibility condition

$$2 * \pi * x_2 * x_1 \leq 1.72$$

$$x_1 \geq 0.3$$

$$x_1 \leq 0.9$$

#### Orthogonality condition

$$\lambda_1 * (2 * \pi * x_2 * x_1 - 1.72) = 0$$

$$\lambda_2 * (0.3 - x_1) = 0$$

$$\lambda_3 * (x_1 - 0.9) = 0$$

#### Non-Negativity condition

Since we expressed the problem in the canonical form, we know that the multipliers must be non-negative:  $x_1, x_2 \geq 0$

$$\lambda_1, \lambda_2, \lambda_3 \leq 0$$

**c** A solver gives you the following solution ( $x_1 = 0.9, x_2 = 0.304$ ). Determine if it is a local or global maximum

First, we need to calculate the Lagrangian multipliers to make sure that the solution is feasible.

We plug in the numbers provided by the solver in the second gradient condition and obtain:

$$\lambda_1 = \frac{-2 * x_2 - x_1}{2 * \pi * x_1} = -0.267 \leq 0$$

From the second orthogonality condition, we know that  $\lambda_2$  must be zero:

$$\lambda_2 * (0.3 - x_1) = \lambda_2 * (-0.6) = 0$$

$$\lambda_2 = 0$$

Now, we can plug in this value into the first gradient condition to compute  $\lambda_3$ :

$$\lambda_3 = -1/x_1 - x_2 - 2 * \pi * \lambda_1 * x_2 = -0.906$$

Since all the multipliers are non-positive, the solution is feasible. The Hessian is given by:

$$H = \begin{bmatrix} \frac{-1}{x_1^2} & 1 \\ 1 & 2 \end{bmatrix}$$

The determinants of the Hessian are:

$$h_1 = \frac{-1}{x_2^2} \leq 0 \quad \forall x_2 > 0$$

$$h_2 = \frac{-2}{x_2^2} - 1 \leq 0 \quad \forall x_2 > 0$$

Since  $x_2$  is greater than zero, the solution is a global maximum.

### 4.3.8 Mathematical Nirvana

#### Problem Definition

Nathaniel Richards is a young, brilliant scientist that has developed the following Non-Linear Programming (NLP) problem to find the optimal balance between studying and meditating to maximize the overall satisfaction and achieve a state of enlightenment (Mathematical Nirvana).

**Decision Variables:** Let:

$x_1$  Time spent studying (in hours)

$x_2$  = Time spent meditating (in hours)

$x = [x_1, x_2]$  The set of decision variables

$$x_1, x_2 \geq 0$$

**Objective Function:** Maximize the overall satisfaction obtained from studying and meditating:

$$\max z = f(x) = 2 * x_1 + 0.5 * \ln(1 + x_1) + 0.7 * x_2 + 0.3 * \sqrt{x_2}$$

**Constraints:** Subject to: Maximum amount of time available:

$$x_1 + x_2 \leq 10$$

Minimum amount of time studying to ensure academic performance:

$$x_1 \geq 2$$

Unfortunately Nathaniel mysteriously disappeared before he could completely analyse the problem, so you need to complete his work according to the following instructions:

1. Obtain the Kuhn-Tucker conditions To obtain the Kuhn-Tucker conditions, let us first get the Lagrangian function:

$$L(x, \lambda) = f(x) + \lambda_1(g_1(x)) - \lambda_2(g_2(x))$$

$$L(x, \lambda) = 2 * x_1 + 0.5 * \ln(1 + x_1) + 0.7 * x_2 + 0.3 * \sqrt{x_2} + \lambda_1(x_1 + x_2 - 10) + \lambda_2(-x_1 + 2)$$

Note that to obtain the expression above we have first express the problem in a canonical form where all the constraints are inequalities of type  $\leq$ .

Now, we can obtain the Kuhn-Tucker conditions by taking the partial derivatives of the Lagrangian function with respect to the decision variables and the Lagrangian multipliers:

- **Gradient Conditions:**  $\nabla_x L(x, \lambda) = 0$

$$\frac{\partial L}{\partial x_1} = 2 + \frac{0.5}{1+x_1} + \lambda_1 - \lambda_2 = 0$$

$$\frac{\partial L}{\partial x_2} = 0.7 + \frac{0.3}{2\sqrt{x_2}} + \lambda_1 = 0$$

- **Orthogonality Conditions:**  $\lambda_i g_i(x) = 0$

$$\lambda_1(x_1 + x_2 - 10) = 0$$

$$\lambda_2(-x_1 + 2) = 0$$

- **Feasibility Conditions:**  $g_i(x) \leq 0$

$$x_1 + x_2 - 10 \leq 0$$

$$-x_1 + 2 \leq 0$$

- **Non-negativity Conditions:**

$$\lambda_1, \lambda_2 \geq 0 \quad x_1, x_2 \geq 0$$

2. Obtain the Hessian and determine if this solution ( $x_1 = 3.9, x_2 = 6.1$ ) is a global or local maximum To obtain the Hessian matrix, we need to take the second derivatives of the objective function with respect to the decision variables:

$$\frac{\partial^2 f(x)}{\partial x_1^2} = -\frac{0.5}{(1+x_1)^2} \leq 0$$

$$\frac{\partial^2 f(x)}{\partial x_2^2} = -\frac{0.3}{4*x_2^{3/2}} \leq 0$$

$$\frac{\partial^2 f(x)}{\partial x_1 \partial x_2} = 0$$

$$\frac{\partial^2 f(x)}{\partial x_2 \partial x_1} = 0$$

Now, we can obtain the Hessian matrix:

$$H(x) = \begin{bmatrix} -\frac{0.5}{(1+x_1)^2} & 0 \\ 0 & -\frac{0.3}{4*x_2^{3/2}} \end{bmatrix}$$

To determine if the solution is a global or local maximum, we need to check the definiteness of the Hessian matrix for the values  $x_1 = 3.9, x_2 = 6.1$ :

$$H(x) = \begin{bmatrix} -\frac{0.5}{(1+3.9)^2} & 0 \\ 0 & -\frac{0.3}{4*6.1^{3/2}} \end{bmatrix} = \begin{bmatrix} -0.02 & 0 \\ 0 & -0.005 \end{bmatrix}$$

The first determinant is negative ( $h_1 = -0.02$ ), and the second determinant is positive ( $h_2 = -0.02 * -0.005 > 0$ ), so the Hessian matrix is indefinite. Therefore, the test is inconclusive. If we look at the Hessian matrix of the equivalent minimization problem ( $\min f^*(x) = -f(x)$ ), since now the Hessian becomes:

$$H(x) = \begin{bmatrix} 0.02 & 0 \\ 0 & 0.005 \end{bmatrix}$$

we can see that it is positive definite ( $h_1 = 0.02, h_2 = 0.02 * 0.005 > 0$ ), so the solution is a global minimum, and therefore the solution to the original maximization problem is a global maximum.



5. Use the Kuhn-Tucker conditions to calculate the Lagrangian multipliers for this solution, can you explain what they mean? Discuss if this can be an optimal solution to the problem

Now, we can use the Kuhn-Tucker conditions to calculate the Lagrangian multipliers for this solution. From the gradient conditions, we have:

$$\frac{\partial L}{\partial x_1} = 2 + \frac{0.5}{1+3.9} + \lambda_1 - \lambda_2 = 0$$

$$\frac{\partial L}{\partial x_2} = 0.7 + \frac{0.3}{2\sqrt{6.1}} + \lambda_1 = 0$$

From the second gradient condition, we can obtain the value of  $\lambda_1$ :

$$\lambda_1 = -0.7 - \frac{0.3}{2\sqrt{6.1}} = -0.8$$

From the first gradient condition, we can obtain the value of  $\lambda_2$ :

$$\lambda_2 = 2 + \frac{0.5}{1+3.9} + \lambda_1 = 1.35$$

The Lagrangian multipliers obtained do not satisfy the orthogonality conditions and non-negative conditions, so this solution cannot be optimal.



## DECISION THEORY AND GAME THEORY

This chapter provides an introduction to decision theory, where we will focus on solving decision problems where the decision maker must select among various alternatives, which provide different performance. The chapter also covers game theory, a field dedicated to modelling and solving decision problems involving several rational decision makers. Here, we will focus on solving decision problems where the decision maker must select among various alternatives, focusing on competition and collaboration.

### 5.1 Tutorials

This section contains some basic tutorials:

#### 5.1.1 Decision Theory

##### Introduction

Decision theory spans a combination of problem-solving techniques to find the best decision in complex decision problems. In this unit, we will use the following terminology:

- **Alternatives:** Decision variables which are controllable and depend on the decision maker's decision.
- **Uncertainty and states of nature:** External variables, are uncontrollable and need to be estimated or assumed.
- **Performances:** Profit or Cost (utility) of the result of a decision

Hence, the objective is to find the alternative with the highest performance for the decision maker, possibly in situations where the performance may depend on external variables which cannot be controlled.

So far, this set up is not that different from the definitions seen in previous units, however, we will be focusing on the following aspects in this unit:

- **Impacts over time** All the important consequences of a problem do not occur at the same instant in time
- **Uncertainty** At the time the decision-maker must select an alternative, the consequences are not known with certainty.
- **Possibility of acquiring information** Often we can acquire additional information to support decision-making at a cost. For instance, to collect seismic information to decide whether to drill for oil. Decision theory provides methods to evaluate if it is worth acquiring additional information or not.
- **Dynamic aspects** The problem might not end immediately after an alternative is chosen but might require further analysis (e.g. further decisions)

### Pay off Matrix

The pay-off matrix is a tool to represent the performances of the different alternatives against the possible future outcomes of an uncontrolled event, or states of nature. The different rows of the matrix represent the alternatives of the decision maker, and the different columns of the matrix, the different possible states of nature. Finally, every cell in the matrix contains the performance of each alternative given the occurrence of each state of nature. That is, let us note the decision maker's alternatives as  $a_1, a_2, \dots, a_m$ , and the different states of nature as  $s_1, s_2, \dots, s_n$ . Let us note the performance of alternative  $a_i$  when  $s_j$  has occurred as  $u_{ij}$ , then the pay-off matrix is:

$$\begin{bmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ u_{21} & u_{22} & \dots & u_{2n} \\ \dots & \dots & \dots & \dots \\ u_{m1} & u_{m2} & \dots & u_{mn} \end{bmatrix}$$

Note that the pay-off matrix represents all possible outcomes of our decision under uncertainty, given that the states of nature are exhaustive (together they describe all the possible outcomes of the uncontrolled variable) and mutually exclusive (if one occurs, then the rest cannot occur). The pay-off matrix can also be represented in tabular form as:

Decision Alternatives	state 1	state 2	...	state n
$a_1$	$u_{11}$	$u_{12}$	...	$u_{1n}$
$a_2$	$u_{21}$	$u_{22}$	...	$u_{2n}$
...	...	...	...	...
$a_m$	$u_{m1}$	$u_{m2}$	...	$u_{mn}$

Let us, at this point, set one example to be used in the next sections of this presentation. In the example, the decision maker needs to select a financial product among a set of market alternatives: Gold, bonds, stock options, deposit, or hedge fund. Each product provides different benefits or losses depending on the behaviour of the market. The behaviour of the market is not controlled by the decision maker, therefore, it is characterised in this example as a set of states of natures:

- Accumulation Phase: The market is stable and characterised by a slow raise
- Mark-up Phase: The market has been stable for a while, investors feel secure and the market is characterised by a fast raise.
- Distribution Phase: Most investors start selling to collect benefits, the market stabilises and there is little or no change.
- Mark-down Phase: Some investors try to hold positions as there is a large fall of the market

The pay-off matrix shows the expected performance in euros of the investment of each product under each market cycle phase:

Decision Alternatives	Accumulation	Mark-up	Distribution	Mark-down
Gold	-100	100	200	0
Bonds	250	200	150	-150
Stock options	500	250	100	-600
Deposit	60	60	60	60
Hedge fund	200	150	150	-150

### Dominance

Dominance is a property of alternatives that provide a better performance than others for every state of nature. In the example above, it can be seen that the bond provides a higher benefit than the option hedge in any market scenario, therefore the bond option is dominant over the hedge fund option. Dominance allows us to simplify the decision-making process, eliminating the dominated alternatives in favour of the dominant alternatives. Taking into account dominance, the pay-off matrix above can be written as:

Decision Alternatives	Accumulation	Mark-up	Distribution	Mark-down
Gold	-100	100	200	0
Bonds	250	200	150	-150
Stock options	500	250	100	-600
Deposit	60	60	60	60

Any rational decision maker would select bonds before stock options under any market phase and therefore, the latter can be ignored.

### Decision Rules

Decision rules are criteria used by a rational decision maker to make systematic decisions, id est, to select the best alternative. This section describes some of the most important criteria in related literature:

#### MinMax

The MinMax criteria aims to minimise loss in the worst-case scenario. It is therefore a conservative, or pessimistic criteria. Also known as Maximin (minimise the maximum loss), this criteria first finds the minimum value of the performance of each decision variables in every scenario:

$$m_i = \min(u_{i1}, u_{i2}, \dots, u_{in}) \quad \forall i = [1, 2, \dots, m]$$

These values represent the worst case scenario in every decision alternative, then the minmax criteria selects the maximum of these values:

$$d = \operatorname{argmax}(m_1, m_2, \dots, m_m)$$

The function  $\operatorname{argmax}$  above returns the index of the maximum value rather than the maximum value. Hence, it will return the index of the alternative for which the maximum is value:

In the example above, the minmax criteria yields:

Decision Alternatives	Accumulation	Mark-up	Distribution	Mark-down	$m_i$
Gold	-100	100	200	0	-100
Bonds	250	200	150	-150	-150
Stock options	500	250	100	-600	-600
Deposit	60	60	60	60	60

$$d = \operatorname{argmax}(-100, -150, -600, 60) = 4$$

$$v = \operatorname{max}(-100, -150, -600, 60) = 60$$

The MinMax criteria models the decisions of a conservative rational decision maker and in this example, a conservative investor will invest in deposits.

**MaxMax**

The MaxMax criteria on the other hand aims to maximise profit in the best-case scenario. It is therefore an aggressive, and optimistic criteria. This criteria first finds the maximum value of the performance of each decision variables in every scenario:

$$o_i = \max(u_{i1}, u_{i2}, \dots, u_{in}) \quad \forall i = [1, 2, \dots, m]$$

These values represent the best case scenario in every decision alternative, then the MaxMax criteria selects the maximum of these values:

$$d = (m_1, m_2, \dots, m_n)$$

In the example above, the MaxMax criteria yields:

Decision Alternatives	Accumulation	Mark-up	Distribution	Mark-down	$o_i$
Gold	-100	100	200	0	200
Bonds	250	200	150	-150	250
Stock options	500	250	100	-600	500
Deposit	60	60	60	60	60

$$d = (200, 250, 500, 60) = 3$$

$$v = \max(200, 250, 500, 60) = 500$$

The MaxMax criteria models the decisions of a conservative rational decision maker and in this example, a conservative investor will invest in deposits.

**Bayesian**

The Bayesian criteria takes into account additional information about the likelihood of each state of nature. It is therefore the criteria applied by an informed decision maker which wants to factor in the decision the likelihood of each alternative. The probabilities of each state or nature as defined as  $p(s_j)$ . The Expected Monetary Value (EMV) of each alternative is defined as:

$$EMV_i = \sum ij * p(s_j)$$

The EMV can be interpreted as the weighted average value of the alternative given the probabilities of occurrence of each state of nature.

The Bayesian criteria selects the alternative with the maximum EMV:

$$d = \operatorname{argmax}(textEMV_1, textEMV_2, \dots, textEMV_m)$$

Let us use again the example above to apply this decision criteria.

States of nature	Accumulation	Mark-up	Distribution	Mark-down	EMV
Probabilities ( $p(s_j)$ )	0.2	0.3	0.3	0.2	EMV
Gold	-100	100	200	0	70
Bonds	250	200	150	-150	125
Stock options	500	250	100	-600	85
Deposit	60	60	60	60	60

$$d = \operatorname{argmax}(70, 125, 85, 60) = 2$$

$$v = \max(70, 125, 85, 60) = 125$$

The Bayesian criteria selects the decision alternative with index 2 (Bonds) and its EMV is 125.

## 5.1.2 Decision Trees

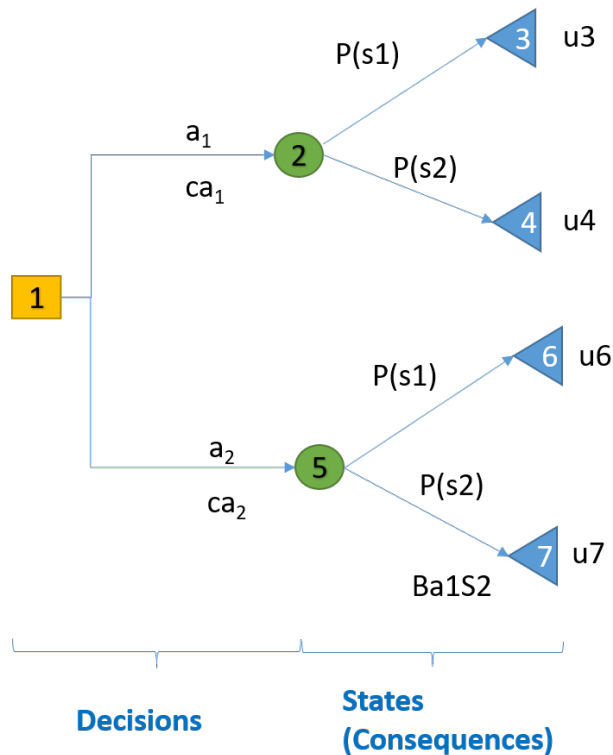
### Introduction

Decision Trees are chronological representations of complex decision processes that can be used to find an optimal decision based on a specific criteria, using a divide-and-conquer approach.

Decision trees have three main graphical components:

- **Decision Nodes:** Represent decisions under certainty. Each branch represents a decision alternative and is represented by a utility or cost as a measure of its performance.
- **Random nodes:** Represent possible outcomes under uncertainty. Each branch corresponds to a possible state and is represented by a probability  $P$
- **Leaves:** Possible outcomes of the decision process. They are represented by the performance, the utility or cost value corresponding to the connected decision nodes and random nodes.

The image below shows these components and their relationships as a baseline example.



The example shows just a decision node, labeled as 1, with two different alternatives, noted as  $a_1$  and  $a_2$ . Each alternative has a different utility value ( $ca_1$  for  $a_1$  and  $ca_2$  for  $a_2$ ). There is some uncontrolled variable that can impact our decision, and therefore, for each of the output branches of the decision node, there are two random nodes, labeled as 2 and 5, which represent the uncontrolled variable. From each node, there are two different branches that represent the two possible states of the decision variable. Each branch is labeled with the corresponding probability of the same state of nature,  $P(s_1)$  and  $P(s_2)$ . Finally, the leaf nodes, are labeled according to the utility values of each decision under each state of nature ( $u_3, u_4, u_6, u_7$ ). Note that the same information can be represented in a Pay-off matrix as:

Alternatives	State 1	State 2
Probabilities	$P(s_1)$	$P(s_2)$
$a_1$	$u_3 + ac_1$	$u_4 + ac_1$
$a_2$	$u_6 + ac_2$	$u_7 + ac_2$

However, the decision tree allows to represent more complex set-ups adding more nodes and uncontrolled variable, even dealing with different combinations of uncontrolled variables and decision nodes at each branch. To define a decision tree, it is necessary to apply the following steps

1. **Structure the problem:** define decision steps, states, objectives and their performance measures. In general, a decision problem can have multiple objectives. Each objective has a performance measure to represent preferences of the decision maker (e.g. An objective can be to minimize costs and the performance metric can be a monetary unit like euros). If there are multiple objectives, it is necessary to take into account the trade-offs among different objectives to obtain a single utility function to represent preferences. However, for the rest of the unit, we will deal with single objective decision problems. Utility functions help us quantify the value of each decision. Each alternative can have a positive impact or negative impact on the value.
2. **Describe consequences:** describe the probability of each possible state of nature. In situations of certainty, each decision will have a positive or negative impact in the value of the decision. In situations of uncertainty or risk, each possible state will have a different probability of occurrence. We need an estimation of these probabilities to quantify the value of each decision.
3. **Assign a value to each decision alternative:** Define the utility of each decision. We need to estimate the utility of each possible outcome of the decision problem.
4. **Decide which of the proposed alternatives is preferred:** To decide which of the proposed alternative is preferred, it is necessary to *solve* the decision tree, as described in the next section.

### Solving Decision Trees

Decision trees are solved from right to left, that is, from the leaves, back to the initial decision node. At every decision step, it is necessary to quantify the value of each alternative, taking into account the probability of states of nature. Moreover, the values of random nodes and decision nodes are calculated differently.

The value of a random node is calculated using a Bayesian approach, as the weighted expected value across the different states:

$$V_r = \sum_{i=1}^n P(s_i) * V_i$$

Where  $P(s_i)$  is the probability of state i and  $V_i$  is the value at branch or leaf i.

The value of a decision node on the other hand is calculated as:

$$V_n = \max\{V_i - ca_k, V_j - ca_p, \dots\}$$

That is, a rational decision would always select the alternative with the highest value across the different alternatives, taking into account their costs.

With this two simple rules, the value of a tree can be calculated from its leaves, no matter how complex the set-up is and regardless of the number of steps.

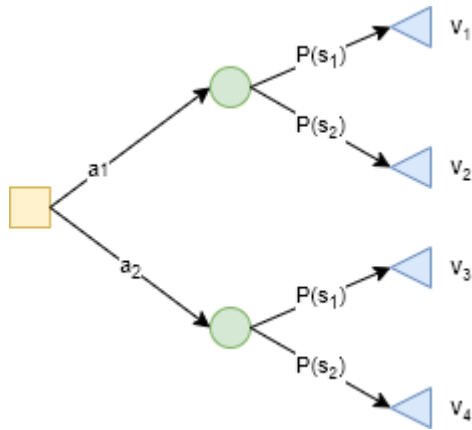


## Imperfect Information

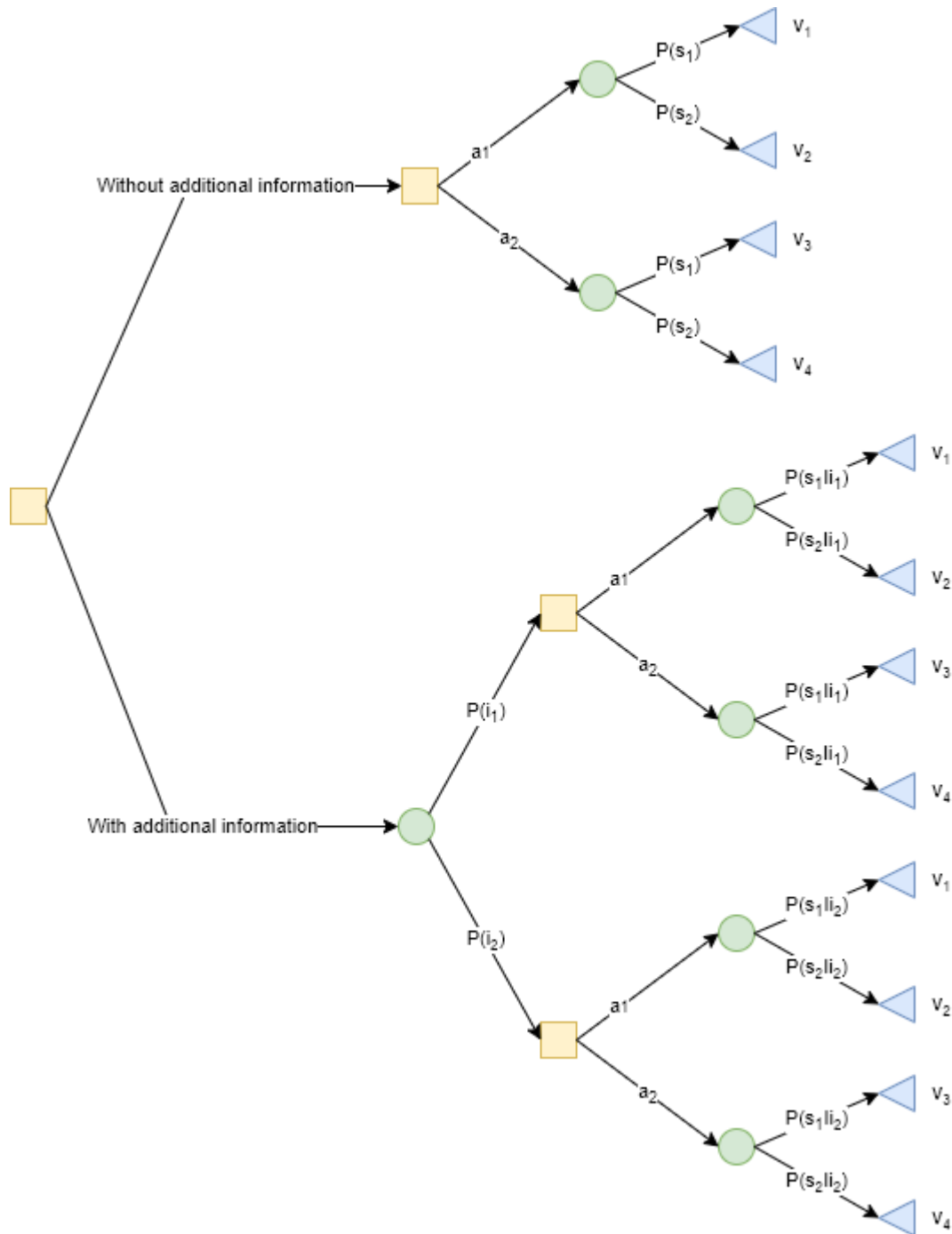
Sometimes we need to decide whether to incorporate into the decision **imperfect information**. Imperfect information is information about uncontrolled variables that we can incorporate into the decision (possibly at a cost) to improve the performance. For instance, coming back to the previous example, imagine that our decision problem is to decide a financial product. To make a better decision, we may purchase a forecast report of the market trends for the following year. The report forecasts a given market trend, but the state is not known with certainty. It is possible to evaluate the usefulness of incorporating this information into the problem. Applying the **Bayes Theorem**, the probabilities of the different states of random nodes now will be conditioned on the outcome of the imperfect information (in the example, of the forecast report).

In these cases, it is necessary to review the different probabilities. After imperfect information, the probabilities of occurrence of the different states become revised (posterior) probabilities. The revised probabilities are calculated by applying the Bayes Theorem.

Let us assume that there are  $n$  different states and that, consequently, the imperfect information may provide additional information about the occurrence of each of the states. The introduction of imperfect information can be modeled first as a decision node that models the decision of whether to incorporate imperfect information or not, and then as a random node in the branch of the imperfect information state  $i$ . For instance, let us consider a basic example with two alternatives and two possible states.



Let us now introduce the acquisition of imperfect information about the states. First, the decision maker needs to decide whether not to incorporate this information (make the decision without imperfect information) or to incorporate this information (make the decision with additional information, possibly at a cost). The diagram below introduces this first decision as an additional decision node. Then, since the result of the forecast is not controlled by the decision maker, its two possible results -  $i_1$  when it forecasts  $s_1$  and  $i_2$  when it forecasts  $s_2$  - are modeled as a random node with two states.



Note that now the probability of state  $s_j$  is conditioned to the event  $i_i$  in every branch. We can apply the Bayes Theorem to calculate the probability that event  $s_j$  occurs given that the imperfect information forecasts  $i_i$ :

$$(j | i) = \frac{(j) \cdot (i | j)}{(i)} = \frac{(j) \cdot (i | j)}{\sum_k (k) \cdot (i | s_k)}$$

In the example below, this yields:

$$P(i_1) = (1) \cdot (i_1 | s_1) + (2) \cdot (i_1 | s_2)$$

$$(1 | 1) = \frac{(1) \cdot (1 | 1)}{P(i_1)}$$

$$(2 | 1) = \frac{(2) \cdot (1 | 2)}{P(i_1)}$$

$$P(i_2) = (1) \cdot (i_2 | s_1) + (2) \cdot (2 | 2)$$

$$(1 | 2) = \frac{(1) \cdot (2 | 1)}{P(i_2)}$$

$$(2 | 2) = \frac{(2) \cdot (2 | 2)}{P(i_2)}$$

$P(s_1)$  and  $P(s_2)$  are the **a priori** probabilities, or the probabilities without further information. The conditional probabilities:  $(1 | 1)$ ,  $(1 | 2)$  refer to the probability that information  $i_1$  occurs under states  $s_1$  and  $s_2$ , respectively. The same applies to  $(2 | 1)$  and  $(2 | 2)$ . This conditional probabilities can be estimated from a historical record of events.

For instance, connecting with our previous example, let us assume that  $s_1$  represents a market raise state and  $s_2$  a market fall state, and  $a_1$  and  $a_2$  two financial products. We may purchase a financial report to guide our investment decision, at a cost. The report has a previous record of predicting 95% of the times a market fall correctly, that is  $(1 | 1) = 0.95$  and consequently  $(1 | 2) = 0.05$ . The record also tells us that the forecast is right 90% of the time when the market falls, therefore  $(2 | 2) = 0.9$  and  $(2 | 1) = 0.1$ . With these values, we can calculate the revised probabilities and solve the tree to determine whether to incorporate the report or not.

### Value of Perfect Information

The previous section described how to calculate the value of a decision tree after incorporating additional information, which may come at a cost. An obvious question that arises is, how much should the decision maker be willing to pay for additional information? This quantity is known as the Value of Perfect Information. Perfect Information is an abstract concept that represents information that would make the occurrence of a state of nature certain. For instance, a forecast report which can tell us with total certainty which is going to be the trend of the market in the future. Obviously, this type of forecast does not actually exist in reality, but we can model its effect in the decision tree and solve it to calculate its final value. The **Expected Value with Perfect Information (EVPI)** is the expected final value of the tree given that we had perfect information. It is also an abstract concept to assess the value of the decision-making problem with access to perfect a priori information.

For instance in the example above, perfect information means that the forecast report is always right, and therefore we can plug in  $(1 | 1) = 1$  and  $(2 | 2) = 1$  since we know that the information about the states of the report is always right, and since it is always right, then also this means that  $(1 | 2) = 0$  and  $(2 | 1) = 0$ . This simplifies the tree and the value of the node with additional information becomes:

$$EVPI = P(s_1) * \max(v_1, v_3) + P(s_2) * \max(v_2, v_4)$$

Since this is the maximum value of that node provided that we had perfect a priori information, we can compare the EMV that we obtain solving the report to the EVPI to see how close we are to perfect information. This difference is known as the **Value of Perfect Information (PVI)**:

$$VPI = EVPI - EMV$$

### 5.1.3 Decision trees in machine learning

Try me



## Introduction

So far, we have seen how decision trees can be used as tools to support decision-making in complex decision problems. However, decision trees provide the basis for a family of machine learning techniques. The idea behind these techniques is to learn from training data a decision tree that can be used to solve a machine learning tasks. Normally, decision trees are applied in **classification problems** (recall that these problems the objective is to classify input data into known categories based on the features available in the training data).

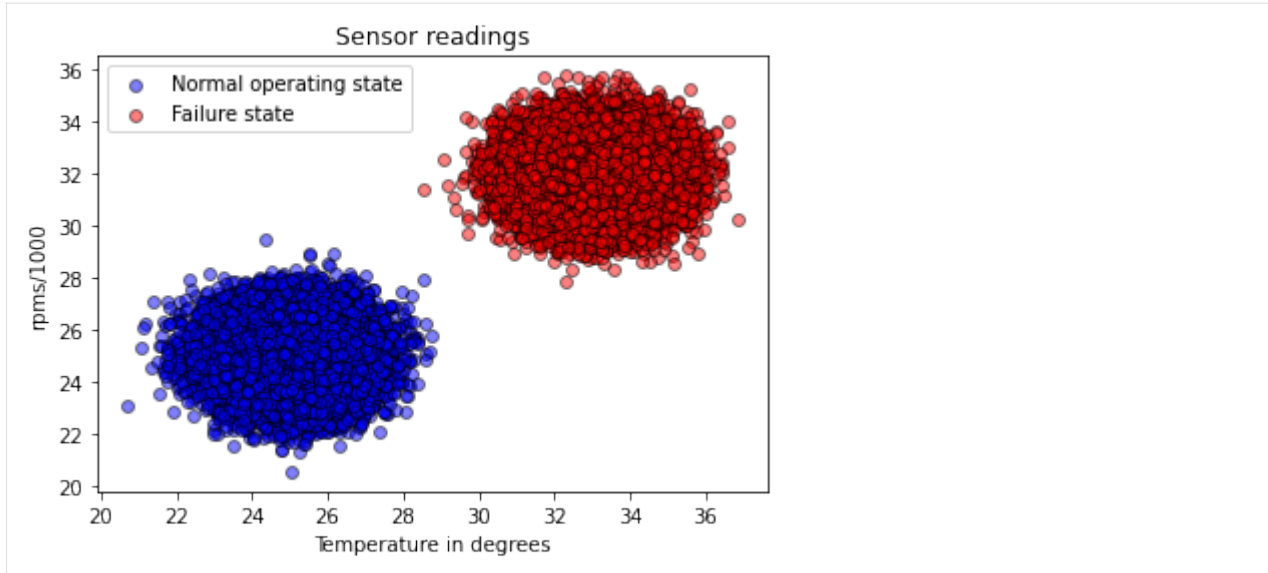
The objective is to generate a decision tree which represents the decisions that the machine will take to classify a data sample based on its features. For instance, Let's generate some synthetic data representing sensor data from two sensors, temperature and encoding speed, in two different operating states, normal operation and failure. The following script generates some synthetic data that we can use to guide the example:

```
[ ]: from sklearn.datasets import make_blobs
import numpy as np
import matplotlib.pyplot as plt

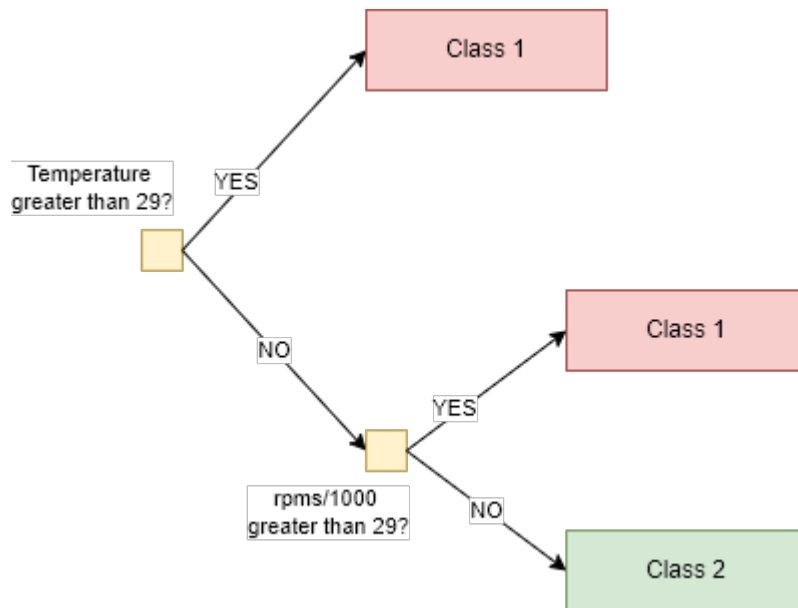
n_samples = 50000
centers = [(25, 25), (33, 32)]
X, y = make_blobs(n_samples=n_samples, centers=centers, shuffle=False, random_
↳state=42)
plt.figure()
y_unique = np.unique(y)
colors = ['blue', 'red']
classes = ['Normal operating state', 'Failure state']
for this_y, color in zip(y_unique, colors):
    this_X = X[y == this_y]

    plt.scatter(
        this_X[:, 0],
        this_X[:, 1],
        c=color,
        alpha=0.5,
        edgecolor="k",
        label=classes[this_y],
    )
plt.legend(loc="best")
plt.ylabel('rpms/1000')
plt.xlabel('Temperature in degrees')
plt.title("Sensor readings")

Text(0.5, 1.0, 'Sensor readings')
```



Analyzing the data, we can see that the machine is in normal operating state if the rpms are below 29000 rpms and the temperature below 29 degrees Celsius. The decision tree below shows how these decisions perform the classification.



As you can see in the figure, we first check the temperature reading and compare it to a threshold value, and it will take one decision branch or the other depending on whether the reading is greater or equal to the threshold. Then, in a subsequent decision node, it will check the readings of the encoder and depending on the value, and the branch, determine if the machine is in normal operating state or in failure state.

The script below represents the data and the resulting decision regions:

```
[ ]: plt.figure()

for this_y, color in zip(y_unique, colors):
    this_X = X[y == this_y]

    plt.scatter(
```

(continues on next page)

(continued from previous page)

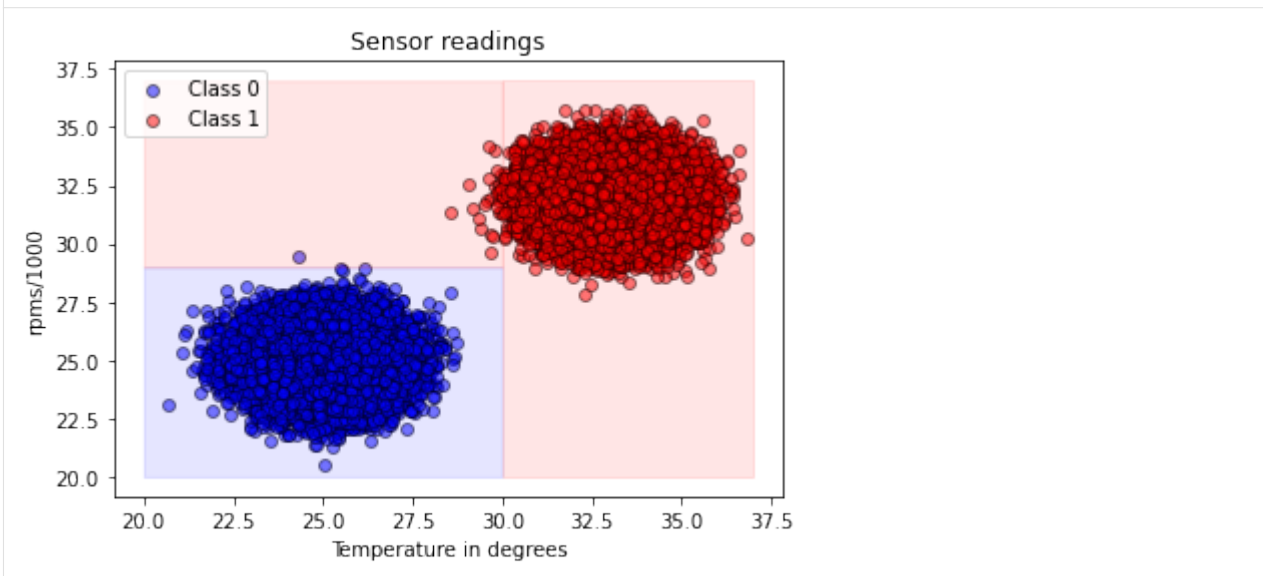
```

        this_X[:, 0],
        this_X[:, 1],
        c=color,
        alpha=0.5,
        edgecolor="k",
        label="Class %s" % this_y,
    )

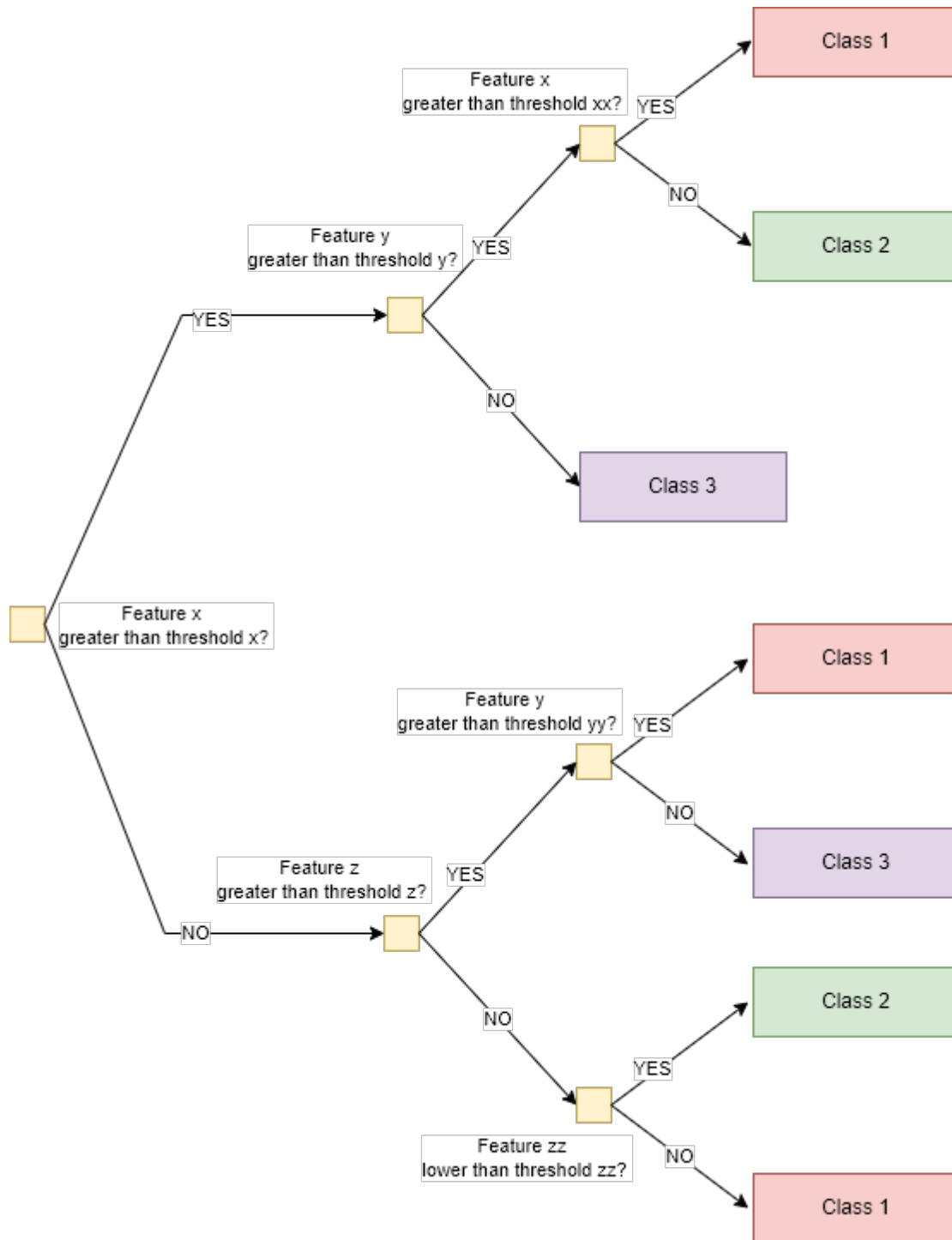
x = np.linspace(20, 37, 1000)
sep1 = 0*x + 29
sep2 = 0*x + 20
plt.fill_between(x, sep1, sep2, where=x<=30, alpha=0.1, color=colors[0])
sep3 = 0*x + 37
plt.fill_between(x, sep3, sep1, where=x<=30, alpha=0.1, color=colors[1])
plt.fill_between(x, sep3, sep2, where=x>30, alpha=0.1, color=colors[1])
plt.legend(loc="best")
plt.ylabel('rpms/1000')
plt.xlabel('Temperature in degrees')
plt.title("Sensor readings")

```

Text(0.5, 1.0, 'Sensor readings')



Now, the objective is to develop an algorithm that is able to learn this decision tree from training data, that is, use the training data to decide which decision nodes are needed at each stage and what are the values of the thresholds that need to be applied. The result is a decision tree that can be used to classify new data, with the following generic structure:



The following section describes the most common strategy used in decision trees algorithms to obtain these decision trees.

## Decision Tree Learning Algorithms

### Strategy: Recursive Binary Splitting

The most common strategy used to build decision trees from training data consists in applying a **greedy approach** to split the dataset into subsets, step by step, choosing at every step the feature that *best splits* the data into two subsets, until each subset contains only data samples from one class, or until the subsets are *homogeneous* or *pure* enough, or in other words, the level of **entropy** of the data is low enough. Moreover, the strategy is to iteratively select one feature to split the data until:

- All (or almost all) data samples in a leaf node belong to the same class (minimum impurity or minimum entropy).
- The tree has reached the maximum pre-defined number of splits (maximum depth).
- The tree has tried all features and found that none of them is useful to split the data anymore.

There are some concepts in the paragraph above that we need to unpack, mainly related to the concepts of data homogeneity and entropy.

#### Data Homogeneity

Data homogeneity is the condition that all data samples in a leaf node belong to the same class. The **Gini** impurity measure is a measure of data homogeneity in a dataset  $S$ ), expressed as:

$$\text{Gini}(S) = 1 - \sum_{i=1}^c p_i^2$$

Where  $C = 1, 2, \dots, c$  is the set of possible classes, and  $p_i$  is the probability of class  $i$ , defined as the number of data samples of class  $i$  divided by the total number of data samples in the dataset. From the definition above, we can see that the impurity is 0 if all the data samples in the dataset are of the same class.

#### Entropy

In information theory, the entropy is a measure of the level of uncertainty in a system. Formally, the entropy of a system is the sum of the probability of each possible state of the system multiplied by the logarithm of the probability of each possible state of the system:

$$\text{Entropy}(S) = \sum_{i=1}^c p_i \log_2 p_i$$

To understand this measure, consider tossing a coin. If the coin is fair, the probability of getting heads is equal to the probability of getting tails, and the entropy of the system is 1, representing total disorder, or chaos, because the uncertainty of the result of the toss is maximum. However, if the coin is completely biased, for instance, the probability of getting heads is 1, then the entropy is 0, because the result is certain, and completely predictable.

We can see that the measure of entropy and the measure of impurity are equivalent, in the sense that we can use the entropy to assess data homogeneity in a data subset.



## Recursive partitioning

As mentioned above, the tree is built by partitioning the dataset into two subsets, selecting in each iteration the feature and the threshold that maximizes data homogeneity (minimizes entropy or impurity). The partition is **recursive** because it is applied to every node of the tree (that is, the first partition results into two subsets, and then the partition is applied again to each subset, resulting in four subsets, and so forth).

At every iteration, the data at any node  $m$  is partitioned into two datasets, and the data Homogeneity of the two datasets is evaluated for every candidate split  $\theta$ , expressed as two decision variables, the feature  $j$  and a threshold  $t_m$ :  $\theta = (j, t_m)$ . Then, for instance, gradient descent methods can be used to find the optimal  $\theta$  for the current node  $m$  that minimizes the data homogeneity.

Formally, let the data subset at node  $m$  be  $S_m$ , consisting of  $N_m$  data samples, each containing a feature vector  $x$  and a label  $y$ . The candidate  $\theta = (j, t_m)$  partition the data into two subsets  $S_m^{left}(\theta)$  such that:

$$S_m^{left}(\theta) = x \in S_m : x_j \leq t_m$$

And and  $S_m^{right}(\theta)$  such that:

$$S_m^{right}(\theta) = x \in S_m : x_j > t_m$$

Let us now define the loss function for the current node  $m$ :

$$L(S_m, \theta) = \frac{N_m^{left}}{N_m} H(S_m^{left}(\theta)) + \frac{N_m^{right}}{N_m} H(S_m^{right}(\theta))$$

where  $N_m^{left}$  and  $N_m^{right}$  are the number of data samples in the left and right subsets, respectively, and  $H(S)$  is a measure of the data entropy or impurity in the dataset  $S$ .

This way, the learning problem is reduced to a minimization of the loss function, which is equivalent to finding the  $\theta$  that minimizes the data Homogeneity.

## Information gain

Another approach is to maximise the information gain, which is the difference between the data Homogeneity of the data subset at node  $m$  and the data homogeneity of the two resulting subsets.

$$I(S_m) = H(S_m) - \frac{N_m^{left}}{N_m} H(S_m^{left}) - \frac{N_m^{right}}{N_m} H(S_m^{right})$$

That is, the objective now is to find the  $\theta$  that maximizes the difference between the data homogeneity before and after the split.

## Algorithms

There are different algorithms that use this approach to build decision trees, the most popular being Iterative Dichotomiser 3 (ID3), its successors C4.5 and C5.0., and finally the Classification and Regression Trees (CART) algorithm.

ID3 uses the greedy recursive partitioning algorithm finding for each node the feature and threshold that maximizes the information gain, with the limitation that the features must be categorical or qualitative (that is, it can only take one of a finite number of values). C4.5 and C5.0 on the other hand remove this restriction. CART is very similar to C4.5, but it provides additional support to solve regression problems.

### Pruning

All the algorithm above have a common drawback: They tend to overfitting the training data. To overcome this, they use a pruning strategy that removes the nodes that do not improve the model performance. One common pruning strategy is cost complexity pruning, where the nodes are removed (pruned) if they do not improve the model performance. Minimum cost complexity pruning finds the best trade-off between the tree complexity (expressed as the number of terminal nodes) and performance.

## 5.1.4 Machine learning with decision trees

### Try me



In this notebook we are going to train a decision tree to solve a classification problem in Python.

### Problem definition

The goal is to train a model to get the category of a wine according to a set of physicochemical properties (e.g. alcohol percentage, color intensity and hue, or flavonoids). Imagine that the categories have been assigned by our customers on a given number of wines (e.g. they have determined which wines that are awesome, which wines are ok, or which ones are only drinkable). Then, with such a model, given a new wine, we could predict its category based on the physicochemical properties of the wine provided by a physicochemical test.

The objective is then to train a model to predict the category of the wine based on 13 different properties of wine.

### Load data

We are going to use a dataset provided by [SKlearn](#). The dataset contains 177 samples of physicochemical properties of different types of wines, classified into three different classes, class 1, class 2, and class 3.

The dataset in SKLearn is loaded with the function `load_wine`. The object returned contains the following properties:

- `data`: A 2D Numpy array where each row is a different physicochemical test performed over a different wine (that is, each row represents the properties of a wine) and each column contains the data of a specific property.
- `target`: A 1D Numpy (column vector) array where each row represents the category of the corresponding wine
- `feature_names`: The names of the columns in `data`
- `target_names`: The names of the wine categories

You can check the [docs](#) for further details.

Let's load the data into a Pandas dataframe so that we can explore its properties:

```
[ ]: from sklearn.datasets import load_iris, load_wine
import pandas as pd

# Load the wine dataset
wine_data = load_wine()
# Let's create a dataframe to explore the data
df_wine = pd.DataFrame(wine_data.data, columns=wine_data.feature_names)
display(df_wine)
```

```

    alcohol  malic_acid  ash  alcalinity_of_ash  magnesium  total_phenols  \
0      14.23      1.71  2.43      15.6      127.0      2.80
1      13.20      1.78  2.14      11.2      100.0      2.65
2      13.16      2.36  2.67      18.6      101.0      2.80
3      14.37      1.95  2.50      16.8      113.0      3.85
4      13.24      2.59  2.87      21.0      118.0      2.80
..      ...      ...  ...      ...      ...      ...
173    13.71      5.65  2.45      20.5      95.0      1.68
174    13.40      3.91  2.48      23.0      102.0      1.80
175    13.27      4.28  2.26      20.0      120.0      1.59
176    13.17      2.59  2.37      20.0      120.0      1.65
177    14.13      4.10  2.74      24.5      96.0      2.05

    flavanoids  nonflavanoid_phenols  proanthocyanins  color_intensity  hue  \
0           3.06                0.28                2.29            5.64  1.04
1           2.76                0.26                1.28            4.38  1.05
2           3.24                0.30                2.81            5.68  1.03
3           3.49                0.24                2.18            7.80  0.86
4           2.69                0.39                1.82            4.32  1.04
..      ...      ...      ...      ...      ...
173         0.61                0.52                1.06            7.70  0.64
174         0.75                0.43                1.41            7.30  0.70
175         0.69                0.43                1.35           10.20  0.59
176         0.68                0.53                1.46            9.30  0.60
177         0.76                0.56                1.35            9.20  0.61

    od280/od315_of_diluted_wines  proline
0                3.92      1065.0
1                3.40      1050.0
2                3.17      1185.0
3                3.45      1480.0
4                2.93       735.0
..      ...      ...
173             1.74       740.0
174             1.56       750.0
175             1.56       835.0
176             1.62       840.0
177             1.60       560.0

[178 rows x 13 columns]

```

### Understanding the data

Now, let us explore the data so that we understand the dataset better. First, we use the `describe()` method of the pandas dataframe to show some statistics of each class:

```

[ ]: from IPython.core.display import Markdown
target_classes = range(0, 3)

# For each class
for target_class in target_classes:
    # Show some markdown to introduce the table
    display(Markdown('### ' + wine_data.target_names[target_class]))
    display(Markdown('Find below the statistiscal details of the physicochemical_
↳properties of the class'))

    # Mask the rows of the class

```

(continues on next page)

(continued from previous page)

```
class_mask = wine_data.target == target_class
class_df = df_wine.iloc[class_mask]
```

```
# Display the resulting dataframe
display(class_df.describe())
```

###class\_0

Find below the statistiscal details of the physicochemical properties of the class

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	\
count	59.000000	59.000000	59.000000	59.000000	59.000000	
mean	13.744746	2.010678	2.455593	17.037288	106.338983	
std	0.462125	0.688549	0.227166	2.546322	10.498949	
min	12.850000	1.350000	2.040000	11.200000	89.000000	
25%	13.400000	1.665000	2.295000	16.000000	98.000000	
50%	13.750000	1.770000	2.440000	16.800000	104.000000	
75%	14.100000	1.935000	2.615000	18.700000	114.000000	
max	14.830000	4.040000	3.220000	25.000000	132.000000	

	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	\
count	59.000000	59.000000	59.000000	59.000000	
mean	2.840169	2.982373	0.290000	1.899322	
std	0.338961	0.397494	0.070049	0.412109	
min	2.200000	2.190000	0.170000	1.250000	
25%	2.600000	2.680000	0.255000	1.640000	
50%	2.800000	2.980000	0.290000	1.870000	
75%	3.000000	3.245000	0.320000	2.090000	
max	3.880000	3.930000	0.500000	2.960000	

	color_intensity	hue	od280/od315_of_diluted_wines	proline
count	59.000000	59.000000	59.000000	59.000000
mean	5.528305	1.062034	3.157797	1115.711864
std	1.238573	0.116483	0.357077	221.520767
min	3.520000	0.820000	2.510000	680.000000
25%	4.550000	0.995000	2.870000	987.500000
50%	5.400000	1.070000	3.170000	1095.000000
75%	6.225000	1.130000	3.420000	1280.000000
max	8.900000	1.280000	4.000000	1680.000000

###class\_1

Find below the statistiscal details of the physicochemical properties of the class

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	\
count	71.000000	71.000000	71.000000	71.000000	71.000000	
mean	12.278732	1.932676	2.244789	20.238028	94.549296	
std	0.537964	1.015569	0.315467	3.349770	16.753497	
min	11.030000	0.740000	1.360000	10.600000	70.000000	
25%	11.915000	1.270000	2.000000	18.000000	85.500000	
50%	12.290000	1.610000	2.240000	20.000000	88.000000	
75%	12.515000	2.145000	2.420000	22.000000	99.500000	
max	13.860000	5.800000	3.230000	30.000000	162.000000	

	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	\
count	71.000000	71.000000	71.000000	71.000000	
mean	2.258873	2.080845	0.363662	1.630282	
std	0.545361	0.705701	0.123961	0.602068	
min	1.100000	0.570000	0.130000	0.410000	
25%	1.895000	1.605000	0.270000	1.350000	

(continues on next page)

(continued from previous page)

50%	2.200000	2.030000	0.370000	1.610000
75%	2.560000	2.475000	0.430000	1.885000
max	3.520000	5.080000	0.660000	3.580000
	color_intensity	hue	od280/od315_of_diluted_wines	proline
count	71.000000	71.000000	71.000000	71.000000
mean	3.086620	1.056282	2.785352	519.507042
std	0.924929	0.202937	0.496573	157.211220
min	1.280000	0.690000	1.590000	278.000000
25%	2.535000	0.925000	2.440000	406.500000
50%	2.900000	1.040000	2.830000	495.000000
75%	3.400000	1.205000	3.160000	625.000000
max	6.000000	1.710000	3.690000	985.000000

###class\_2

Find below the statistical details of the physicochemical properties of the class

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	\
count	48.000000	48.000000	48.000000	48.000000	48.000000	
mean	13.153750	3.333750	2.437083	21.416667	99.312500	
std	0.530241	1.087906	0.184690	2.258161	10.890473	
min	12.200000	1.240000	2.100000	17.500000	80.000000	
25%	12.805000	2.587500	2.300000	20.000000	89.750000	
50%	13.165000	3.265000	2.380000	21.000000	97.000000	
75%	13.505000	3.957500	2.602500	23.000000	106.000000	
max	14.340000	5.650000	2.860000	27.000000	123.000000	
	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	\	
count	48.000000	48.000000	48.000000	48.000000	48.000000	
mean	1.678750	0.781458	0.447500	1.153542		
std	0.356971	0.293504	0.12414	0.408836		
min	0.980000	0.340000	0.170000	0.550000		
25%	1.407500	0.580000	0.397500	0.855000		
50%	1.635000	0.685000	0.470000	1.105000		
75%	1.807500	0.920000	0.530000	1.350000		
max	2.800000	1.570000	0.630000	2.700000		
	color_intensity	hue	od280/od315_of_diluted_wines	proline		
count	48.000000	48.000000	48.000000	48.000000		
mean	7.396250	0.682708	1.683542	629.895833		
std	2.310942	0.114441	0.272111	115.097043		
min	3.850000	0.480000	1.270000	415.000000		
25%	5.437500	0.587500	1.510000	545.000000		
50%	7.550000	0.665000	1.660000	627.500000		
75%	9.225000	0.752500	1.820000	695.000000		
max	13.000000	0.960000	2.470000	880.000000		

The statistical information is useful, but it is impossible or in most cases really hard to determine manually the rules to separate the different classes in each of the 19 properties in the data set. Just to certify this, Let's display some data using Pyplot:

```
[ ]: from matplotlib import pyplot as plt

FIG_SIZE = (10, 7)
# Create figure
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=FIG_SIZE)
```

(continues on next page)

(continued from previous page)

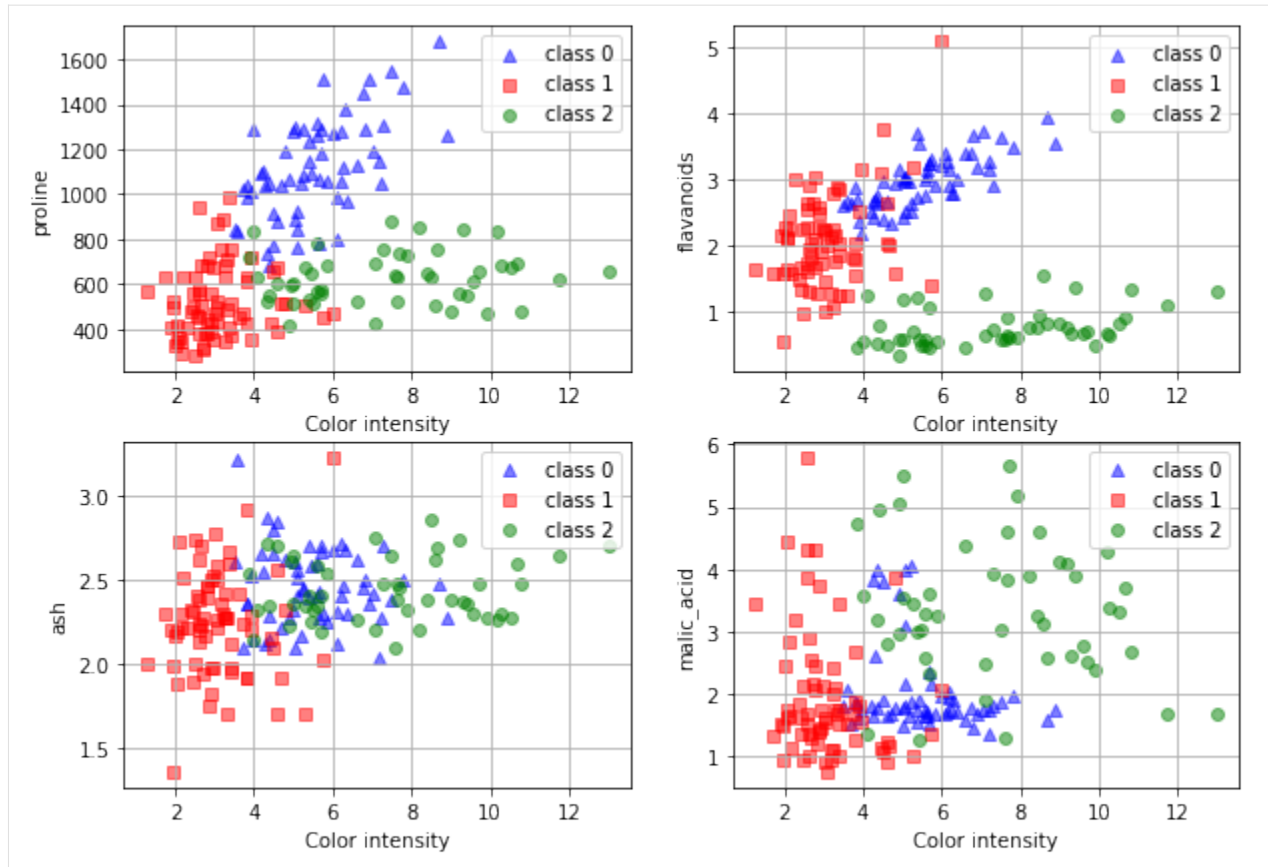
```
# Get some columns
columns = ['proline', 'flavanoids', 'ash', 'malic_acid']

# Now, we define some styles and colors for the 3 target classes
target_classes = range(0, 3)
colors = ("blue", "red", "green")
markers = ("^", "s", "o")

#Let's iterate on the target classes, so that we plot the values in
# of each with different colors
for target_class, color, marker in zip(target_classes, colors, markers):
    # The color intensity is going to be the x-axis in every plot:
    x_data = df_wine['color_intensity'][wine_data.target == target_class]
    # Now Let's get 4 different columns to plot in each of the subplots
    for idx in range(0,4):
        # Get the axes in which we are going to plot our data
        ax = axes[idx//2][idx%2]
        col = columns[idx]

        # Let's get the data, similar as we did for the x-axis
        y_data = df_wine[columns[idx]][wine_data.target == target_class]

        # Plot the data in the corresponding subplot
        ax.scatter(
            x=x_data,
            y=y_data,
            color=color,
            label=f"class {target_class}",
            alpha=0.5,
            marker=marker,
        )
        ax.set_xlabel("Color intensity")
        ax.set_ylabel(columns[idx])
        ax.legend(loc="upper right")
        ax.grid()
```



In the plot, we see 4 different properties (proline, flavanoids, ash, malic acid) against the color intensity. We can see that the data is not linearly separable in any of them, and this is only 4 out of  $13^2$  possible pairs of parameters, it is clear that is unfeasible to come up with some simple rules to separate the datasets easily. But can we get the decision tree to separate the data in the three classes automatically, using a decision tree? Let's find out in the following section.

## Training a decision tree to classify the dataset

### Dividing the dataset into training and validation data

The first step is to divide the dataset into the training dataset and the validation dataset. With this step, we want to ensure that we leave some portion of the dataset so that we can validate our model. In the code below, the size of the data left out to validate the model is around 30% of the available data.

We use the function `train_test_split` to split the data between the training data set and the validation dataset. (`test_size=0.3`).

```
[ ]: from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(wine_data.data, wine_data.target,
↪ test_size = 0.3, random_state=42)
```

Note that the `train_test_split` function returns four values:

- `X_train`: This Numpy array contains the features in the training data set, each row corresponds to one of the tests in the training data, and each column to a different feature (physicochemical property).
- `y_train`: These are the categories of the wines in the training dataset.
- `X_test`: This Numpy array contains the features in the test data set, similar to `X_train`.

- `y_test`: And these are the categories of the wines in the test dataset.

Here are the full details of the `train_test_split` function.

### Train the model

We are going to train our model, using the training data. We only need to create an instance of a `DecisionTreeClassifier` class, and use the class method `fit` passing the training data:

```
[ ]: from sklearn.tree import DecisionTreeClassifier
      clf_model = DecisionTreeClassifier(random_state=0)
      clf_model.fit(X_train, y_train)
```

```
DecisionTreeClassifier(random_state=0)
```

All the details of the `DecisionTreeClassifier` class can be found in [here](#).

### Validate the decision tree

#### Checking the accuracy

Ok, once that we have trained our model, let us test its, [accuracy](#). We are going to use some built-in functions provided in SKLearn. The first one, `accuracy_score` computes the accuracy of a model, passing as parameters, the real categories and the predicted categories. So Let's predict the categories in the test data set and get teh accuracy score:

```
[ ]: from sklearn.metrics import accuracy_score

      # Use the method predict to predict the category of the wines in the test dataset
      y_predict = clf_model.predict(X_test)

      # Calculate the accuracy with accurate_score, using the real categories in the test_
      ↪dataset
      # and the predicted categories
      accuracy_score(y_test, y_predict)
```

```
0.9629629629629629
```

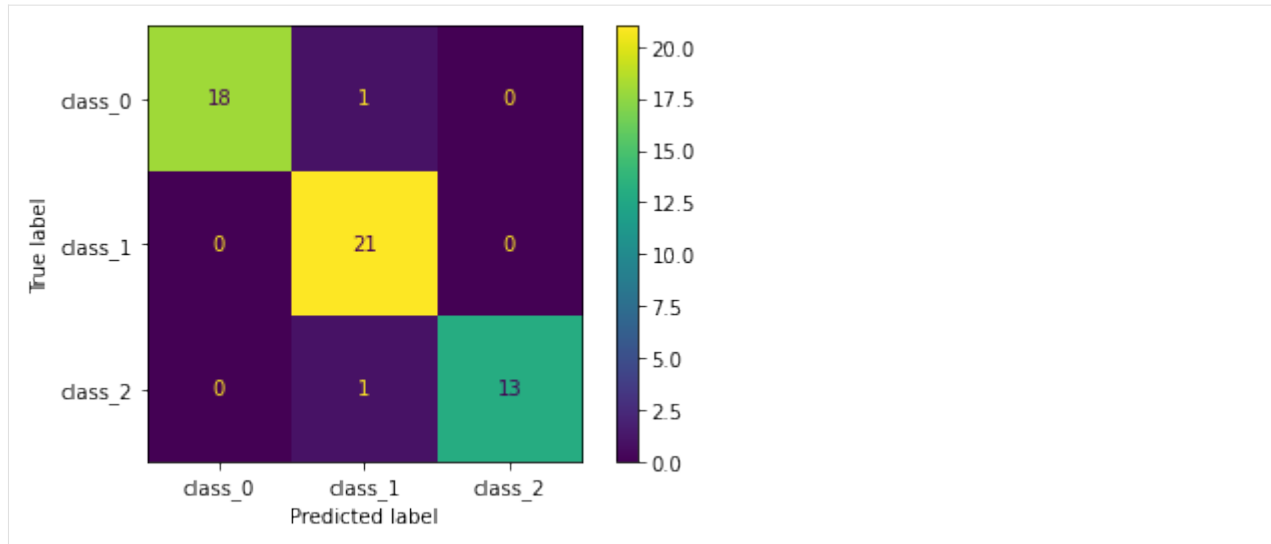
### Represent the Confusion matrix

SKLearn also allow us to display the confusion matrix given a set of real categories and a set of predicted categories, with the function `confusion_matrix` (see docs [here](#)), and display the confusion matrix with `ConfusionMatrixDisplay` (see docs [here](#)):

```
[ ]: from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
      from matplotlib import pyplot as plt
      cm = confusion_matrix(y_test, y_predict)
      disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                                   display_labels=wine_data.target_names)

      disp.plot()
      plt.show()
```





### Render the decision tree

Finally, Let's render the decision tree using Graphviz and the provided class method `export_graphviz`. Read the docs [here](#).

```
[ ]: from sklearn import tree
import graphviz
dot_data = tree.export_graphviz(clf_model,
                               out_file=None,
                               feature_names=wine_data.feature_names,
                               class_names=wine_data.target_names,
                               filled=True,
                               label='none',
                               leaves_parallel=True,
                               proportion=True,
                               rounded=True,
                               special_characters=True)
graph = graphviz.Source(dot_data)

graph
```

### Testing the model

You are now ready to test the model! You only need to create an array with where the Physicochemical properties are added in the right order:

Physicochemical properties	Array position	Example
alcohol	0	15.0
malic_acid	1	1.7
ash	2	2.1
alcalinity_of_ash	3	15.0
magnesium	4	105
total_phenols	5	2.3
flavanoids	6	1.32
nonflavanoid_phenols	7	0.5
proanthocyanins	8	2.0
color_intensity	9	3.9
hue	10	1.1
od280/od315_of_diluted_wines	11	3.0
proline	12	890

The following code cell shows the class of a wine that has the physicochemical properties in the example:

```
[ ]: example = [[15.0, 1.7, 2.1, 15.0, 105, 2.3, 1.32, 0.5, 2.0, 3.9, 1.1, 3.0, 890]]
example_class = clf_model.predict(example)
print(example_class)
```

### Questions for analysis

1. Now that you have trained the model, you can use the `predict` method to get the category of a new wine. Use this method to get the category of the wonderful Casa Quemada (CQ) from Viña Elena’s winerie, given its physicochemical properties:

Physicochemical properties	CQ
alcohol	14.38
malic_acid	1.83
ash	2.36
alcalinity_of_ash	17.2
magnesium	104
total_phenols	2.42
flavanoids	1.39
nonflavanoid_phenols	0.42
proanthocyanins	1.97
color_intensity	3.84
hue	1.23
od280/od315_of_diluted_wines	2.87
proline	990

```
[ ]: # Enter the physicochemical properties of the wine you would like to test, in order_
      ↪ and using commas to separate the values
cq = [[]]
cq_class = clf_model.predict(cq)
print(cq_class)
```

How good is the wine? Now look at the decision rules in the tree and solve the tree manually. Do you get the same result?

2. The confusion matrix shows the errors made by the model, whereas the accuracy is just a metric of the number of errors. Imagine that you use the different categories to guide your marketing policies, and that you will invest more money in the marketing of wines you expect will have better ratings (being class 2 the category of best rated wines) more than in wines with lower expectations. Do you think that the confusion matrix provides valuable information for analysis?
3. The accuracy is just one out of different metrics used to evaluate machine learning models. Another interesting metric is called precision, and is also implemented in SKLearn. You can check the docs in this [link](#). Based on your response to the previous question, do you think that the precision is a relevant metric in this use case scenario?
4. Run the experiment again with different sizes of the test data (e.g. 20% and 40%). Recall the concepts of overfitting and underfitting and note what happens with the accuracy. What are your conclusions?
5. Look now to the right hand of the decision tree. Looking to the left, it would be interesting to plot the proline against the malic acid. Check the code cells used to render some properties against the color intensity to plot the proline against the malic acid for a. all wines, b. only wines in classes 0 and 1 c. wines where the color intensity is greater than 3.82 and flavanoids greater than 1.4.
6. SKLearn has other [datasets](#) you can use to step up your machine learning skills. Do you dare to train a model for a different training dataset?

## 5.2 Decision Theory Exercises

In this section you have a collection of decision tree problems sorted by difficulty:

- **Easy problems:** You can tackle these problems after the first lesson on decision trees.
- **Normal problems:** These problems require that you are familiar with concepts like decision trees and perfect information value.
- **Hard problems:** This is the type of problems that definitively get you ready for the exam.

### 5.2.1 Starting Five

#### Problem Definition

You are the trainer of a basketball team playing an inter-university tournament over a weekend. You won the first game of the tournament yesterday, and now you have to decide whether to play with the same starting 5 or make changes in the team. You want to assess your decision considering the possibilities of winning ( $p_w$ ) or loosing ( $p_l$ ) the game and of having any player injured ( $p_i$ ) or all healthy ( $p_h$ ) after the game. Since it is a friendly tournament, your priority is that none of your classmates ends up injured, so you establish the following priorities between the utility values of the possible outcomes:  $v_1$  (victory – no injuries)  $>$   $v_2$  (loose – no injuries)  $>$   $v_3$  (win – any injured)  $>$   $v_4$  (loose – any injured).

- a. Draw a decision tree indicating the probabilities and utility values of the possible outcomes, considering that the probability of winning or loosing and the probability of having any player injured or all healthy are independent and also independent of your decision to make changes in the team.
- b. How would the decision tree change if you consider that the probability of having any player injured is not independent of your decision to make changes?.

## 5.2.2 Training Pokemon

### Problem Definition

Ash Ketchum is a renowned Pokémon trainer. He thinks that his most beloved Pokémon, Pikachu, could have been injured in the last battle of the Winners Trophy play-offs. Therefore, the question is:

should he use Pikachu in the next battle against Team Rocket?

The trainer thinks that there is a probability of 0.1 of victory. However, if Pikachu participates in the fight, his injury could worsen. Ash thinks that there is a probability of 0.2 that Pikachu is actually injured from the last battle.

The utilities (in points) that the trainer employs are:

- if Pikachu fights, and Ash wins the battle, and Pikachu is not injured, +100;
- if Pikachu fights, and Ash wins the battle, and Pikachu is injured, +50;
- if Pikachu fights, and Ash loses the battle, and Pikachu is not injured, 0;
- if Pikachu fights, and Ash loses the battle, and Pikachu is injured, -50
- If Pikachu does not fight and Pikachu is injured, the utility is -10,
- If Pikachu does not fight and Pikachu is not injured, the utility is 0.

Ash wants to use decision theory to support his decision-making in the battle against Team Rocket.

- Draw the decision tree for this problem considering that the probabilities of the result of the battle and the injury are independent.
- Determine which is the best decision for Ash.
- Ash could acquire further information about the probabilities of the result of the battle. The additional information determines the result of the battle (Win or lose), and based on previous results, it has a certain accuracy to successfully predict the result of the battle. Draw again the decision tree to incorporate into the decision problem the acquisition of this imperfect information about the result of the battle, indicating in the corresponding branches the revised probabilities

## 5.2.3 Planning a factory site

### Problem Definition

Dinoco, a multinational firm in the automobile sector, is deciding on whether to build an assembly plant in Brazil or in Mississippi (USA). The cost to build this plant in Brazil is 10 million euros and the cost of building the plant in Mississippi is 20 million. However, if the firm builds the plant in Brazil and local demand drops over the following 5 years, the project will be stopped and the firm will lose 10M€ (and it will still have to build a plant in Mississippi). A priori, Dinoco believes that the probability of demand for cars dropping in Brazil over the following 5 years is 20%. For 1M€, Dinoco can hire a market research firm to analyse demand for cars in Brazil, which will indicate whether demand for cars will drop or not. The market research firm's record indicates that his record shows that they are able to predict the occurrence of a drop in demand with a 95% success rate, and 90% when the demand does not drop.

- Draw the decision tree.
- Solve the tree and find the best alternative for Dinoco.
- Calculate the Imperfect Information Value and the Perfect Information Value associated to the market research (demand in Brazil)

## 5.2.4 Selecting Suppliers

### Problem Definition

Your company, Altered Silicon, manufactures solar energy panels, and it has received an urgent order for 40 panels from a new German customer which is willing to pay 1,000€ per panel. Altered Silicon has committed its production capacity to other orders, and the only alternative left to serve the new order is to subcontract the production involved.

Altered Silicon has located two possible suppliers: - Supplier A: based in Romania, placed an offer of 500€ per panel - Supplier B: from Poland, which would sell each panel for 550€

The cost to transport one batch of 40 panels from either of the two suppliers to the German firm is 1,000€. Nonetheless, the customer's technical specifications are very demanding, so Altered Silicon estimates that Supplier A could supply one correct batch with a probability of 70%, whereas Supplier B could do it with a probability of 80%. If the batch were incorrect, Altered Silicon could sell it to another customer at the price of 200€ per panel. Altered Silicon is considering the possibility of asking for a test panel from each supplier to test it before placing an order. Altered Silicon's testing system guarantees that a faulty panel is always detected, while there is a 5% margin of error with correct panels. The testing system is prepared to assess two test panels, and the overall cost (cost of the two panels + transport + cost of the test) comes to 2,000€.

- a** Draw the corresponding decision tree and indicate with a letter per branch the associated probability or the associated cost (depending on the case)
- b** Solve the decision tree
- c** Up to how much should SUN2 pay for the test?

## 5.2.5 Blind wine tasting

### Problem Definition

María and Alberto (names have been selected randomly, any resemblance with reality is pure coincidence) are two classmates that want to celebrate the end of the course with a game. They will go to their favourite bar and Alberto will randomly select a bottle of wine from the wine list. María will do a blind testing of the wine, with no possibility to see the wine selected by Alberto. If María guesses the type of wine (red or white) just by holding her glass, Alberto will pay 20€. Otherwise, if she fails, María will pay Alberto 10€. She may also taste the wine for 5€ (that is, Alberto will discount 5€ of the bet if she guesses the type of wine, or María will pay 5€ extra if she does not).

María thinks she can use decision theory to improve her chances of winning the bet. She knows that in the wine list of their favourite bar, there are 15 bottles of red wine and 11 of white wine. She has also done some blind wine tasting to prepare for the bet and found out that it is actually not that easy to guess the colour of the wine just by tasting it blindfolded. She guessed correctly 3/4 of the times she tried a white wine and 2/4 of the times she tried a red wine.

- a** Draw the decision tree for María without taking into account that she may test the wine to get additional information.
- b** Modify the decision tree for María now taking into account that she may test the wine.
- c** What is the best alternative for María?

## 5.2.6 The eye of Agamoto

### Problem Definition

Doctor Stephen Strange is the chief surgeon at the Metropolitan General Hospital in New York. Doctor Strange needs to decide whether to perform surgery or not on a patient with a rare disease. If the patient dies, the hospital must pay a 3M€ compensation, and the cost of the surgery is estimated in 1.5M€. Based on available data, the medical department estimates that the probability of the patient surviving without surgery is 20%, while the probability of the patient surviving with surgery is 75%. Let us use operational research to determine if Doctor Strange should hold the knife:

- a. Draw the decision tree to support doctor Strange with this decision problem
- b. Solve the decision tree
- c. Doctor Strange can use a procedure with a device known as the Eye of Agamoto to get additional information of the result of the surgery, with a cost of 0.5M€. Draw the decision tree incorporating the use of the Eye of Agamoto in the decision problem

## 5.3 Decision Theory Solved Exercises

In this section you have the solution to the decision theory exercises.

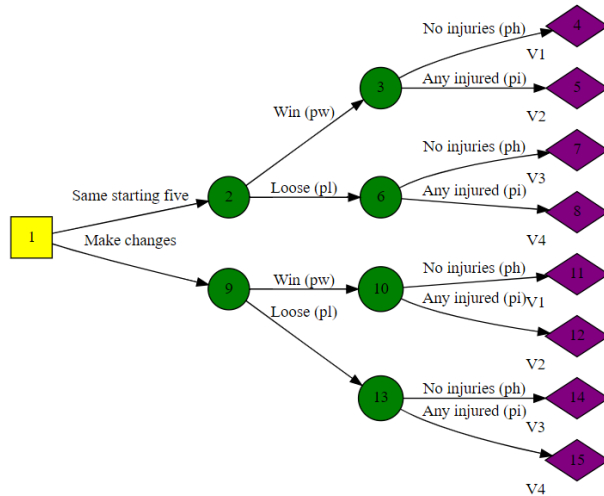
### 5.3.1 Starting Five

#### Problem Definition

You are the trainer of a basketball team playing an inter-university tournament over a weekend. You won the first game of the tournament yesterday, and now you have to decide whether to play with the same starting 5 or make changes in the team. You want to assess your decision considering the possibilities of winning ( $p_w$ ) or loosing ( $p_l$ ) the game and of having any player injured ( $p_i$ ) or all healthy ( $p_h$ ) after the game. Since it is a friendly tournament, your priority is that none of your classmates ends up injured, so you establish the following priorities between the utility values of the possible outcomes:  $v_1$  (victory – no injuries)  $>$   $v_2$  (loose – no injuries)  $>$   $v_3$  (win – any injured)  $>$   $v_4$  (loose – any injured).

- a. Draw a decision tree indicating the probabilities and utility values of the possible outcomes, considering that the probability of winning or loosing and the probability of having any player injured or all healthy are independent and also independent of your decision to make changes in the team.

The following decision tree represents the solution of the problem:



Run the code if you want to generate it yourself using Python!

```
[ ]: from graphviz import Digraph
# Create Digraph object
din = Digraph(format='png')
din.graph_attr['rankdir'] = 'LR'

# Add root node
din.node('1', shape='square', style="filled", fillcolor='yellow')

# First branch nodes
din.node('2', shape='circle', style="filled", fillcolor='green')
din.node('3', shape='circle', style='filled', fillcolor='green')
din.node('4', xlabel='V1', shape='diamond', style='filled', fillcolor='purple')
din.node('5', xlabel='V2', shape='diamond', style='filled', fillcolor='purple')
din.node('6', shape='circle', style='filled', fillcolor='green')
din.node('7', xlabel='V3', shape='diamond', style='filled', fillcolor='purple')
din.node('8', xlabel='V4', shape='diamond', style='filled', fillcolor='purple')

# Second branch
din.node('9', shape='circle', style="filled", fillcolor='green')
din.node('10', shape='circle', style='filled', fillcolor='green')
din.node('11', xlabel='V1', shape='diamond', style='filled', fillcolor='purple')
din.node('12', xlabel='V2', shape='diamond', style='filled', fillcolor='purple')
din.node('13', shape='circle', style='filled', fillcolor='green')
din.node('14', xlabel='V3', shape='diamond', style='filled', fillcolor='purple')
din.node('15', xlabel='V4', shape='diamond', style='filled', fillcolor='purple')

din.edge('1', '2', 'Same starting five')
din.edge('1', '9', 'Make changes')
din.edge('2', '3', 'Win (pw)')
```

(continues on next page)

(continued from previous page)

```
din.edge('2', '6', 'Loose (pl)')

din.edge('3', '4', 'No injuries (ph)')
din.edge('3', '5', 'Any injured (pi)')

din.edge('6', '7', 'No injuries (ph)')
din.edge('6', '8', 'Any injured (pi)')

din.edge('9', '10', 'Win (pw)')
din.edge('9', '13', 'Loose (pl)')

din.edge('10', '11', 'No injuries (ph)')
din.edge('10', '12', 'Any injured (pi)')

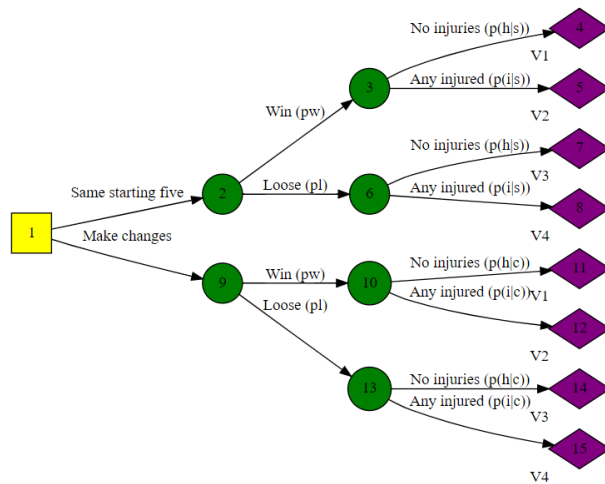
din.edge('13', '14', 'No injuries (ph)')
din.edge('13', '15', 'Any injured (pi)')

din
```

**b.** How would the decision tree change if you consider that the probability of having any player injured is not independent of your decision to make changes?

We need to take into account the conditional probabilities of: -  $p(i|s)$ : Probability of having an injury given that we play with the same (s) team. -  $p(h|s)$ : Probability of not having any injury given that we play with the same team -  $p(i|c)$ : Probability of having an injury given that we make changes (c) -  $p(h|c)$ : Probability of not having any injury given that we play with a different starting team

With this posterior probabilities, we can redraw the decision tree as follows:



This image was generated with the following Python script:

```
[ ]: from graphviz import Digraph
# Create Digraph object
din = Digraph(format='png')
din.graph_attr['rankdir'] = 'LR'

# Add root node
din.node('1', shape='square', style="filled", fillcolor='yellow')

# First branch nodes
```

(continues on next page)



(continued from previous page)

```

din.node('2', shape='circle', style="filled", fillcolor='green')
din.node('3', shape='circle', style='filled', fillcolor='green')
din.node('4', xlabel='V1', shape='diamond', style='filled', fillcolor='purple')
din.node('5', xlabel='V2', shape='diamond', style='filled', fillcolor='purple')
din.node('6', shape='circle', style='filled', fillcolor='green')
din.node('7', xlabel='V3', shape='diamond', style='filled', fillcolor='purple')
din.node('8', xlabel='V4', shape='diamond', style='filled', fillcolor='purple')

# Second branch
din.node('9', shape='circle', style="filled", fillcolor='green')
din.node('10', shape='circle', style='filled', fillcolor='green')
din.node('11', xlabel='V1', shape='diamond', style='filled', fillcolor='purple')
din.node('12', xlabel='V2', shape='diamond', style='filled', fillcolor='purple')
din.node('13', shape='circle', style='filled', fillcolor='green')
din.node('14', xlabel='V3', shape='diamond', style='filled', fillcolor='purple')
din.node('15', xlabel='V4', shape='diamond', style='filled', fillcolor='purple')

din.edge('1', '2', 'Same starting five')
din.edge('1', '9', 'Make changes')

din.edge('2', '3', 'Win (pw)')
din.edge('2', '6', 'Loose (pl)')

din.edge('3', '4', 'No injuries (p(h|s))')
din.edge('3', '5', 'Any injured (p(i|s))')

din.edge('6', '7', 'No injuries (p(h|s))')
din.edge('6', '8', 'Any injured (p(i|s))')

din.edge('9', '10', 'Win (pw)')
din.edge('9', '13', 'Loose (pl)')

din.edge('10', '11', 'No injuries (p(h|c))')
din.edge('10', '12', 'Any injured (p(i|c))')

din.edge('13', '14', 'No injuries (p(h|c))')
din.edge('13', '15', 'Any injured (p(i|c))')

din

```

## 5.3.2 Training Pokemon

### Problem Definition

Ash Ketchum is a renowned Pokémon trainer. He thinks that his most beloved Pokémon, Pikachu, could have been injured in the last battle of the Winners Trophy play-offs. Therefore, the question is:

should he use Pikachu in the next battle against Team Rocket?

The trainer thinks that there is a probability of 0.1 of victory. However, if Pikachu participates in the fight, his injury could worsen. Ash thinks that there is a probability of 0.2 that Pikachu is actually injured from the last battle.

The utilities (in points) that the trainer employs are:

- if Pikachu fights, and Ash wins the battle, and Pikachu is not injured, +100;
- if Pikachu fights, and Ash wins the battle, and Pikachu is injured, +50;
- if Pikachu fights, and Ash loses the battle, and Pikachu is not injured, 0;
- if Pikachu fights, and Ash loses the battle, and Pikachu is injured, -50
- If Pikachu does not fight and Pikachu is injured, the utility is -10,
- If Pikachu does not fight and Pikachu is not injured, the utility is 0.

Ash wants to use decision theory to support his decision-making in the battle against Team Rocket.

a. Draw the decision tree for this problem considering that the probabilities of the result of the battle and the injury are independent.

Ash has one decision to make, with two different alternatives:

- Pikachu fights: Pikachu fights in the next tournament
- Pikachu does not fight:

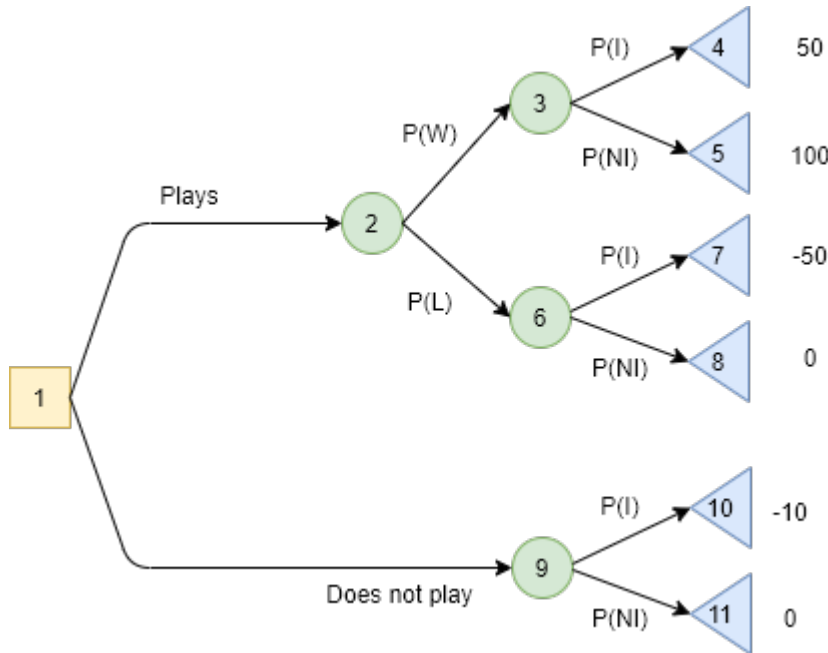
If Pikachu fights, the result of the game can be modeled as a random node with two possible outcomes:

- Win: Win the battle with probability  $P(W) = 0.1$
- Lose: Lose the battle with a probability  $P(L) = 1 - P(W) = 0.9$

Independent of the result of the battle, Pikachu might be injured from a previous battle. This can be modeled as a random event:

- Pikachu injured: Pikachu is injured with probability  $P(I) = 0.2$
- Pikachu is not injured: Pikachu is not injured with probability  $P(NI) = 0.8$

This yields the following decision tree:



b. Determine which is the best decision for Ash. We need to determine the expected value of the intermediate random nodes, applying the probabilities:

$$V_3 = 50 \cdot 0.2 + 0.8 \cdot 100 = 90$$

$$V_6 = -50 \cdot 0.2 + 0.8 \cdot 0 = -10$$

$$V_2 = 90 \cdot 0.1 - 10 \cdot 0.9 = 0$$

$$V_9 = -10 \cdot 0.2 + 0.8 \cdot 0 = -2$$

Finally, to solve the tree, the value of the decision node is calculated as the maximum value minus cost of the different alternatives:

$$V_1 = \max(V_2, V_9) = 0$$

and the decision is given by the argument of the maximum (argmax), which corresponds to alternative “Pikachu plays”.

c. Ash could acquire further information about the probabilities of the result of the battle. The additional information determines the result of the battle (Win or lose), and based on previous results, it has a certain accuracy to successfully predict the result of the battle. Draw again the decision tree to incorporate into the decision problem the acquisition of this imperfect information about the result of the battle, indicating in the corresponding branches the revised probabilities

The addition information would make Ash review the probabilities of the result of the battle.

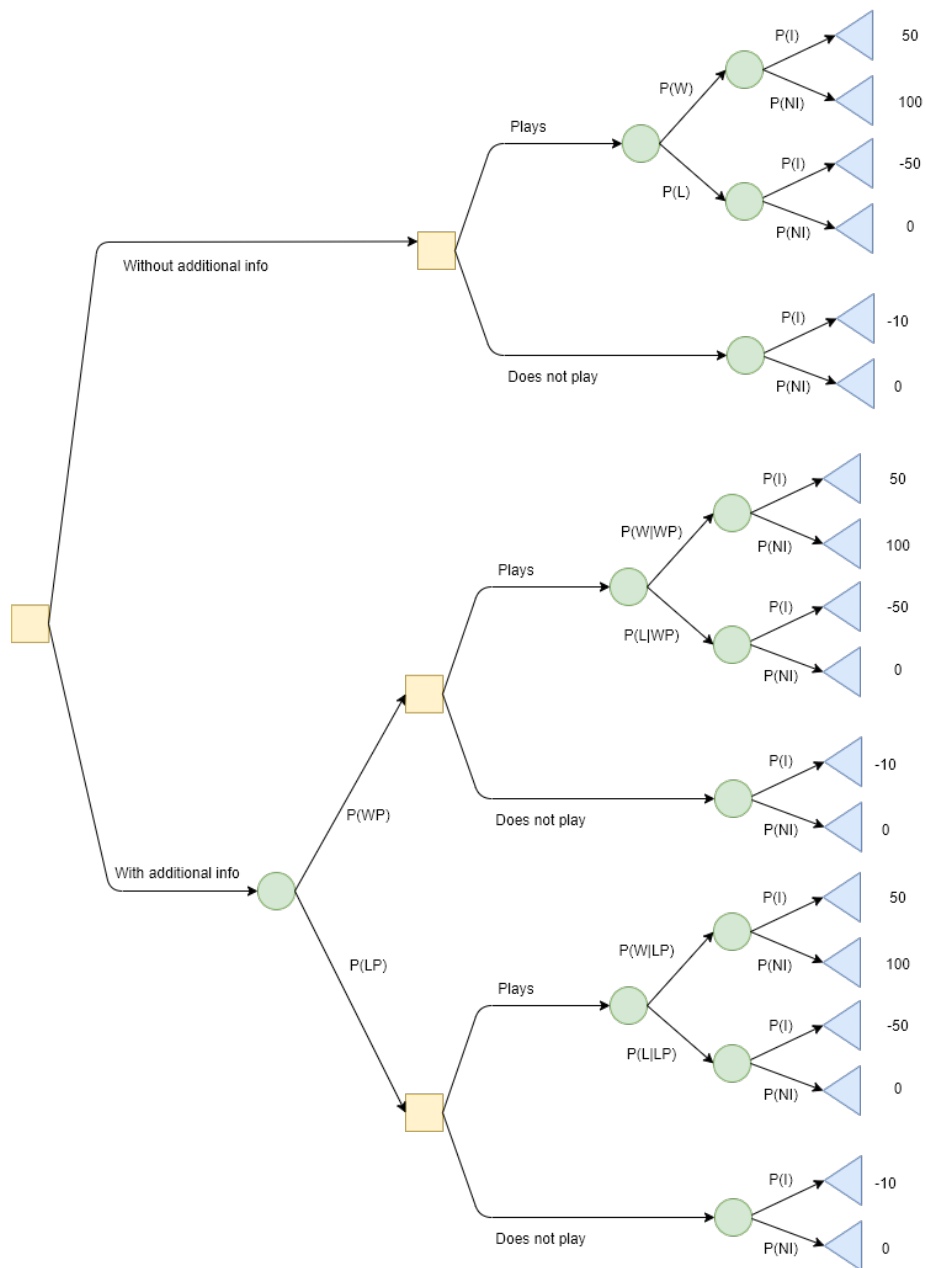
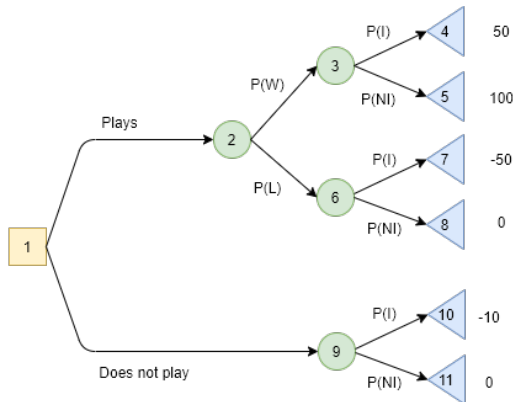
Now, we can model the acquisition of additional information as a decision node with two alternatives: - Do not acquire additional information - Acquire additional information

The result of the additional information can be modeled as a random node with possible events:

- Predict a win: The additional information may predict a win with probability P(WP)
- Predict a Loss: The additional information may predict a loss, with probability P(LP)

Finally, it is required to modify the probabilities of the result of the game to take into account the additional information.

This yields the following decision tree



### 5.3.3 Planning a factory site

#### Problem Definition

Dinoco, a multinational firm in the automobile sector, is deciding on whether to build an assembly plant in Brazil or in Mississippi (USA). The cost to build this plant in Brazil is 10 million euros and the cost of building the plant in Mississippi is 20 million. However, if the firm builds the plant in Brazil and local demand drops over the following 5 years, the project will be stopped and the firm will lose 10M€ (and it will still have to build a plant in Mississippi). A priori, Dinoco believes that the probability of demand for cars dropping in Brazil over the following 5 years is 20%. For 1M€, Dinoco can hire a market research firm to analyse demand for cars in Brazil, which will indicate whether demand for cars will drop or not. The market research firm's record indicates that his record shows that they are able to predict the occurrence of a drop in demand with a 95% success rate, and 90% when the demand does not drop.

- a. Draw the decision tree.
- b. Solve the tree and find the best alternative for Dinoco.
- c. Calculate the Imperfect Information Value and the Perfect Information Value associated to the market research (demand in Brazil)

The following diagram shows the decision tree of the problem:

The image has been generated using Python, if you want to do it yourself, you can try the following code cell in a Notebook!

```
[ ]: from graphviz import Digraph
# Create Digraph object
din = Digraph(format='png')
din.graph_attr['rankdir'] = 'LR'

# Add root node
din.node('1', shape='square', style="filled", fillcolor='yellow')

# First branch nodes
din.node('2', shape='square', style="filled", fillcolor='yellow')
din.node('3', shape='circle', style='filled', fillcolor='green')
din.node('4', xlabel='-30', shape='diamond', style='filled', fillcolor='purple')
din.node('5', xlabel='-10', shape='diamond', style='filled', fillcolor='purple')
din.node('6', xlabel='-20', shape='diamond', style='filled', fillcolor='purple')

din.edge('1', '2', 'Without research')
din.edge('2', '3', 'Build in Brazil')
din.edge('2', '6', 'Build in Missisipi')
din.edge('3', '4', 'Demand Drops')
din.edge('3', '5', 'Demand does not drop')

din.node('7', shape='circle', style='filled', fillcolor='green')
din.edge('1', '7', 'With research')

# Second branch
```

(continues on next page)

(continued from previous page)

```

din.node('8', shape='square', style="filled", fillcolor='yellow')
din.edge('7', '8', 'Predict demand drops')

# Second branch nodes
din.node('9', shape='circle', style='filled', fillcolor='green')

din.node('10', xlabel='-30', shape='diamond', style='filled', fillcolor='purple')
din.node('11', xlabel='-10', shape='diamond', style='filled', fillcolor='purple')
din.node('12', xlabel='-20', shape='diamond', style='filled', fillcolor='purple')

din.edge('8', '9', 'Build in Brazil')
din.edge('8', '12', 'Build in Missisipi')
din.edge('9', '10', 'Demand Drops')
din.edge('9', '11', 'Demand does not drop')

# Third branch
din.node('13', shape='square', style="filled", fillcolor='yellow')
din.edge('7', '13', 'Predict demand does not drop')

# Third branch nodes
din.node('14', shape='circle', style='filled', fillcolor='green')

din.node('15', xlabel='-30', shape='diamond', style='filled', fillcolor='purple')
din.node('16', xlabel='-10', shape='diamond', style='filled', fillcolor='purple')
din.node('17', xlabel='-20', shape='diamond', style='filled', fillcolor='purple')

din.edge('13', '14', 'Build in Brazil')
din.edge('13', '17', 'Build in Missisipi')
din.edge('14', '15', 'Demand Drops')
din.edge('14', '16', 'Demand does not drop')

din
    
```

The initial (a priori) probabilities are:

Probability that demand drops:  $P(D_D) = 0.2$

Probability that demand does not drop:  $P(D_{ND}) = 0.8$

The conditional probabilities with additional information are:

- Record of predicted *market drops* events:

$P(PD_D|D_D) = 0.95$

$P(PD_D|D_{ND}) = 0.05$

Record of predicted *market does not drop* events:

$$P(PD_{ND}|D_D) = 0.1$$

$$P(PD_{ND}|D_{ND}) = 0.9$$

With these, the probabilities of occurrence of imperfect information are:

$$P(PD_D) = P(D_D) \cdot P(PD_D|D_D) + P(D_{ND}) \cdot P(PD_D|D_{ND}) = 0.27$$

$$P(PD_{ND}) = P(D_D) \cdot P(PD_{ND}|D_D) + P(D_{ND}) \cdot P(PD_{ND}|D_{ND}) = 0.73$$

Now with these data, we can calculate the posterior probabilities using the Bayes Theorem:

$$P(D_D|PD_D) = \frac{P(D_D) \cdot P(PD_D|D_D)}{P(PD_D)} = 0.7037$$

$$P(D_{ND}|PD_D) = \frac{P(D_{ND}) \cdot P(PD_D|D_{ND})}{P(PD_D)} = 0.2963$$

$$P(D_D|PD_{ND}) = \frac{P(D_D) \cdot P(PD_{ND}|D_D)}{P(PD_{ND})} = 0.0137$$

$$P(D_{ND}|PD_{ND}) = \frac{P(D_{ND}) \cdot P(PD_{ND}|D_{ND})}{P(PD_{ND})} = 0.9863$$

Now, entering this information in the graph, we can calculate the values of the different nodes:

$$V_3 = P(D_D) \cdot V_4 + P(D_{ND}) \cdot V_5 = 0.2 \cdot (-30) + 0.8 \cdot (-10) = -14 \quad V_2 = \max(V_3, V_6) = -14$$

$$V_9 = P(D_D|PD_D) \cdot V_{10} + P(D_{ND}|PD_D) \cdot V_{11} = 0.7037 \cdot (-30) + 0.2963 \cdot (-10) = -24.074 \quad V_8 = \max(V_9, V_{12}) = -20$$

$$V_{14} = P(D_D|PD_{ND}) \cdot V_{15} + P(D_{ND}|PD_{ND}) \cdot V_{16} = 0.0137 \cdot (-30) + 0.9863 \cdot (-10) = -10.274 \quad V_{13} = \max(V_{14}, V_{17}) = -10.274$$

$$V_7 = P(PD_D) \cdot V_9 + P(PD_{ND}) \cdot V_{13} = -12.9$$

$$V_1 = \max(V_2, V_7 - 1) = -13.9$$

According to the decision tree, it is recommended to hire the consultant. The expected cost is -13.9M€. If the report predicts a drop in the demand, the best alternative is to build the plant in Mississippi and if the report does not predict a drop, in Brazil.

The Expected Return with Perfect Information is:

$$ERPI = 0.2 \cdot (-20) + 0.8 \cdot (-10) = -12$$

And the Perfect Information Value =

$$PIV = ERPI - (-13.9) = -12 - (-13.9) = 1.9$$

### 5.3.4 Selecting Suppliers

#### Problem Definition

Your company, Altered Silicon, manufactures solar energy panels, and it has received an urgent order for 40 panels from a new German customer which is willing to pay 1,000€ per panel. Altered Silicon has committed its production capacity to other orders, and the only alternative left to serve the new order is to subcontract the production involved.

Altered Silicon has located two possible suppliers: - Supplier A: based in Romania, placed an offer of 500€ per panel - Supplier B: from Poland, which would sell each panel for 550€

The cost to transport one batch of 40 panels from either of the two suppliers to the German firm is 1,000€. Nonetheless, the customer's technical specifications are very demanding, so Altered Silicon estimates that Supplier A could supply one correct batch with a probability of 70%, whereas Supplier B could do it with a probability of 80%. If the batch were incorrect, Altered Silicon could sell it to another customer at the price of 200€ per panel. Altered Silicon is considering the possibility of asking for a test panel from each supplier to test it before placing an order. Altered Silicon's testing system guarantees that a faulty panel is always detected, while there is a 5% margin of error with correct panels. The

testing system is prepared to assess two test panels, and the overall cost (cost of the two panels + transport + cost of the test) comes to 2,000€.

**a** Draw the corresponding decision tree and indicate with a letter per branch the associated probability or the associated cost (depending on the case)

The company needs to decide whether to conform the test panel or not. So the alternatives for the first decision node are:

- Not conform test panel: Cost 0
- Conform test panel: Cost  $CP = 2K€$

The test panel will provide additional information on the possibility that the batches provided by each provider are bad. The outcome of the test panel can be modeled as a random node with 4 possible outcomes:

- The test on the sample from supplier A is correct (PAC), with Probability  $P(PAC)$  and the test on the sample from supplier B is correct (PBC), with probability  $P(PBC)$ :  $P(PAC) \cdot P(PBC)$
- The test on the sample from supplier A is correct (PAC), with Probability  $P(PAC)$  and the test on the sample from supplier B is incorrect (PBI), with probability  $P(PBI)$ :  $P(PAC) \cdot P(PBI)$
- The test on the sample from supplier A is incorrect (PAI), with Probability  $P(PAI)$  and the test on the sample from supplier B is correct (PBC), with probability  $P(PBC)$ :  $P(PAI) \cdot P(PBC)$
- The test on the sample from supplier A is incorrect (PAI), with Probability  $P(PAI)$  and the test on the sample from supplier B is incorrect (PBI), with probability  $P(PBI)$ :  $P(PAI) \cdot P(PBI)$

In the probabilities above, we assumed that the results on the samples from different suppliers are independent events.

Let us assume that the 3 alternatives for the company are:

- Select supplier A (Supplier A):  $CA = -(0.5 \cdot 40) - 1 = -21$
- Select supplier B (Supplier B):  $CB = -(0.55 \cdot 40) - 1 = -23$
- Do not purchase from any supplier (No supplier):  $CN = 0$

Now, if we select provider A, the batch may be correct or incorrect. This event can be modeled as a random node with the following alternatives and corresponding initial probabilities:

- Batch is correct (AC), with probability  $P(AC) = 0.7$
- Batch is incorrect (AI), with probability  $P(AI) = 0.3$

Similarly, for player B, the batch may also be correct or incorrect, with (a priori) probabilities:

- Batch is correct (BC), with probability  $P(BC) = 0.8$
- Batch is incorrect (BI), with probability  $P(BI) = 0.2$

Now, the problem also gives some conditional probabilities on further information, related to the reliability of the test. From historic results, the probability that a test gave an incorrect result given that the batch was correct is 0.05. Hence:

- $P(PAC|AC) = 0.95$
- $P(PAI|AC) = 0.05$
- $P(PBC|BC) = 0.95$
- $P(PBI|BC) = 0.05$



Likewise, in the past, the test has never given an incorrect result and therefore:

- $P(\text{PAC}|\text{AI}) = 0$
- $P(\text{PAI}|\text{AI}) = 1$
- $P(\text{PBC}|\text{BI}) = 0$
- $P(\text{PBI}|\text{BI}) = 1$

By applying the Bayes rule, the probabilities of occurrence of the test results are:

$$P(\text{PAC}) = P(\text{PAC}|\text{AC})P(\text{AC}) + P(\text{PAC}|\text{AI})P(\text{AI}) = 0.665 \quad P(\text{PAI}) = 1 - P(\text{PAC}) = 0.335$$

$$P(\text{PBC}) = P(\text{PBC}|\text{BC})P(\text{BC}) + P(\text{PBC}|\text{BI})P(\text{BI}) = 0.76 \quad P(\text{PAI}) = 1 - P(\text{PBC}) = 0.24$$

Which yields:

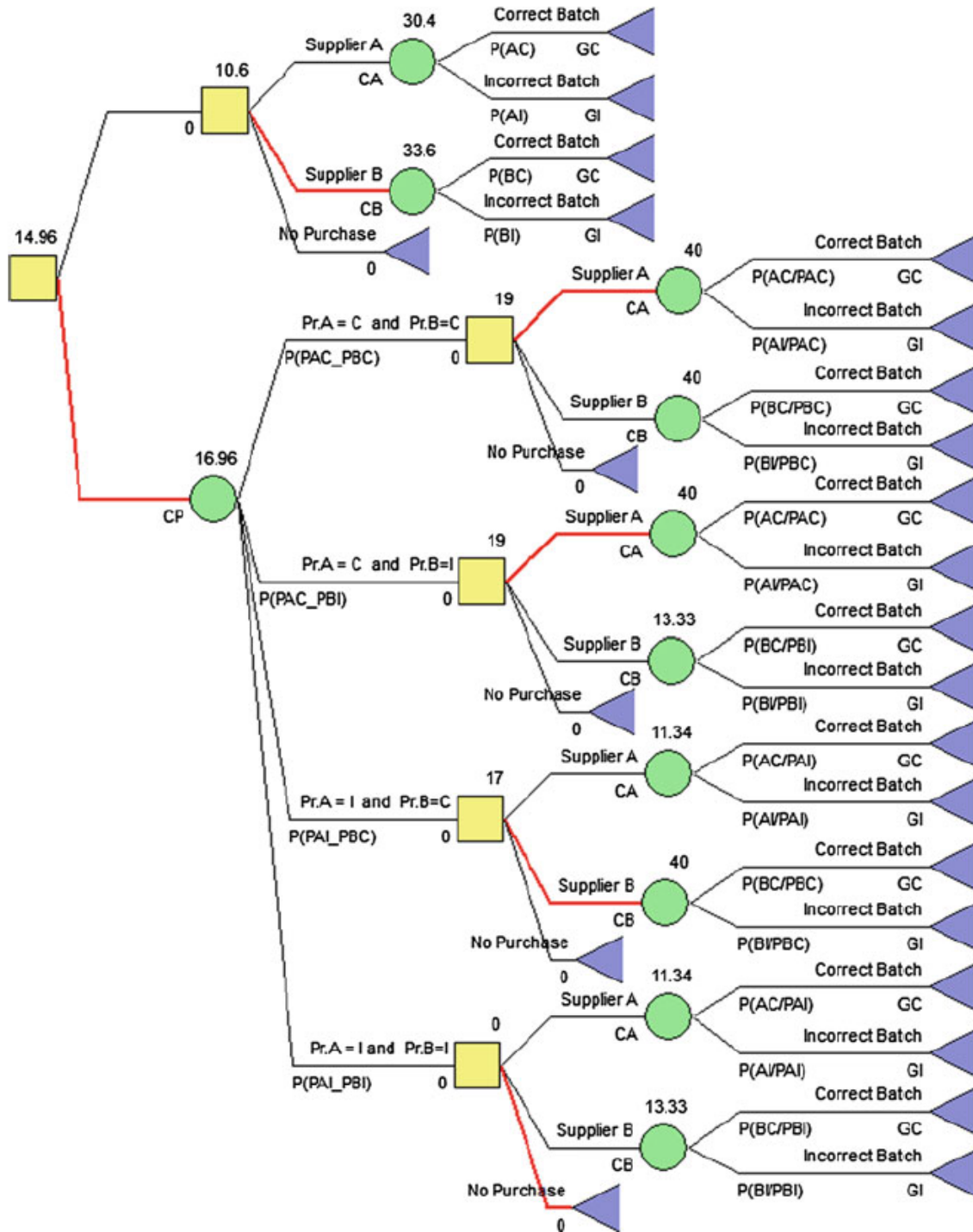
$$P(\text{PAC})P(\text{PBC}) = 0.5054 \quad P(\text{PAC})P(\text{PBI}) = 0.1596 \quad P(\text{PAI})P(\text{PBC}) = 0.254 \quad P(\text{PAI})P(\text{PBI}) = 0.0804$$

And applying Bayes yields

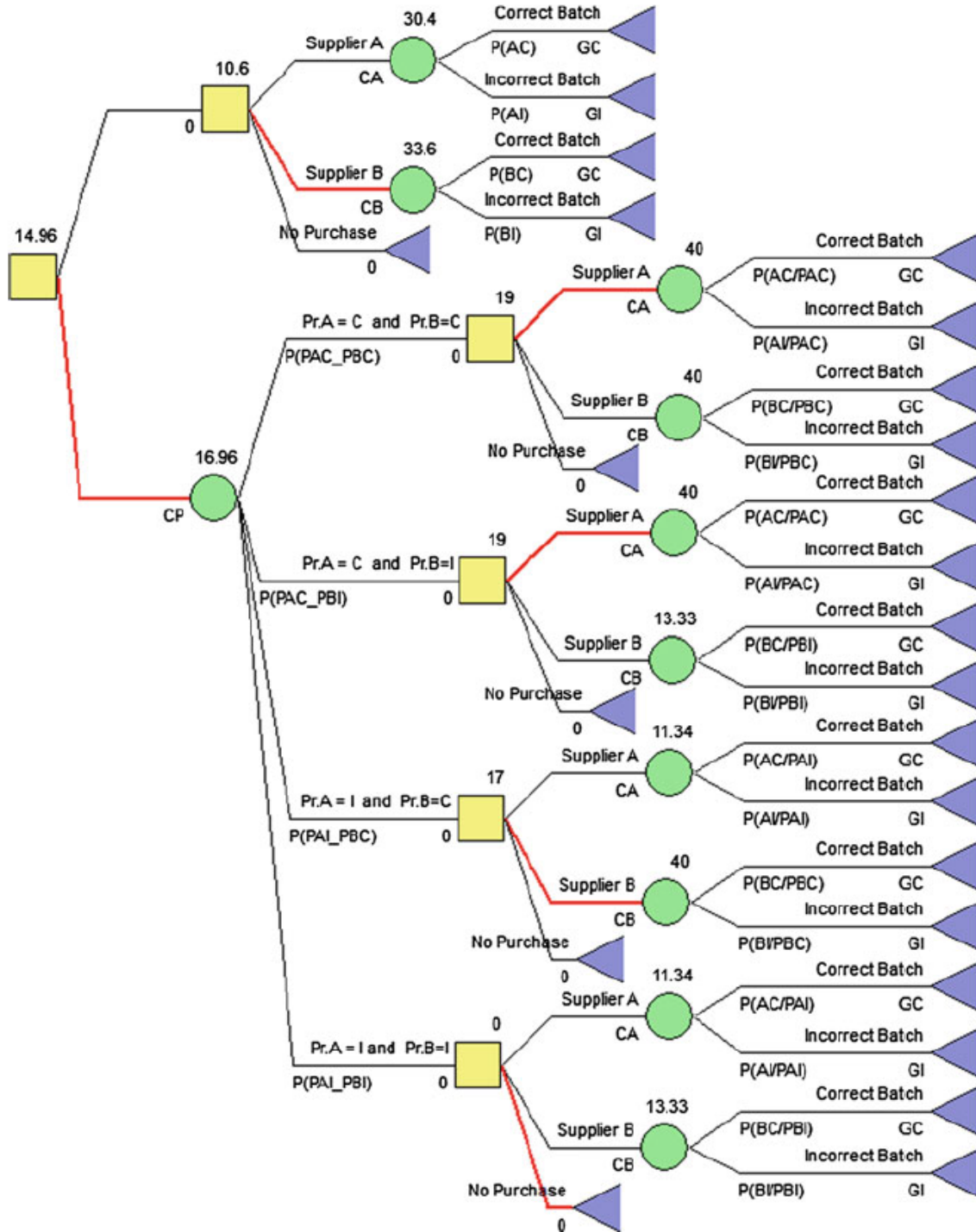
$$P(\text{AC}|\text{PAC}) = 1 \quad P(\text{AI}|\text{PAC}) = 0 \quad P(\text{AC}|\text{PAI}) = 0.1045 \quad P(\text{AI}|\text{PAI}) = 0.8955$$

$$P(\text{BC}|\text{PBC}) = 1 \quad P(\text{BI}|\text{PBC}) = 0 \quad P(\text{BC}|\text{PBI}) = 0.1667 \quad P(\text{BI}|\text{PBI}) = 0.8333$$

Finally, we have the following outcomes in the different leave nodes: - When the batch is correct  $\text{GC}=40\text{K€}$  - When the batch is incorrect  $\text{GI}=8\text{K€}$  - No purchase = 0



**b** Solve the decision tree To solve, the decision tree, we need to calculate the expected value of random nodes, and select the alternative that provides the maximum benefit (outcome minus cost) in decision nodes. The following figure shows the solution of obtained in the decision tree:



c Up to how much should SUN2 pay for the test?

The company could pay p to the difference between the values of the two alternatives, without or with test:

$$16.96 - 10.60 = 6.36$$

### 5.3.5 Blind wine tasting

#### Problem Definition

María and Alberto (names have been selected randomly, any resemblance with reality is pure coincidence) are two classmates that want to celebrate the end of the course with a game. They will go to their favourite bar and Alberto will randomly select a bottle of wine from the wine list. María will do a blind testing of the wine, with no possibility to see the wine selected by Alberto. If María guesses the type of wine (red or white) just by holding her glass, Alberto will pay 20€. Otherwise, if she fails, María will pay Alberto 10€. She may also taste the wine for 5€ (that is, Alberto will discount 5€ of the bet if she guesses the type of wine, or María will pay 5€ extra if she does not).

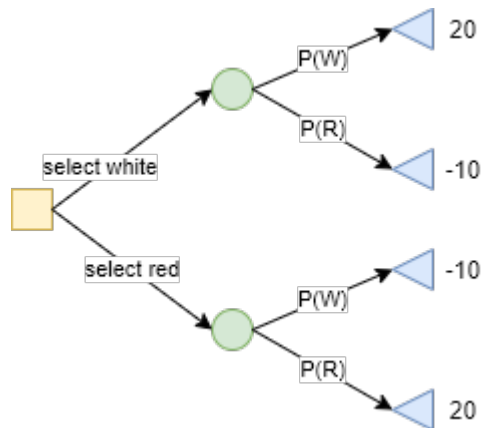
María thinks she can use decision theory to improve her chances of winning the bet. She knows that in the wine list of their favourite bar, there are 15 bottles of red wine and 11 of white wine. She has also done some blind wine tasting to prepare for the bet and found out that it is actually not that easy to guess the colour of the wine just by tasting it blindfolded. She guessed correctly 3/4 of the times she tried a white wine and 2/4 of the times she tried a red wine.

**a** Draw the decision tree for María without taking into account that she may test the wine to get additional information.

In this option, we only consider the possibility that María guesses the type of wine just by holding the glass of wine. In this scenario, there is only one decision node, with two alternatives, either white wine or red wine. For every alternative, we can model the selection of Alberto as a random event, without any María's control. Without further information, let us consider that all wines have the same probability of being selected by Alberto. Under this assumption, each random node has two possible states:

- White wine: Alberto selects a white wine with probability  $P(W) = 11/26$
- Red wine: Alberto selects a red wine with probability  $P(R) = 15/26$

Now, since María gets 20€ when she correctly guesses the type of wine and -10 otherwise, the decision tree is:

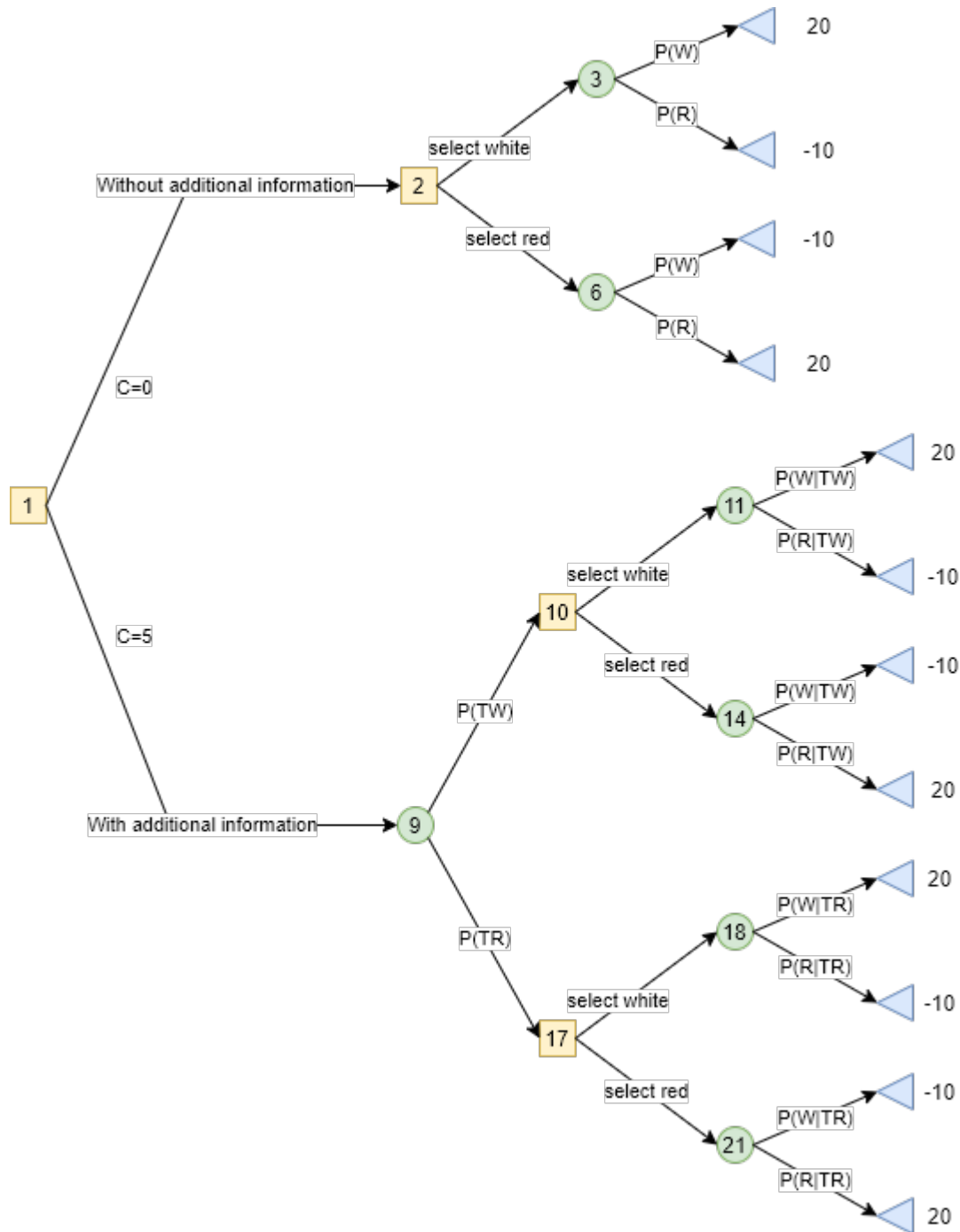


**b** Modify the decision tree for María now taking into account that she may test the wine. Now, to modify the decision tree taking into account that she may test the wine, we need to introduce a new decision node to model the decision of tasting the wine or not. Given that we have the record of attempts of María, we can assume that tasting the wine will provide to María additional imperfect information about the type of wine, and model the result of the tasting through a random node with two different alternatives:

- María tastes the wine, and she thinks it tastes like white wine:  $P(TW)$
- María tastes the wine, and she thinks it tastes like red wine:  $P(TR)$

Note that tasting the wine and choosing the wine are not the same think. María may test the wine and then trust her taste or not, taking into account the reliability of previous tests, for which we have information.

With this, the decision tree becomes:



We can calculate the posterior probabilities from the record of tests conducted by María, applying Bayes. From her tests we know that:

- Probability that María tasted a white wine given that the wine was actually white  $P(TW|W) : 3/4$
- Probability that María tasted a white wine given that the wine was actually red  $P(TW|R) : 2/4$
- Probability that María tasted a red wine given that the wine was actually red  $P(TR|R) : 2/4$
- Probability that María tasted a red wine given that the wine was actually white  $P(TR|W) : 1/4$

Now, using the law of total probability we can calculate the probabilities of tasting a white wine and a red wine:

$$P(TW) = P(TW|W) * P(W) + P(TW|R) * P(R) = 3/4 * 11/26 + 2/4 * 15/26 = 0.605$$

$$P(TR) = P(TR|R) * P(R) + P(TR|W) * P(W) = 2/4 * 15/26 + 1/4 * 11/26 = 0.394$$

And from these results, we can calculate the posterior probabilities using Bayes:

$$P(W|TW) = P(TW|W) * P(W) / P(TW) = 3/4 * 11/26 / (0.605) = 0.52$$

$$P(R|TW) = P(TW|R) * P(R) / P(TW) = 2/4 * 15/26 / (0.605) = 0.48$$

$$P(W|TR) = P(TR|W) * P(W) / P(TR) = 1/4 * 11/26 / (0.394) = 0.27$$

$$P(R|TR) = P(TR|R) * P(R) / P(TR) = 2/4 * 15/26 / (0.394) = 0.73$$

Note that the information of the tests is not very reliable, specially when María tests the red wine, this is an important fact to consider when we calculate the tree.

**c** What is the best alternative for María?

Now, to solve the tree, we need to calculate the values of the nodes:

$$V_3 = 20 * 11/26 - 10 * 15/26 = 70/26 \quad V_6 = -10 * 11/26 + 20 * 15/26 = 190/26$$

$$V_2 = \max(V_3, V_6) = 190/26 = 7.3$$

$$V_{11} = 20 * 0.52 - 10 * 0.48 = 5.6$$

$$V_{14} = -10 * 0.52 + 20 * 0.48 = 4.4$$

$$V_{10} = \max(V_{11}, V_{14}) = 5.6$$

$$V_{18} = 20 * 0.27 - 10 * 0.73 = -1.9$$

$$V_{21} = -10 * 0.27 + 20 * 0.73 = 11.9$$

$$V_{17} = \max(V_{18}, V_{21}) = 11.9$$

$$V_9 = 0.605 * 5.6 + 0.394 * 11.9 = 8.08$$

$$V_1 = \max(7.3, 8.08 - 5) = 7.3$$

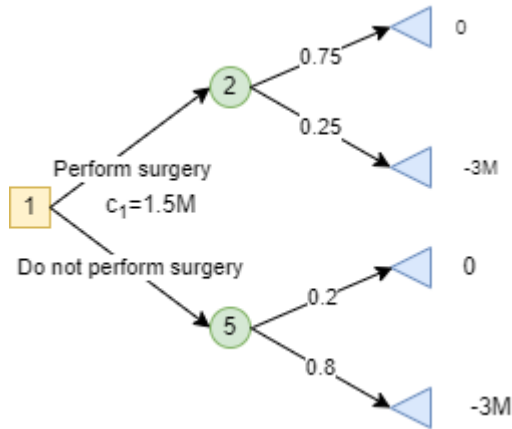
According to the tree, María should try to guess the type of wine just by holding the glass and just guess that is a red wine. This is also a logical conclusion taking into account that the tests she conducted showed that tasting the wine was not a very reliable source of information.

### 5.3.6 The eye of Agamoto

#### Problem Definition

Doctor Stephen Strange is the chief surgeon at the Metropolitan General Hospital in New York. Doctor Strange needs to decide whether to perform surgery or not on a patient with a rare disease. If the patient dies, the hospital must pay a 3M€ compensation, and the cost of the surgery is estimated in 1.5M€. Based on available data, the medical department estimates that the probability of the patient surviving without surgery is 20%, while the probability of the patient surviving with surgery is 75%. Let us use operational research to determine if Doctor Strange should hold the knife:

**a.** Draw the decision tree to support doctor Strange with this decision problem The following image represents the decision tree for Doctor Strange:



The first node represents the decision of whether to perform surgery (at a cost of 1.5M€) or not. In both branches, the random nodes represent the event of the patient surviving the procedure. Note that the probabilities of surviving are different in both branches, because they are conditioned on whether the operation is performed or not.

Also note that the value if the patient survives is 0 while the value if the patient passes away is -3M€ in both cases.

**b.** Solve the decision tree Let us solve the tree from right to left. If Strange performs the operation, the value of the random node (in M€) is:

$$V_2 = 0.75 * 0 + 0.25 * -3 = -3/4$$

And, if he does not, the value of the random node is:

$$V_5 = 0.2 * 0 + 0.8 * -3 = -4 * 3/5$$

Now, the value of the decision node is going to be the maximum taking into account the cost:

$$V_1 = \max(V_2 - 1.5, v_5) = \max(-3/4 - 3/2, -4 * 3/5) = \max(-9/4, -12/5) = -9/4$$

So, the best alternative for Dr. Strange is to perform the operation.

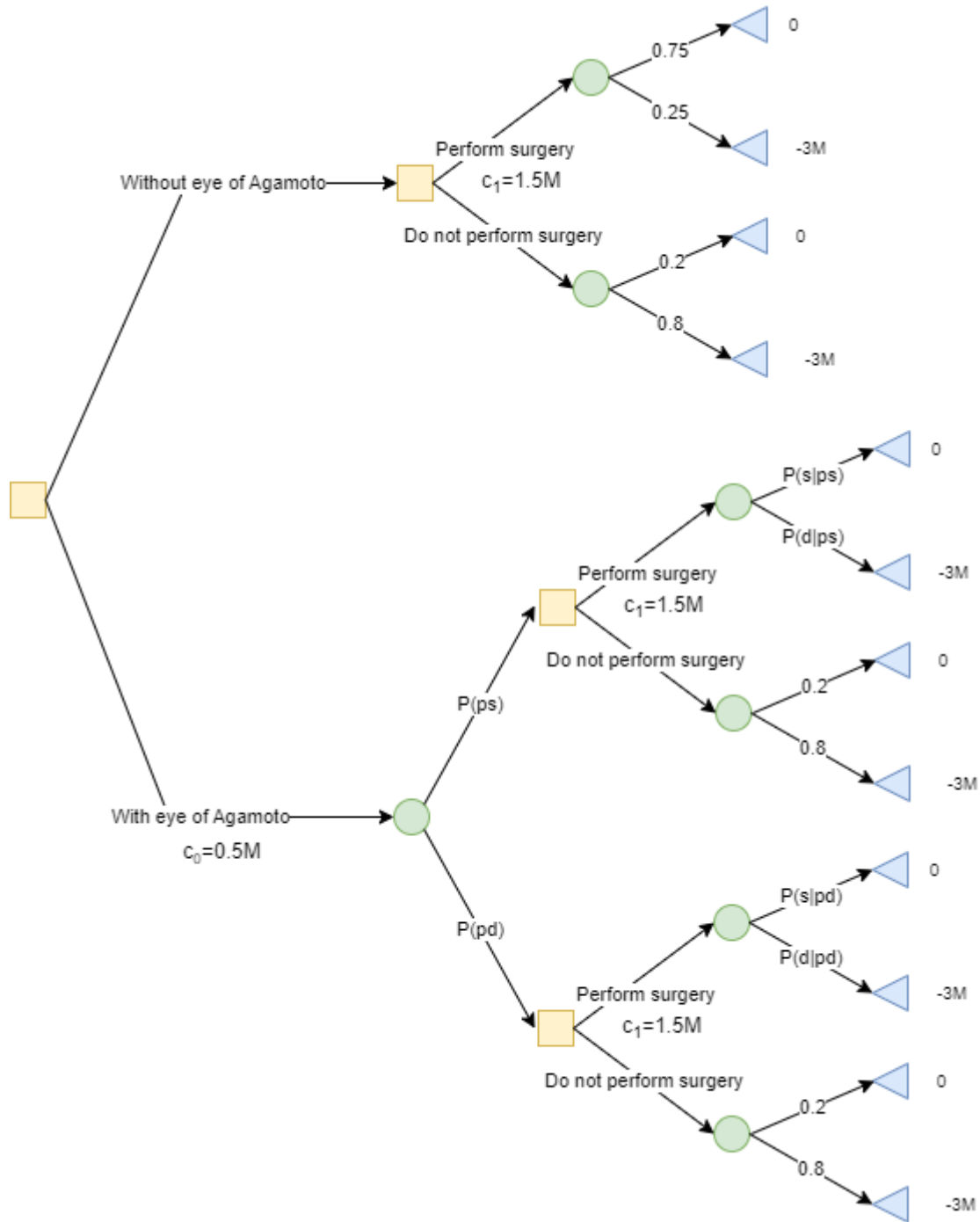
**c.** Doctor Strange can use a procedure with a device known as the Eye of Agamotto to get additional information of the result of the surgery, with a cost of 0.5M€. Draw the decision tree incorporating the use of the Eye of Agamotto in the decision problem Now, the following diagram represents the decision with additional information provided by the Eye of Agamotto:

If Dr. Strange decides not to use the Eye of Agamotto, the decision tree looks exactly the same as in the previous section, because the decision process will be equivalent.

If Dr. Strange decides to use the Eye of Agamotto, we can model the result of the prediction provided by this tool as a random node with two possible states:

- Survival predicted (sp): The eye of Agamotto will predict that the patient survives the operation with probability  $P(sp)$
- Death predicted (sp): The eye of Agamotto will predict that the patient does not survive the operation with probability  $P(sp)$

Since the Eye of Agamotto provides additional information about the result of the procedure, we need to plug in the revised probabilities in the branches where Dr. Strange decides to perform the procedure, but the probabilities without surgery remain the same.





## 5.4 Game Theory Exercises

In this section you have a collection of decision tree problems sorted by difficulty:

- **Easy problems:** You can tackle these problems after the first lesson on game theory.
- **Normal problems:** These problems require that you are familiar with concepts like game equilibrium.
- **Hard problems:** This is the type of problems that definitively get you ready for the exam.

### 5.4.1 Beer prices in Paradise Beach

#### Problem definition

There are two bars in the paradise beach resort, Awesome Sloth and Brave Flamingo. Each one of the two bars charges its own price for a beer, either 2€, 4€, or 5€. The cost of obtaining and serving the beer can be neglected. It is expected that 60% of the tourists are relaxed and casual and choose the bars randomly regardless of the price, while 40% are more thoughtful and will go to the bar with the lowest price, and split evenly in case both bars offer the same price.

a) Consider a sample of 100 guests and represent the beer price decision-making as a game with two players where the objective is to maximise revenues

b) Represent the beer price decision-making as a zero-sum game where the objective is to maximise the percentage of revenues per beer sales in the resort

### 5.4.2 Products Improvement Strategy Between Competing Firms

#### Problem definition

Two manufacturers, AdeFab1 and AdeFab2, compete for the sales of two different product lines which are equally profitable. In both cases, AdeFab2's sales volume triples that of AdeFab1. Due to some technological advances, both manufacturers can make substantial improvements to the two products, but they are not sure as to which development and commercialisation strategy they should adopt. If the improvements to the two products are done at the same time, neither manufacturer can have them ready to sell under 12 months. One alternative is to carry out an "intensive programme" to first develop one of the two products and to try to commercialise it before the competition does. If this were the case, AdeFab2 can have one product ready to sell in 9 months, while AdeFab1 needs 10 months (due to previous commitments in its production installations). Either manufacturer can have the second product ready after another 9-month period. For either product line, if both manufacturers commercialised the improved models simultaneously, it is estimated that AdeFab1 can increase the percentage of all the future sales of this product by 8 % of the total (from 25 % to 33 %). Likewise, AdeFab1 can increase its total sales in 20, 30 and 40 % of the total if it commercialises the product 2, 6 and 8 months before AdeFab2 does, respectively. Moreover, AdeFab1 can lose 4, 10, 12 and 14 % of the total if AdeFab2 manages to commercialise it 1, 3, 7 and 10 month(s) before it, respectively. Formulate this problem as if it were a zero-sum game with two players and then establish the strategy the manufacturers must adopt according to the MiniMax criterion.

### 5.4.3 Setting oil prices in the 70s

#### Problem definition

In the 1970s, the OPEP countries adopted a price-linking strategy and oil prices went from €3/barrel to a level above €30/barrel. Analysts foresaw that prices could rise to over €100/barrel by the end of the twentieth century. Unexpectedly however, the cartel failed and prices went down again. Therefore, what are the conditions for these cartels to survive? In other words, what determines the equilibrium between cooperation and competition? This problem is similar to the prisoner's dilemma.

To simplify matters, it is assumed that only two producers exist, Saudi Arabia and Iran, and that there are only three possible production levels: 2, 4 and 6 million barrels/day. So, depending on the two players' decisions, total world production is 4, 6, 8, 10 and 12 million barrels/day. Accordingly, world prices are 35, 30, 25, 15 or only 10€/barrel. The production costs of one barrel are considered to be €2 for Saudi Arabia and €4 for Iran.

- a) The intention is to maximise the difference between the profit of a producer and that of its competitor. Consider and solve the case as if it were a zero-sum game with two players.
- b) If the idea is to seek OPEP's prime objective by avoiding mutual competition and regulating oil production, sales and prices, what is the best option for Saudi Arabia and for Iran if they cooperate with each other and try to simultaneously obtain the best profit each?

### 5.4.4 Developing a High-Tech Product

#### Problem definition

BIOSENS and TECHSENS, two competing firms, are developing a new state-of-the-art biomedical sensor, whose total sales will amount to 1,000 million dollars. Both must opt for using the most advanced "e1" or "e2" technology. Should both firms decide on the same technology, BIOSENS would obtain 60% of sales, since they know how to sell their products better than their competitor, which sells similar products. However, if both opt for different technologies, then TECHSENS would obtain a 60 % market share because it exploits differentiation better.

- a) Determine which strategy each firm should adopt

### 5.4.5 Investments Strategy

#### Problem Definition

An investor wishes to invest €12,000 over a 1-year period, although he/she must decide to invest in market shares and/or gold. The profitability ratios estimated for each investment are based on the possible states of economy (see Table).

Investment	Growth	Mean Growth	Stabilisation	Poor Growth
Market shares	5	4	3	-1
Gold	2	3	4	5

Consider a zero-sum Games Theory model with two players and answer the following:

- a) How should the investor invest the €12,000 to carry out the optimal strategy?
- b) What mean profitability value is obtained? What is the profit for the investor?

### 5.4.6 Playing Bluff

#### Problem definition

Two students of EDEM, Jaime and Lucas, are playing a game named Bluff, that goes as follows. Jaime writes a number on a piece of paper. Without showing Lucas what he has written, he tells him what he wrote. Jaime might be lying or telling the truth. Then Lucas must guess if Jaime is lying or telling the truth, or he can choose to pass and not to continue playing. If Lucas discovers that Jaime lied, Jaime must pay Lucas €50. If Lucas wrongly accuses Jaime of lying, Lucas pays him €25. If Jaime told the truth and Lucas guesses this, then Jaime pays Lucas €5. If Jaime lied and Lucas does not guess that he lied, then Lucas pays Jaime €24. Should Lucas decide to pass, then neither player wins or loses anything.

**a** Consider this a 2-player's zero-sum game in game theory and write down the tabular representation.

**b** How would you determine the game mean value and the mixed strategies of both players in this case?

**c** Using the method seen in class to calculate the mixed strategies, a solver provides the following solution for Lucas ( $u_1=0$ ,  $u_2=0$ ,  $u_3=0.02$ ), where  $u_1$  is the decision variable associated to the alternative "Jaime lied",  $u_2$  the decision variable associated to the alternative "Jaime told the truth" and  $u_3$  the decision variable associated to "Pass". Write down the model and determine his optimal strategy.

### 5.4.7 The Squid Game

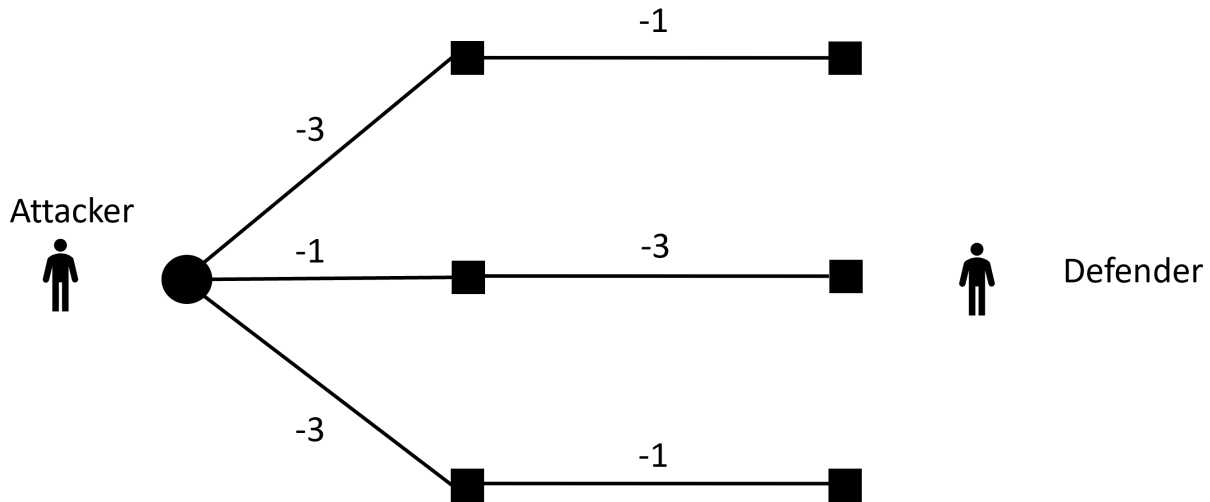
#### Problem definition

You were out celebrating the end of the exams at EDEM and the next thing you know is you are trapped in an island playing for your life to a game known as the Squid Game. There are two players in the Squid Game, the attacker and the defender. The attacker must pass through the field, choosing one out of three possible paths, up, middle, and down. The defender must guess which path the attacker has selected, and follow the same path to meet him in the middle of the field.

Both players have a limited endurance, measured in endurance points, and start with 10 endurance points. The winner is the player that resists the longest time without losing all endurance points.

The field is fraught with traps that cause injuries to the players. If the attacker crosses the 'up' or 'down' paths, the damage caused to the attacker is assessed as -3 endurance points. If the attacker chooses the 'middle' path, the damage is assessed as -1 endurance points. If the defender chooses the 'up' or 'down' paths, the damage is -1 endurance points. The damage for the defender is -3 endurance points if he chooses the middle path.

Both players are blindfolded and choose the path they would like to follow at the same time. If the defender chooses the same path as the attacker, the damage for the attacker is -5 endurance points. If on the other hand the defender chooses a different path, the damage for the defender is -2 endurance points. The following image shows the Squid Game field and the damage in storage points in each path.



- Write down the tabular representation of the game.
- Find the move each player would choose following the minmax criteria. Discuss briefly if there is equilibrium.
- Now think of the game as a zero-sum game where the gain in each move is the difference in damage points between the attacker and the defender for each combination of pairs of paths selected. Write down the linear programming problems to calculate the optimal probabilities of the mixed strategies for both players.

## 5.4.8 Strategies for political campaigns

### Problem Definition

Suppose there are two political parties, The Avengers Party (a coalition of superheroes who have come together to fight for the greater good. They believe in using their powers to protect the people and promote equality and justice) and the Brotherhood party (This party represents a group of mutants who feel oppressed by society and seek to overthrow the existing power structures). They are competing in an election. Each party has three possible strategies: they can focus their campaign on young citizens (below 30), middle-aged citizens (30-50), or senior citizens (over 50). The parties use national statistics to estimate the number of votes that can shift in each demographic group. Here are the estimated values for each demographic group:

- Young voters: They represent 60% of the undecided voters.
- Middle-aged voters: They represent 25% of the undecided voters.
- Senior citizens: They represent 15% of the undecided voters.

From past elections, the Avengers party has collected the following estimation about undecided voters on each scenario:

- **Young voters:** If both parties focus on young voters, the Avengers party will get 40% of the young voters and the Brotherhood party 60% of the young voters. If the Avengers party focuses on the young voters and the Brotherhood party does not, the Avengers party will get 70% of the young voters. If the Brotherhood party focuses on the young voters and the Avengers party does not, the Avengers party will get 10% of the young voters. If neither party focuses on the young voters, the Avengers party will get 40% of the young voters.
- **Middle-aged voters:** If both parties focus on middle-aged voters, the Avengers party will get 60% of the middle-aged voters and the Brotherhood party 40% of the middle-aged voters. If the Avengers party focuses on the middle-aged voters and the Brotherhood party does not, the Avengers party will get 90% of the middle-aged voters. If the Brotherhood party focuses on the middle-aged voters and the Avengers party does not, the Avengers party will get 30% of the middle-aged voters. If neither party focuses on the middle-aged voters, the Avengers party will get 60% of the middle-aged voters.

- **Senior citizens:** If both parties focus on senior citizens, the Avengers party will get 70% of the senior citizens and the Brotherhood party 30% of the senior citizens. If the Avengers party focuses on the senior citizens and the Brotherhood party does not, the Avengers party will get 90% of the senior citizens. If the Brotherhood party focuses on the senior citizens and the Avengers party does not, the Avengers party will get 20% of the senior citizens. If neither party focuses on the senior citizens, the Avengers party will get 70% of the senior citizens.

Let us use Game Theory to model the strategies of both political parties, following these steps:

1. Identify the alternatives for both players and draw a tabular representation of the game were Player A is the Avengers Party
2. Complete the tabular representation of the game, estimating the total percentage of undecided voters that would vote for each party
3. Calculate the strategy of each party using the MinMax criteria

## 5.5 Game Theory Solved Exercises

In this section you have the solution to the different exercises.

### 5.5.1 Setting beer prices

#### Problem definition

There are two bars in the paradise beach resort, Awesome Sloth and Brave Flamingo. Each one of the two bars charges its own price for a beer, either 2€, 4€, or 5€. The cost of obtaining and serving the beer can be neglected. It is expected that 60% of the tourists are relaxed and casual and choose the bars randomly regardless of the price, while 40% are more thoughtful and will go to the bar with the lowest price, and split evenly in case both bars offer the same price.

a) Consider a sample of 100 guests and represent the beer price decision-making as a game with two players where the objective is to maximise revenues

The pay-off matrix in this case is:

BF AS	2€	4€	5€	min:
2€	100/100	140/120	140/150	100
4€	120/140	200/200	280/150	120
5€	150/140	150/280	250/250	150
min:	100	120	150	

Both bars would select the highest price.

b) Represent the beer price decision-making as a zero-sum game where the objective is to maximise the percentage of revenues per beer sales that each bar gets from the total sales in the resort and find the optimal prices in this case.

The value for each bar can be represented in percentages as:

AS/BF	2€	4€	5€
2€	50/50	53.85/46.15	48.27/51.72
4€	46.15/53.85	50/50	65.11/34.88
5€	51.72/48.27	34.88/65.11	50/50

We need only to take into consideration the values of player 1 to represent it as a zero-sum game:

BF/AS	2€	4€	5€
2€	50	53.85	48.27
4€	46.15	50	65.11
5€	51.72	34.88	50

Now, if we calculate the minimum of each column and the maximum of each row:

BF/AS	2€	4€	5€	min:
2€	50	53.85	48.27	48.27
4€	46.15	50	65.11	46.25
5€	51.72	34.88	50	34.88
max:	51.72	53.85	65.11	

With the MinMax criteria, the bar Awesome Sloth will select the maximum minimum loss, that is the maximum between the minimum of each row and set a price of 2€. The bar Brave Flamingo will select the minimum maximum win, that is, the minimum across the maximum values of each column, also setting a price of 2€.

[ ]:

## 5.5.2 Products Improvement Strategy Between Competing Firms

### Problem definition

Two manufacturers, AdeFab1 and AdeFab2, compete for the sales of two different product lines which are equally profitable. In both cases, AdeFab2's sales volume triples that of AdeFab1. Due to some technological advances, both manufacturers can make substantial improvements to the two products, but they are not sure as to which development and commercialisation strategy they should adopt. If the improvements to the two products are done at the same time, neither manufacturer can have them ready to sell under 12 months. One alternative is to carry out an "intensive programme" to first develop one of the two products and to try to commercialise it before the competition does. If this were the case, AdeFab2 can have one product ready to sell in 9 months, while AdeFab1 needs 10 months (due to previous commitments in its production installations). Either manufacturer can have the second product ready after another 9-month period. For either product line, if both manufacturers commercialised the improved models simultaneously, it is estimated that AdeFab1 can increase the percentage of all the future sales of this product by 8 % of the total (from 25 % to 33 %). Likewise, AdeFab1 can increase its total sales in 20, 30 and 40 % of the total if it commercialises the product 2, 6 and 8 months before AdeFab2 does, respectively. Moreover, AdeFab1 can lose 4, 10, 12 and 14 % of the total if AdeFab2 manages to commercialise it 1, 3, 7 and 10 month(s) before it, respectively. Formulate this problem as if it were a zero-sum game with two players and then establish the strategy the manufacturers must adopt according to the MiniMax criterion.

### Solution

The table below contains the pay-off matrix for both companies, including the calculations needed to calculate the costs and the MinMax criteria for each company.

AdeFab2 AdeFab1	Simultaneous Improvement Prod 1: 12 months Prod 2: 12 months	Intensive Programme A Prod 1: 9 months Prod 2: 18 months	Intensive Programme B Prod 1: 18 months Prod 2: 9 months	Min-Max Ade-Fab1
Simultaneous Improvement Prod 1: 12 months Prod 2: 12 months	Prod 1 (33) Prod 2 (33) 33 + 33 = 66	Prod 1 (25-10 = 15) Prod 2 (25+30 = 55) 15 + 55 = 70	Prod 1 (25+30 = 55) Prod 2 (25-15 = 15) 55 + 15 = 70	66 <-
Intensive Programme A Prod 1: 10 months Prod 2: 19 months	Prod 1 (25+20 = 45) Prod 2 (25-10 = 15) 45 + 15 = 60	Prod 1 (25-4 = 21) Prod 2 (25-4 = 21) 21 + 21 = 42	Prod 1 (25+40 = 65) Prod 2 (25-14 = 11) 65 + 11 = 76	42
Intensive Programme B Prod 1: 19 months Prod 2: 9 months	Prod 1 (25-12 = 13) Prod 2 (25+20 = 45) 13 + 45 = 58	Prod 1 (25-14 = 11) Prod 2 (25+40 = 65) 11 + 65 = 76	Prod 1 (25-4 = 21) Prod 2 (25-4 = 21) 21 + 21 = 42	42
MinMax AdeFab2	66 <-	76	76	

Each cell contains the percentage of sales for adeFab1, adding the percentage of sales for both products. The percentage of sales of each product depends on the difference in the time to market in each scenario. For instance, if both companies opt for simultaneous improvement, the products reach the market at the same time, and the percentage of sales for adeFab1 is 33% for the first product and 33% for the first product, which yields a total percentage of 66%. However, if AdeFab1 opts for simultaneous improvement, and AdeFab2 chooses the intensive program for product 1, AdeFab1 will lose 10% of the 25% market share of product 1, yielding 15%, whereas it will get a 30% extra for product 2, yielding 55%, and the sum of both products yields 15% + 55% = 70%. We can compute in a similar way the market shares in each scenario.

### 5.5.3 Setting oil prices in the 70s

In the 1970s, the OPEP countries adopted a price-linking strategy and oil prices went from €3/barrel to a level above €30/barrel. Analysts foresaw that prices could rise to over €100/barrel by the end of the twentieth century. Unexpectedly however, the cartel failed and prices went down again. Therefore, what are the conditions for these cartels to survive? In other words, what determines the equilibrium between cooperation and competition? This problem is similar to the prisoner’s dilemma.

To simplify matters, it is assumed that only two producers exist, Saudi Arabia and Iran, and that there are only three possible production levels: 2, 4 and 6 million barrels/day. So, depending on the two players’ decisions, total world production is 4, 6, 8, 10 and 12 million barrels/day. Accordingly, world prices are 35, 30, 25, 15 or only 10€/barrel. The production costs of one barrel are considered to be €2 for Saudi Arabia and €4 for Iran.

- a) The intention is to maximise the difference between the profit of a producer and that of its competitor. Consider and solve the case as if it were a zero-sum game with two players.
- b) If the idea is to seek OPEP’s prime objective by avoiding mutual competition and regulating oil production, sales and prices, what is the best option for Saudi Arabia and for Iran if they cooperate with each other and try to simultaneously obtain the best profit each?

**Solution**

a) Pay-off Matrix:

b) Iran a) Saudi Arabia	2 million barrels / day	4 million barrels / day	6 million barrels/ day	Saudi Arabia Mini-max
2 million barrels day	4	-48	-80	-80
4 million barrels / day	60	8	-14	-14
6 million barrels / day	96	34	12	12 <-
Iran MiniMax	96	34	12 <-	

This is a Pure Strategy Game. Thus, even though the dominated strategies are eliminated and the MiniMax approach is used, the production level of 6 million barrels/day will always be more profitable for Saudi Arabia. When both players adopt a dominant strategy, which is a production of 6 million barrels/day in this case, the players gain 48 and 36 million dollars, respectively.

b) OPEP's Pay-off Matrix:

b) Iran a) Saudi Arabia	2 million barrels / day	4 million barrels / day	6 million barrels/ day
2 million barrels day	66/62	56/104	46/126
4 million barrels / day	112/52	92/84	52/66
6 million barrels / day	138/42	78/44	48/36

The best option is that which offers the best profit to them both at the same time; that is, production of 4 million barrels/day, which implies profits of 92 and 84 million dollars for Saudi Arabia and Iran, respectively.

**5.5.4 Developing a High-Tech Product**

**Problem definition**

BIOSENS and TECHSENS, two competing firms, are developing a new state-of-the-art biomedical sensor, whose total sales will amount to 1,000 million dollars. Both must opt for using the most advanced “e1” or “e2” technology. Should both firms decide on the same technology, BIOSENS would obtain 60% of sales, since they know how to sell their products better than their competitor, which sells similar products. However, if both opt for different technologies, then TECHSENS would obtain a 60 % market share because it exploits differentiation better.

a) Determine which strategy each firm should adopt

Pay-off Matrix:

Techsens Biosens	e1	e2	
e1	60	40	p
e2	40	60	1-p
	q	1-q	



$$v = 60 \cdot p \cdot q + 40 \cdot p \cdot (1 - q) + 40 \cdot q \cdot (1 - p) + 60 \cdot (1 - p) \cdot (1 - q)$$

Player A Strategy:

$$q = 1$$

$$60 \cdot p + 40 \cdot (1 - p)$$

$$q = 0$$

$$40 \cdot p + 60 \cdot (1 - p)$$

$$20 \cdot p + 40 = -20 \cdot p + 60$$

$$p = 20/40 = 0.5$$

Player B Strategy:

$$p = 1$$

$$60 \cdot q + 40 \cdot (1 - q)$$

$$q = 0$$

$$40 \cdot q + 60 \cdot (1 - q)$$

$$20 \cdot q + 40 = -20 \cdot q + 60$$

$$q = 20/40 = 0.5$$

$$EMV = 200 \cdot 0.5 \cdot 0.5 = 50\% = 500M\text{€ each}$$

### 5.5.5 Investments Strategy

#### Problem definition

An investor wishes to invest €12,000 over a 1-year period, although he/she must decide to invest in market shares and/or gold. The profitability ratios estimated for each investment are based on the possible states of economy (see Table).

Investment	Growth	Mean Growth	Stabilisation	Poor Growth
Market shares	5	4	3	-1
Gold	2	3	4	5

Consider a zero-sum Games Theory model with two players and answer the following:

**a)** How should the investor invest the €12,000 to carry out the optimal strategy? The interpretation of this Games Theory problem is that the investor is playing against the states of nature (the states of economy). Since there is no data available regarding the probabilities of the states of nature, the optimal strategy may be based on a mixed strategy considering the market as the other player. Let us add a constant  $k = 1$  to make all the profitability ratios positive. Adding this constant value to the table yields:

Investment	Growth	Mean Growth	Stabilisation	Poor Growth
Market shares	6	5	4	0
Gold	3	4	5	6

The optimal strategy for the investor is obtained through the following continuous linear programming problem:

$$\min z = x_1 + x_2$$

s.t.

$$6 * x_1 + 3 * x_2 \geq 1$$

$$5 * x_1 + 4 * x_2 \geq 1$$

$$4 * x_1 + 5 * x_2 \geq 1$$

$$0 * x_1 + 6 * x_2 \geq 1$$

where:

$$p_1 = \frac{x_1}{(x_1+x_2)}$$

Is the probability that the investor will invest in market shares if he/she chooses the optimal strategy, and:

$$p_2 = \frac{x_2}{(x_1+x_2)}$$

Is the probability that the investor will invest in gold if he/she chooses the optimal strategy.

Therefore, given 12K€ to invest, the investor should invest  $12K€p_1$  in marketing and  $12K€p_2$  in gold.

We would need a solver to find the actual values. The values provided by a solver are  $p_1 = 0.33$  and  $p_2 = 0.67$ , therefore the investor should invest  $12K€0.33 = 4K€$  in marketing and  $12K€0.67 = 8K€$  in gold.

**b)** What mean profitability value is obtained? Recall that the objective function is inversely proportional to the lower bound of the profitability ratio, and that we added a constant  $k = 1$  to make all the profitability ratios positive, so:

$$z = x_1 + x_2 = \frac{p_1+p_2}{v} = \frac{1}{v}$$

With this expression we can obtain the mean profitability value, recalling that we need to subtract  $k$ , since we added the constant to make all the profitability ratios positive.

### 5.5.6 Playing Bluff

#### Problem definition

Two students of EDEM, Jaime and Lucas, are playing a game named Bluff, that goes as follows. Jaime writes a number on a piece of paper. Without showing Lucas what he has written, he tells him what he wrote. Jaime might be lying or telling the truth. Then Lucas must guess if Jaime is lying or telling the truth, or he can choose to pass and not to continue playing. If Lucas discovers that Jaime lied, Jaime must pay Lucas €50. If Lucas wrongly accuses Jaime of lying, Lucas pays him €25. If Jaime told the truth and Lucas guesses this, then Jaime pays Lucas €5. If Jaime lied and Lucas does not guess that he lied, then Lucas pays Jaime €24. Should Lucas decide to pass, then neither player wins or loses anything.

**a** Consider this a 2-player's zero-sum game in game theory and write down the tabular representation. Let us note Jaime as player A and Lucas as player B. The game can be represented through the following matrix:

Player B	Player A	Player A lied	Player A told truth	Pass
lie		-50	24	0
Tell the truth		25	-5	0

**b** How would you determine the game mean value and the mixed strategies of both players in this case?

Since this is a 2x3 game, let us formulate a Continuous Linear Programming problem to find the mixed strategies and the game mean value with these strategies.

**c** Using the method seen in class to calculate the mixed strategies, a solver provides the following solution for Lucas ( $u_1=0$ ,  $u_2=0$ ,  $u_3=0.02$ ), where  $u_1$  is the decision variable associated to the alternative "Jaime lied",  $u_2$  the decision variable associated to the alternative "Jaime told the truth" and  $u_3$  the decision variable associated to "Pass". Write down

the model and determine his optimal strategy. First, let us add a constant K to the game such that game values are non-negative:

$K = -50$  yields

Player B \ Player A	Player A lied	Player A told truth	Pass
lie	0	74	50
Tell the truth	75	45	50

Now, for problem A, the CLP:

$$\min z = x_1 + x_2$$

*s.t*

$$0 * x_1 + 75 * x_2 \geq 1$$

$$74 * x_1 + 45 * x_2 \geq 1$$

$$50 * x_1 + 50 * x_2 \geq 1$$

can be used to calculate the optimal strategies for player A. Once we obtain the values, the probabilities that define the optimal strategy for player A are defined a

$$p_1 = \frac{x_1}{x_1 + x_2}$$

$$p_2 = \frac{x_2}{x_1 + x_2}$$

And the mean game value is:

$$z = \frac{1}{x_1 + x_2} - 50$$

The optimal strategy for Player B can be obtained from the dual problem:

$$\max z = u_1 + u_2 + u_3$$

*s.t*

$$0 * u_1 + 74 * u_2 + 50 * u_3 \leq 1$$

$$75 * u_1 + 45 * u_2 + 50 * u_3 \leq 1$$

In this case the probabilities are given by:

$$q_1 = \frac{u_1}{u_1 + u_2 + u_3}$$

$$q_2 = \frac{u_2}{u_1 + u_2 + u_3}$$

$$q_3 = \frac{u_3}{u_1 + u_2 + u_3}$$

The value of the objective value can also be computed from the decision variables of the dual as:

$$z = \frac{1}{u_1 + u_2 + u_3} - 50$$

Now, plugin-in the values provided by the solver, we see that the optimal strategy for lucas is always to pass:

$$q_1 = \frac{u_1}{u_1 + u_2 + u_3} = \frac{0}{0+0+0.02} = 0$$

$$q_2 = \frac{u_2}{u_1 + u_2 + u_3} = \frac{0}{0+0+0.02} = 0$$

$$q_3 = \frac{u_3}{u_1 + u_2 + u_3} = \frac{0.02}{0+0+0.02} = 1$$

Also, the game value obtained is 0:

$$z = 1/0.02 - 50 = 0$$

### 5.5.7 The Squid Game

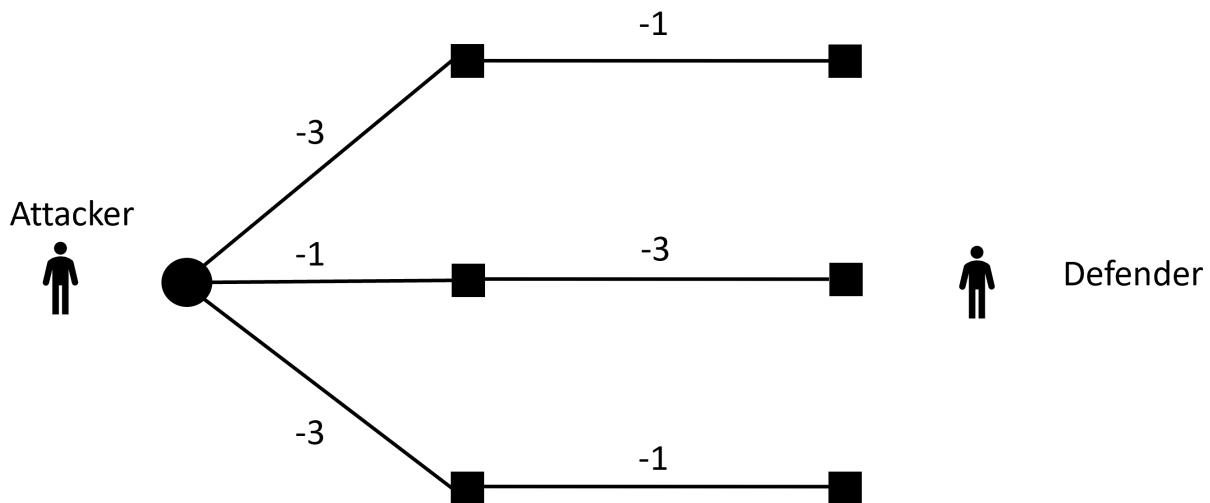
#### Problem definition

You were out celebrating the end of the exams at EDEM and the next thing you know is you are trapped in an island playing for your life to a game known as the Squid Game. There are two players in the Squid Game, the attacker and the defender. The attacker must pass through the field, choosing one out of three possible paths, up, middle, and down. The defender must guess which path the attacker has selected, and follow the same path to meet him in the middle of the field.

Both players have a limited endurance, measured in endurance points, and start with 10 endurance points. The winner is the player that resists the longest time without losing all endurance points.

The field is fraught with traps that cause injuries to the players. If the attacker crosses the 'up' or 'down' paths, the damage caused to the attacker is assessed as -3 endurance points. If the attacker chooses the 'middle' path, the damage is assessed as -1 endurance points. If the defender chooses the 'up' or 'down' paths, the damage is -1 endurance points. The damage for the defender is -3 endurance points if he chooses the middle path.

Both players are blindfolded and choose the path they would like to follow at the same time. If the defender chooses the same path as the attacker, the damage for the attacker is -5 endurance points. If on the other hand the defender chooses a different path, the damage for the defender is -2 endurance points. The following image shows the Squid Game field and the damage in storage points in each path.



a) Write down the tabular representation of the game. Let us note the Attacker as Player A and the defender as Player B. The tabular representation, noting the first value of the tuple is the gain for player A and the second value after the slash is the value for player B is:

Player A / B	Up	Middle	Down
Up	-3/-5=-8/-1	-3/-3=-2=-5	-3/-1=-2=-3
Middle	-1/-1=-2=-3	-1/-5=-6/-3	-1/-1=-2=-3
Down	-3/-1=-2=-3	-3/-3=-2=-5	-3/-5=-8/-1

In short:

Player A / B	Up	Middle	Down
Up	-8/-1	-3/-5	-3/-3
Middle	-1/-3	-6/-3	-1/-3
Down	-3/-3	-3/-5	-8/-1

b) Find the move each player would choose following the minmax criteria. Discuss briefly if there is equilibrium. Applying the min max criteria, we add a new column to note the worst case scenario for each alternative of player A, and a new row to note the worst case alternative for each alternative of player B.

Player A / B	Up	Middle	Down	min
Up	-8/-1	-3/-5	-3/-3	-8
Middle	-1/-3	-6/-3	-1/-3	-6
Down	-3/-3	-3/-5	-8/-1	-8
Min	-3	-5	-3	

Player A would select the Middle as it provides the maximum value across the three worst case scenarios (-6). Player B would select either up or down since they both provide the minimum value (-3). The problem does not have equilibrium, since player B will probably would like to change strategy after few moves.

c) Now think of the game as a zero-sum game where the gain in each move is the difference in damage points between the attacker and the defender for each combination of pairs of paths selected. Write down the linear programming problems to calculate the optimal probabilities of the mixed strategies for both players.

Player a / A	Up	Middle	Down
Up	7	-2	0
Middle	-2	3	-2
Down	0	-2	7

Let us add a constant ( $k = 2$ ) to create an equivalent problem where all the values are positive:

Player a / A	Up	Middle	Down
Up	9	0	2
Middle	0	5	0
Down	2	0	9

Now, the following CLP problem is used to calculate the probabilities of player A:

$$\max z = x_1 + x_2 + x_3$$

$$9 * x_1 + 0 * x_2 + 2 * x_3 \leq 1$$

$$0 * x_1 + 5 * x_2 + 0 * x_3 \leq 1$$

$$2 * x_1 + 0 * x_2 + 9 * x_3 \leq 1$$

The strategy for Player B is obtained with the dual problem:

$$\min z = u_1 + u_2 + u_3$$

$$9 * u_1 + 0 * u_2 + 2 * u_3 \geq 1$$

$$0 * u_1 + 5 * u_2 + 0 * u_3 \geq 1$$

$$2 * u_1 + 0 * u_2 + 9 * u_3 \geq 1$$

## 5.5.8 Strategies for political campaigns

### Problem Definition

Suppose there are two political parties, The Avengers Party (a coalition of superheroes who have come together to fight for the greater good. They believe in using their powers to protect the people and promote equality and justice) and the Brotherhood party (This party represents a group of mutants who feel oppressed by society and seek to overthrow the existing power structures). They are competing in an election. Each party has three possible strategies: they can focus their campaign on young citizens (below 30), middle-aged citizens (30-50), or senior citizens (over 50). The parties use national statistics to estimate the number of votes that can shift in each demographic group. Here are the estimated values for each demographic group:

- Young voters: They represent 60% of the undecided voters.
- Middle-aged voters: They represent 25% of the undecided voters.
- Senior citizens: They represent 15% of the undecided voters.

From past elections, the Avengers party has collected the following estimation about undecided voters on each scenario:

- **Young voters:** If both parties focus on young voters, the Avengers party will get 40% of the young voters and the Brotherhood party 60% of the young voters. If the Avengers party focuses on the young voters and the Brotherhood party does not, the Avengers party will get 70% of the young voters. If the Brotherhood party focuses on the young voters and the Avengers party does not, the Avengers party will get 10% of the young voters. If neither party focuses on the young voters, the Avengers party will get 40% of the young voters.
- **Middle-aged voters:** If both parties focus on middle-aged voters, the Avengers party will get 60% of the middle-aged voters and the Brotherhood party 40% of the middle-aged voters. If the Avengers party focuses on the middle-aged voters and the Brotherhood party does not, the Avengers party will get 90% of the middle-aged voters. If the Brotherhood party focuses on the middle-aged voters and the Avengers party does not, the Avengers party will get 30% of the middle-aged voters. If neither party focuses on the middle-aged voters, the Avengers party will get 60% of the middle-aged voters.
- **Senior citizens:** If both parties focus on senior citizens, the Avengers party will get 70% of the senior citizens and the Brotherhood party 30% of the senior citizens. If the Avengers party focuses on the senior citizens and the Brotherhood party does not, the Avengers party will get 90% of the senior citizens. If the Brotherhood party focuses on the senior citizens and the Avengers party does not, the Avengers party will get 20% of the senior citizens. If neither party focuses on the senior citizens, the Avengers party will get 70% of the senior citizens.

Let us use Game Theory to model the strategies of both political parties, following these steps:

1. Identify the alternatives for both players and draw a tabular representation of the game were Player A is the Avengers Party

Note that each political party can choose only one out of three possible strategies: Focusing on young, middle-aged, or senior citizens. Therefore, the Avengers Party has three alternatives, noted as:

- Young: Focus on young citizens
- Middle-aged: Focus on middle-aged citizens
- Senior: Focus on senior citizens

The Brotherhood Party also has three possible strategies:

- Young: Focus on young citizens
- Middle-age: Focus on middle-aged citizens
- Senior: Focus on senior citizens.

For the sake of simplicity, The Avengers Party will be Player A and the Brotherhood Party will be Player B.

1. Complete the tabular representation of the game, estimating the total percentage of undecided voters that would vote for each party

Now, since each player has three different strategies, this is 3x3 game, and therefore we have a total of 9 possible scenarios. In each possible scenario, we need to assess the total percentage of undecided voters that would vote for each party. For example, if both parties focus on young voters, the Avengers Party will get 40% of the young voters and the Brotherhood party 60% of the young voters. Therefore, the Avengers Party will get 40% of 60% of the undecided voters in this segment, which is 24% of the total undecided voters. Similarly, the Brotherhood Party will get 60% of 60% = 36% of the total undecided voters. Additionally, since we know that both parties are focused on young voters, neither will focus on middle-age voters. Therefore, the Avengers Party will get 60% of 25% = 15% of the total undecided voters in this segment, and the Brotherhood Party will get 40% of 25% = 10% of the total undecided voters in this segment. Similarly, the Avengers Party will get 70% of 15% = 10.5% of the total undecided voters in the senior segment, and the Brotherhood Party will get 30% of 15% = 4.5% of the total undecided voters in the senior segment:

- Player A: Young, Player B: Young
  - Player A:  $0.60 \cdot 4 + 0.60 \cdot 25 + 0.7 \cdot 0.15 = 0.24 + 0.15 + 0.105 = 0.495 = 49.5\%$
  - Player B:  $0.60 \cdot 6 + 0.60 \cdot 25 + 0.3 \cdot 0.15 = 0.36 + 0.15 + 0.045 = 0.555 = 55.5\%$

Note that the sum of the percentages for each player is 100%, since we are considering all the undecided voters, and therefore we can also represent the game as a zero-sum game and do only have of the required calculations.

If we repeat this process for each of the 9 possible scenarios, we will get the following table:

		Player B (Brotherhood Party)		
		Young	Middle-aged	Senior
Player A (Avengers Party)	Young	49.5, 50.5	60, 40	60, 40
	Middle-aged	39, 61	49.5, 50.5	49.5, 50.5
	Senior	34.5, 65.5	45, 55	49.5, 50.5

2. Calculate the strategy of each party using the MinMax criteria

Now, if we apply the MinMax criteria, for the Avengers Party, we first need to calculate the value for the worst case scenario in each of the alternatives, and then select the alternative with the highest value. The worst case scenario can be calculated using the value for Player A and calculating the minimum of each row, that is:

- Young:  $\min(49.5, 60, 60) = 49.5$
- Middle-aged:  $\min(39, 49.5, 49.5) = 39$
- Senior:  $\min(34.5, 45, 49.5) = 34.5$

The best worst case scenario is the one with the highest value, which is  $\max(49.5, 39, 34.5) = 49.5$ , and therefore the Avengers Party should focus on young voters.

Similarly, for the Brotherhood Party, we first need to calculate the value for the worst case scenario in each of the alternatives, now the minimum of the value for Player B in each column, and then select the alternative with the highest value:

- Young:  $\min(50.5, 61, 65.5) = 50.5$

- Middle-aged:  $\min(40, 50.5, 55) = 40$
- Senior:  $\min(40, 50.5, 50.5) = 40$

Now, the best worst case scenario is the one with the highest value, which is  $\max(50.5, 40, 40) = 50.5$ , and therefore the Brotherhood Party should also focus on young voters.

Modelling the problem as a zero-sum game yields the same results.



## SIMULATION AND MARKOV CHAINS

### 6.1 Tutorials

#### 6.1.1 Simulation

##### Introduction

Simulation is a decision support tool that provides deep insights of the behaviour of a system, by representing it through a mathematical model and using a computer to run the model and obtain a virtual representation of the system. Thus, the main concepts are:

- **Simulation Model:** Set of mathematical rules (e.g. differential equations, state machines) that defines the behaviour of the system. Normally, the simulation model is formulated such that future system states are obtained from the current state, to model the system behaviour over time.
- **Simulation:** We refer to the execution of the simulation model as simulation, that is, simulation consists of taking the model through state changes over time to obtain a (virtual) representation of its behaviour

Simulation is used when there is no mathematical or analytical solution to a problem, or when the available methods are just too complex. From a practical point of view, simulation is also used when we want to experiment with the system behaviour before it is build, to get a better understanding of it, or just to visualize, demonstrate, or experiment with its behaviour. In some applications, it is not possible to perform such experiments with the real system, due to economic reasons (it is too expensive), security (it is too dangerous) or ethical reasons. Computers can run simulation models through state changes at much higher rates than the actual system behaviour. For instance, a computer may run a simulation of a model of the evolution of the earth climate through thousand of years in a matter of hours. Thus, simulation can also be used to save time. Finally, by running simulations before the real system we may also gain control over system conditions.

##### Simulation Model Types

###### System Dynamics

System dynamic models use differential equations to model the system behaviour of the model. System Dynamics models are also known as Continuous Simulation models or Differential-Algebraic Equations (DAE) models. One of the main characteristics of these models is that the lack of individuality of the inner components of the system. A set of differential equations model the system behaviour as a hole, where its individual elements are not represented individually, but instead, they are grouped into entities like Queues or Stocks. The model just defines the casual relationships and flows between the stocks.

For instance, a system dynamics model can be used to model the adoption of a product. In this example, all potential customers are modeled as a stock variable that depends on time  $p_s(t)$ . The simulation is initialised with an initial population of potential customers of  $p_s(0)$ . The actual customers on the other hand are modeled as another stock variable  $c_s(t)$ . Let us assume that there are no customers at the beginning of the simulation and that therefore  $c_s(0) = 0$ . The adoption rate

$a$  represents the flow of customers between these two stocks, that is  $\frac{\delta c_s(t)}{\delta t} = a$  and  $\frac{\delta c_p(t)}{\delta t} = -a$ . Thus, we can obtain the number of potential customers or customers at any given time  $t$  as:

$$c_s(t) = c_s(0) + a * t = a * t$$

$$p_s(t) = p_s(0) - a * t$$

Note that in this model, individual customers are not represented.

### Discrete Events

In discrete events' simulation, the system is modeled as a flow chart that represents the system as a sequence of steps. Therefore, the blocks in the flow chart represent different steps in the overall system. The flowchart blocks are modeled as stochastic processes that drive the simulation model execution. These stochastic processes are computed at discrete instant time events to represent the system state through the flow chart.

### Agent based

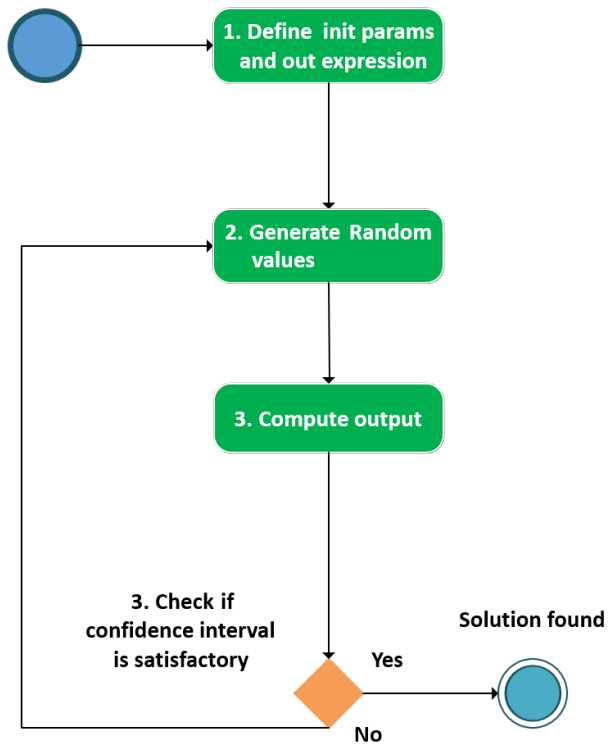
Agent based models are decentralised models, where each agent has individual behavioural rules that define how they interact with each other. This is a bottom-up approach where the system model is build from individual models of its inner components.

### Montecarlo Simulation

Montecarlo Simulation is probably the first simulation technique, and it basically consists on the computation of a mathematical model of the system which depends on random variables. The model is computed for a number of iterations (simulations) and the output is computed as the average. By the central limit theorem, this average will converge to a normal distribution, and therefore, it is possible to establish some confidence intervals to bound the simulation result.

Thus, the steps in a Montecarlo simulation are:

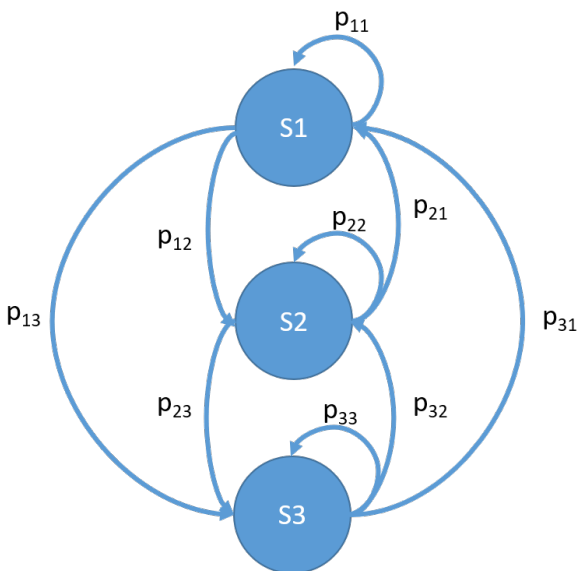
- **Define the output expression:** Define a mathematical model of the variable(s) that we want to simulate, the model implements random variables.
- **Generate random values:** Generate values for the random input variables of the model. The generated values must be defined in a valid input domain using a probability distribution (e.g. normal or uniform distribution)
- **Compute output:** Compute the value of the output expression for the random input values using the output expression. The output expression is computed a given number of iterations and the results are collected to get some statistics. The output of the simulation is the average value of the different iterations.
- **Check confidence interval of solution:** Finally, we check the confidence intervals for a given confidence level and if the confidence interval is not satisfactory, the number of simulations should be increased.



## 6.1.2 Markov Chain

### Introduction

Markov chains are stochastic processes in which the state of the system alters between a given set of possible states (i.e. a state machine), according to a set of transition probabilities between states. This concept is illustrated in the following image:



Note that the transition probabilities do not depend on previous states. This *memorylessness* characteristic is known as a **Markov property** and is the most outstanding characteristic of Markov Chains.

Markov chains allow to analyse the performance of a system and compare various alternatives to support decision making.

### Set up

A Markov Chain is a sequence of stochastic variables:

$$= 1, 2, 3, \dots, t$$

That represent the sequence of states of a system in a sequence of discrete time events ( $t \in [0, 1, \dots, T]$ ). At any given instance of time, the system can only be in one of the possible  $n$  states  $S = [S_1, S_2, \dots, S_n]$ , that is:

$$X_t \in [S_1, S_2, \dots, S_n] \quad \forall t \in [1, 2, \dots, T]$$

As in the figure above, an  $n$ -state Markov system is characterised by a  $n \times n$  transition probability matrix which contains the different transition probabilities. For instance, the 3-state system in the image above is a system with 3 possible states  $= [1, 2, 3]$ . The 3-state system is characterised by the matrix:

$$P^{(1)} = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix}$$

where  $p_{ij}$  is the probability that the system is in state  $j$  in the next instant of time, given that the system is in state  $i$ :

$$p_{ij} = P(X_{t+1} = S_j | X_t = S_i) \quad \forall t \in [1, 2, \dots, T]$$

In other words, the probability that the system is in state  $j$  in instant  $t + 1$  only depends on the state  $i$  in which the system was in the previous instant  $t$ , and does not depend on the states in which it was before. This probability is known as a **one-step transition** and consequently, the matrix  $P^{(1)}$  is known as the one-step probability matrix. The markov property states that this probability does not change with time, that it can be considered stationary. This is the main modeling assumption made when using Markov chains to model the behavior of a system.

Let us now vector represent the probability of being at any given state. Then, the probabilities in  $t+1$  can be estimated as:

$$V_{t+1} = V_t * P^{(1)}$$

For instance, in the example above, let us assume that the system is in state 1 at instant  $t$ :

$$V_t = [1 \quad 0 \quad 0]$$

The probabilities of the system at state  $t + 1$  are:

$$V_{t+1} = V_t * P^{(1)} = [p_{11} \quad p_{12} \quad p_{13}]$$

Similarly, the probabilities of the system after  $k$  steps is:

$$V_{t+k} = V_t * P^{(k)} = V_t * (P^{(1)})^k$$

When  $k$  is large enough, the transition probabilities stabilise and the probability that a system is in any given state do not depend on the initial state. These probabilities are called stationary probabilities and are calculated using the following system of equations:

$$\pi_j = \sum_{i=1}^n \pi_i * p_{ij} \quad \forall j$$

$$\sum_{j=1}^n \pi_j = 1$$

where  $\pi_j$  is the stationary probability that the system is in state  $j$ .

Stationary probabilities represent the long term behavior of the system, and can be used to gain insights on the system behaviour.

For instance, if there is a cost  $c_j$  associated to the system state  $j$ , the **long term average cost**  $C$  can be calculated as:

$$C = \sum_{j=1}^n c_j * \pi_j$$

## 6.2 MonteCarlo Method Exercises

In this section you have an example of a simulation solved with the Montecarlo method:

### 6.2.1 Sales commissions forecast

Try me



#### Problem Definition

A company with 5 sales managers wants to provision budget to pay sales commissions to sales managers. The company applies a commission rate based on the percentage of target sales obtained, given by the following table:

Sales / Target ratio	Commission Rate
0-90 %	2%
90-100%	3%
>= 100%	4%

Each salesman will be compensated with the commission rate times the total sales obtained. The following table shows the target sales for the five sales managers:

Sales Manager	Sales Target (€)
1	100,000
2	200,000
3	75,000
4	400,000
5	500,000

- a) Estimate the budget for sales commissions the company has to pay in the scenario where all sales managers get exactly the 100% of the sales target (naive approach).
- b) The company has a historic record of sales for the five sales managers and from this record, it can estimate that the Percent to Plan (The ratio between the actual sales and the sales target) can be modelled by a normal distribution with a mean of 100% and standard deviation of 10%. Use this insight to estimate the budget for sales commissions using a MonteCarlo distribution.

## 6.2.2 Return of Investments for online marketing

Try me



The following metrics and indicators determine the expenditures and return of investments of marketing investments in paid inclusion (including your company's web page in search results) for your company:

- **S (Total Search results in period):** Total number of times the keywords are used in searches in the planning period.
- **CPM (Cost per 1000 impressions):** Cost of the search engine per every 1000 impressions, every time your website appears in a search result
- **CPC (Cost per click):** Cost per every click
- **CTR (Click Through Rate):** rate between the number of clicks and number of impressions
- **CVR (Conversion Rate):** rate between the number of visitors and the number of users

With this, the number of visitors ( $V$ ) and the number of users ( $U$ ) can be calculated as:

$$U = V * CVR = S * CTR * CVR$$

The total costs ( $C_t$ ) of the search engine investmets are:

$$C_t = CPC * V + CPM/1000 * S$$

The **ARPU (Average revenue per user in planning period)** is the average revenue per user, or the total revenue in the planning period ( $R$ ), divided by the number of users:

$$ARPU = R/U$$

The Return of Investments for online marketing is calculated as:

$$ROI = \frac{R - C_t}{C_t}$$

After some research, you have found that the different metrics can be modeled as normal distributions with the following parameters:

Metric	Mean	Standard Deviation
Search results	600000	100000
CTR	0.05	0.01
CVR	0.5	0.1

These are the search engine fees for your keywords: - CPM = 0,3€ - CPC = 0,75€

You estimate your ARPU in 5€

- Calculate the total costs, the total Return of Investments and the average cost per acquisition using Montecarlo Simulation
- Calculate the 95% confidence interval of the ROI

## 6.3 MonteCarlo Method Solved Exercises

In this section you have the solution to the different exercises.

### 6.3.1 Sales commissions forecast

Try me



#### Problem Definition

A company with 5 sales managers wants to provision budget to pay sales commissions to sales managers. The company applies a commission rate based on the percentage of target sales obtained, given by the following table:

Sales / Target ratio	Commission Rate
0-90 %	2%
90-100%	3%
>= 100%	4%

Each salesman will be compensated with the commission rate times the total sales obtained. The following table shows the target sales for the five sales managers:

Sales Manager	Sales Target (€)
1	100,000
2	200,000
3	75,000
4	400,000
5	500,000

**a)** Estimate the budget for sales commissions the company has to pay in the scenario where all sales managers get exactly the 100% of the sales target (naive approach).

**b)** The company has a historical record of sales for the five sales managers and from this record, it can estimate that the Percent to Plan (The ratio between the actual sales and the sales target) can be modelled by a normal distribution with a mean of 100% and standard deviation of 10%. Use this insight to estimate the budget for sales commissions using a MonteCarlo distribution.

### Solution

a) In the requested scenario, the sales obtained by each sales manager are represented in the table below:

Sales Manager	Sales Target (€)	Actual Sales (€)	Percent to Plan (%)	Commission Rate (€)	Commission Amount (€)
1	100,000	100,000	100	4	4,000
2	200,000	200,000	100	4	8,000
3	75,000	75,000	100	4	3,000
4	400,000	400,000	100	4	16,000
5	500,000	500,000	100	4	20,000

The total budget for sales commission can be obtained with the summation of the last column (51,000€)

b) In order to estimate the budget using Montecarlo, we are going to use the Python numpy package to calculate the probability distribution.

First we import the libraries we are going to use:

```
[45]: import pandas as pd
import numpy as np
```

Then we initialise the data needed to model the problem

```
[46]: avg = 1
std_dev = .1
num_simulations = 1000

sales_target_values = np.array([100000, 200000, 75000, 400000, 500000])

# Define a function to calculate the commission rate depending on the rate to target
def calc_com_rate(x):
    if x <= 0.9:
        return 0.02
    elif x <= 1:
        return 0.03
    else:
        return 0.04

# You can also use a lambda:
# calc_com_rate = lambda x: 0.02 if x <= 0.9 else 0.03 if x <= 0.99 else 0.04

# Vectorize the function so that we can apply it to vectors and matrices
v_calc_com_rate = np.vectorize(calc_com_rate)

# Define a list to keep all the results from each simulation that we want to analyze
all_stats = np.zeros((num_simulations, 3))
```

Now we run the simulations in a for loop:



```
[47]: # Loop through simulations
for i in range(num_simulations):

    # Choose random inputs for the sales targets and percent to target
    pct_to_target = np.random.normal(avg, std_dev, len(sales_target_values))

    #Calculate actual sales
    sales = pct_to_target*sales_target_values

    # Determine the commissions rate and calculate it
    commission_rate = v_calc_com_rate(np.array(pct_to_target))

    # Calculate the commission
    commission = sales*commission_rate

    # We want to track sales,commission amounts and sales targets over all the
    ↪simulations
    # Sum values among sales managers and calculate the mean commission rate
    all_stats[i,:] = [np.sum(sales),
                    np.sum(commission),
                    np.mean(commission_rate)]

results_df = pd.DataFrame.from_records(all_stats, columns=['Sales',
                                                         'Commission_Amount',
                                                         'Commission_Rate'])
```

Finally, we represent the results and calculate the confidence interval:

```
[48]: results_df.describe()
```

	Sales	Commission_Amount	Commission_Rate
count	1.000000e+03	1000.000000	1000.000000
mean	1.275014e+06	44052.103207	0.033898
std	6.787145e+04	6867.089676	0.003249
min	1.074120e+06	23942.116488	0.024000
25%	1.229842e+06	39561.533879	0.032000
50%	1.275006e+06	44543.493621	0.034000
75%	1.317589e+06	49169.062538	0.036000
max	1.478058e+06	58539.162650	0.040000

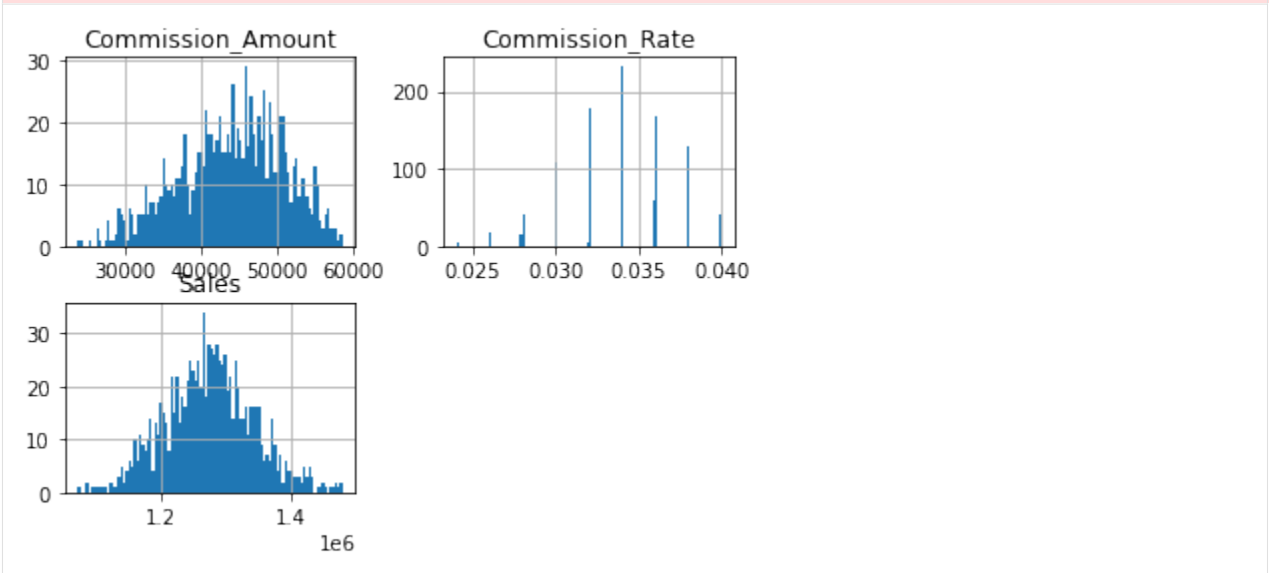
```
[49]: hist = results_df.hist(bins=100)

c:\users\ffraile\appdata\local\programs\python\python37\lib\site-packages\pandas\
↪plotting\_matplotlib\tools.py:307: MatplotlibDeprecationWarning:
The rowNum attribute was deprecated in Matplotlib 3.2 and will be removed two minor
↪releases later. Use ax.get_subplotspec().rowspan.start instead.
    layout[ax.rowNum, ax.colNum] = ax.get_visible()
c:\users\ffraile\appdata\local\programs\python\python37\lib\site-packages\pandas\
↪plotting\_matplotlib\tools.py:307: MatplotlibDeprecationWarning:
The colNum attribute was deprecated in Matplotlib 3.2 and will be removed two minor
↪releases later. Use ax.get_subplotspec().colspan.start instead.
    layout[ax.rowNum, ax.colNum] = ax.get_visible()
c:\users\ffraile\appdata\local\programs\python\python37\lib\site-packages\pandas\
↪plotting\_matplotlib\tools.py:313: MatplotlibDeprecationWarning:
The rowNum attribute was deprecated in Matplotlib 3.2 and will be removed two minor
↪releases later. Use ax.get_subplotspec().rowspan.start instead.
    if not layout[ax.rowNum + 1, ax.colNum]:
```

(continues on next page)

(continued from previous page)

```
c:\users\ffraile\appdata\local\programs\python\python37\lib\site-packages\pandas\
↳plotting\_matplotlib\tools.py:313: MatplotlibDeprecationWarning:
The colNum attribute was deprecated in Matplotlib 3.2 and will be removed two minor
↳releases later. Use ax.get_subplotspec().colspan.start instead.
if not layout[ax.rowNum + 1, ax.colNum]:
```



```
[50]: import scipy.stats as st
#Calculate the 95% confidence interval

# We collect the results from the data frame
a = np.array(results_df['Commission_Amount'])

# loc is used to center the distribution at mean of array
# scale is used to scale the distribution according to the standard error
# of the mean (st.sem)
arr_standard_dev = np.std(a)/(len(a)**0.5)
arr_mean = np.mean(a)
interval = (st.norm.ppf(0.025, loc= arr_mean, scale=arr_standard_dev), st.norm.ppf(0.
↳975, loc= arr_mean, scale=arr_standard_dev))

95% confidence interval is: (43626.697262449656, 44477.5091506711)
or
(43626.697262449656, 44477.5091506711)
```

```
[51]: import pandas as pd
import numpy as np

avg = 1
std_dev = .1
num_reps = 500
num_simulations = 1000

sales_target_values = np.array([100000, 200000, 75000, 400000, 500000])

# Define a lambda function to calculate the ratio
calc_com_rate = lambda x: 0.02 if x <= 0.9 else 0.03 if x <= 0.99 else 0.04
```

(continues on next page)

(continued from previous page)

```
v_calc_com_rate = np.vectorize(calc_com_rate)

# Choose random inputs for the sales targets and percent to target,
# this time create a matrix with as many rows as simulations
pct_to_target = np.random.normal(avg, std_dev, (num_simulations, len(sales_target_
↪values)))

# Reshape the sales target values into a matrix of adequate size
stv = np.broadcast_to(sales_target_values, (num_simulations, len(sales_target_
↪values)))

# Calculate the sales applying the ratio
sales = pct_to_target*stv

# Calculate commission rate
commission_rate = v_calc_com_rate(pct_to_target)

# And commission
commission = sales*commission_rate

# Sum values among sales managers and calculate the mean commission rate
all_stats = [np.sum(sales, axis=1), np.sum(commission, axis=1), np.mean(commission_
↪rate, axis=1)]

results_df = pd.DataFrame.from_records(np.transpose(all_stats), columns=['Sales',
                                                                    'Commission_Amount',
                                                                    'Commission_Rate'])

results_df.describe()
```

[51]:

	Sales	Commission_Amount	Commission_Rate
count	1.000000e+03	1000.000000	1000.000000
mean	1.274653e+06	43849.666014	0.033772
std	6.770009e+04	6868.545178	0.003330
min	1.070847e+06	22201.173382	0.020000
25%	1.228111e+06	39223.966965	0.032000
50%	1.275056e+06	44283.089045	0.034000
75%	1.320497e+06	49146.611184	0.036000
max	1.469480e+06	58207.805777	0.040000

### 6.3.2 Return of Investments for online marketing

Try me



The following metrics and indicators determine the expenditures and return of investments of marketing investments in paid inclusion (including your company’s web page in search results) for your company:

- **S (Total Search results in period):** Total number of times the keywords are used in searches in the planning period.

- **CPM (Cost per 1000 impressions):** Cost of the search engine per every 1000 impressions, every time your website appears in a search result
- **CPC (Cost per click):** Cost per every click
- **CTR (Click-Through Rate):** rate between the number of clicks and number of impressions
- **CVR (Conversion Rate):** rate between the number of visitors and the number of users

With this, the number of visitors (V) and the number of users (U) can be calculated as:

$$U = V * CVR = S * CTR * CVR$$

The total costs ( $C_t$ ) of the search engine investments are:

$$C_t = CPC * V + CPM/1000 * S$$

The **ARPU (Average revenue per user in planning period)** is the average revenue per user, or the total revenue in the planning period (R), divided by the number of users:

$$ARPU = R/U$$

The Return of Investments for online marketing is calculated as:

$$ROI = \frac{R - C_t}{C_t}$$

After some research, you have found that the different metrics can be modeled as normal distributions with the following parameters:

Metric	Mean	Standard Deviation
Search results	600000	100000
CTR	0.05	0.01
CVR	0.5	0.1

These are the search engine fees for your keywords: - CPM = 0,3€ - CPC = 0,75€

You estimate your ARPU in 5€

a. Calculate the total costs, the total Return of Investments and the average cost per acquisition using Montecarlo Simulation

Ok, Let's first import the libraries we will need:

```
[1]: import pandas as pd
import numpy as np
```

Ok, now we can start with the simulation. First, we will define the number of simulations we want to run. We will use 100000 simulations, but you can change this number to see how the results change. Once, we define the number of simulations, we will also define the properties of the random variables we want to simulate with Montecarlo (mean and standard deviation). We will use the numpy library to generate the random variables. Finally, in this basic implementation, we will define a for loop to run the simulation and store the results in a dataframe. We will also calculate the ROI for each simulation and store it in the dataframe. Finally, We will use the describe function to get the statistics of the results.

```
[ ]: # Number of simulations
num_simulations = 100000

# Parameters of the revenue model
#Data Search Results and search engines costs
#Data Search Results and search engines costs
search_results_avg = 600000
```

(continues on next page)

(continued from previous page)

```

search_results_std = 100000

ctr_mean = 0.05
ctr_std = 0.01

cvr_mean = 0.5
cvr_std = 0.1

# Costs and unitary profit
CPM = 0.3
CPC = 0.75

ARPU = 5

# We initialize a numpy array with the number of simulations and the number of
↳ variables we want to simulate
results = np.zeros((num_simulations, 8))

# We run the simulation
for i in range(num_simulations):
    # First we calculate the total number of impacts or searches
    total_impacts = np.random.normal(search_results_avg, search_results_std)

    # Then we calculate the total number of visitors using the Click-Through Rate
↳ (CTR)
    CTR = np.random.normal(ctr_mean, ctr_std)
    total_visitors = total_impacts * CTR

    # Then we calculate the total number of customers using the Conversion Rate (CVR)
    CVR = np.random.normal(cvr_mean, cvr_std)
    total_customers = total_visitors * CVR

    # Finally, we calculate the total cost and the total revenues, using the costs
↳ and the ARPU
    total_cost = CPC*total_visitors + CPM*total_impacts/1000
    total_revenues = total_customers * ARPU
    ROI = (total_revenues - total_cost)/total_cost

    # We store the results in the numpy array, in row i
    results[i,:] = [total_impacts, CTR, total_visitors, CVR, total_customers, total
↳ cost, total_revenues, ROI]

# We create a dataframe with the results
df_results = pd.DataFrame(results, columns = ['Total impacts', 'CTR', 'Total visitors
↳', 'CVR', 'Total customers', 'Total cost', 'Total revenues', 'ROI'])
results.describe()

```

This code works and is easy to interpret, but it does not leverage one of the most outstanding features of Numpy, working with vectors and matrices. Let's see how we can improve the code using vectors and matrices, avoiding the for loop:

```

[2]: num_simulations = 100000

#Data Search Results and search engines costs
search_results_avg = 600000
search_results_std = 100000

```

(continues on next page)

(continued from previous page)

```
ctr_mean = 0.05
ctr_std = 0.01

cvr_mean = 0.5
cvr_std = 0.1

# Costs and unitary profit
CPM = 0.3
CPC = 0.75

ARPU = 5

# Total number of impacts or searches, each row represents a simulation
total_impacts = np.random.normal(search_results_avg, search_results_std, num_
    ↳simulations)

# Click-Through Rate, each row represents a simulation
CTR = np.random.normal(ctr_mean, ctr_std, num_simulations)

# Total clicks or visitors
total_visitors = total_impacts * CTR

# Conversion rate
CVR = np.random.normal(cvr_mean, cvr_std, num_simulations)

#Total customers or users
total_customers = total_visitors * CVR

#Total search engine costs
total_cost = CPC*total_visitors + CPM*total_impacts/1000

#Total revenues estimated from ARPU
total_revenues = total_customers * ARPU

#ROI (total revenues - total costs)/total cost
ROI = (total_revenues - total_cost)/total_cost

results_df = pd.DataFrame({'impacts':total_impacts, 'CTR':CTR, 'Visitors':total_
    ↳visitors, 'CVR':CVR, 'Customers':total_customers, 'Revenues':total_revenues, 'Cost':
    ↳total_cost, 'ROI':ROI})

#Now we can see the statistics using the describe function:
results_df.describe()
```

[2]:

	impacts	CTR	Visitors	CVR \
count	1.000000e+05	100000.000000	100000.000000	100000.000000
mean	5.999030e+05	0.050052	30031.266831	0.499396
std	1.001245e+05	0.010003	7901.093998	0.099647
min	1.627489e+05	0.006749	3111.804727	0.095343
25%	5.327494e+05	0.043324	24461.139430	0.432389
50%	5.997867e+05	0.050051	29532.234129	0.499373
75%	6.674925e+05	0.056801	35082.456403	0.566536
max	1.045158e+06	0.092695	73104.943275	0.958135

(continues on next page)

(continued from previous page)

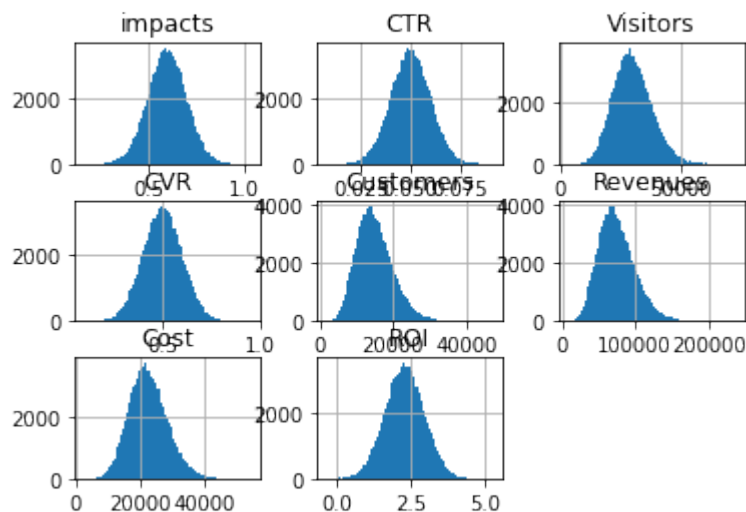
	Customers	Revenues	Cost	ROI
count	100000.000000	100000.000000	100000.000000	100000.000000
mean	14997.328542	74986.642710	22703.421018	2.301716
std	5015.323090	25076.615449	5945.011469	0.658838
min	1425.791553	7128.957767	2472.171271	-0.369106
25%	11421.034780	57105.173899	18513.188951	1.858309
50%	14445.367567	72226.837833	22325.493179	2.301226
75%	18009.074903	90045.374513	26502.929706	2.745805
max	47847.191767	239235.958836	55099.856282	5.337203

Note that the only difference is that now we have used the numpy library to generate bi-dimensional matrices with the random variables, and then just algebraic operations to calculate and store the results. This is much more efficient than using a for loop.

Ok, now we can answer the questions just use the function `hist()` of the dataframe to plot the histogram:

```
[3]: #Check if our distribution is normally distributed
```

```
hist = results_df.hist(bins=100)
```



Now, to calculate the confidence interval of the ROI, we can use the standard distribution, as we have a large number of simulations. We can use the scipy library to get the confidence interval, using the function `norm.ppf()` to get the quantiles of the standard distribution. We can also use the `describe` function to get the statistics of the results:

```
[4]: import scipy.stats as st
```

```
[5]: # Get the confidence interval of the ROI using the standard distribution
interval = (st.norm.ppf(0.025, loc=np.mean(ROI), scale=np.std(ROI)), st.norm.ppf(0.
↪975, loc= np.mean(ROI), scale=np.std(ROI)))
print(interval)
```

```
(1.010423544564778, 3.593008469189609)
```

The 95% confidence interval of the ROI will be between these 2 values

Note that the function `norm.ppf` takes the quantile as the first argument, and the mean and standard deviation as the second and third arguments. Since we want to get the 95% confidence interval, we use 0.025 and 0.975 as the quantiles.

To get the average and standard distribution of the ROI, we use the numpy functions `mean` and `std`, which will return the empirical mean and standard deviation of the data generated during the simulation.

```
[6]: # We can also get the empirical 95% confidence interval of all metrics using describe
results_df.describe(percentiles=[0.025, 0.975])
```

```
[6]:
```

	impacts	CTR	Visitors	CVR	\
count	1.000000e+05	100000.000000	100000.000000	100000.000000	
mean	5.999030e+05	0.050052	30031.266831	0.499396	
std	1.001245e+05	0.010003	7901.093998	0.099647	
min	1.627489e+05	0.006749	3111.804727	0.095343	
2.5%	4.026289e+05	0.030309	15950.632852	0.304002	
50%	5.997867e+05	0.050051	29532.234129	0.499373	
97.5%	7.955853e+05	0.069589	46908.820897	0.693940	
max	1.045158e+06	0.092695	73104.943275	0.958135	

	Customers	Revenues	Cost	ROI
count	100000.000000	100000.000000	100000.000000	100000.000000
mean	14997.328542	74986.642710	22703.421018	2.301716
std	5015.323090	25076.615449	5945.011469	0.658838
min	1425.791553	7128.957767	2472.171271	-0.369106
2.5%	6727.525477	33637.627384	12107.334547	1.010177
50%	14445.367567	72226.837833	22325.493179	2.301226
97.5%	26258.565698	131292.828488	35404.215418	3.587235
max	47847.191767	239235.958836	55099.856282	5.337203

## 6.4 Markov Chain Exercises

In this section you can find different Operations Research Problems modeled with Markov Chains:

### 6.4.1 Consultancy Firm

#### Problem Definition

ABC Consulting is a consultancy firm in which employees move through three levels: Junior Consultant, Senior Consultant and Manager. Changes of level are determined by the Human Resources (HR) Director based on an annual evaluation done for each employee. Only one change of level can be done per evaluation and employees never go down one level. After doing a statistical analysis, the HR Director has determined that the Junior Consultant has a 20 % probability of moving up one level after the evaluation, whereas this probability lowers to 5 % for a Senior Consultant. He has also established that there is a probability of 15, 10 and 5 % that a Junior Consultant, a Senior Consultant or a Manager, respectively, leaves the firm (to work for another firm or to retire) after knowing the result of his or her annual evaluation. For every employee who leaves the firm, a new Junior Consultant is contracted.

a) Model employee carriers as a Markov Chain

b) According to the model, calculate the stationary probabilities to determine What percentage of employees at each level will the company have in the long term



## 6.4.2 Alien Virus

### Problem Definition

Rick is a re-known scientist that is currently analysing the effects and mortality of an alien virus. Rick considers that, under the effect of the virus, a cell can be in three different states  $s_1$ =healthy,  $s_2$ =infected,  $s_3$ =dead.

**a** Rick thinks that the effect of the virus can be modelled as a Markov Process. Consider the transition probabilities as unknowns, taking into account that infected cells may heal but dead cells cannot be alive again. Draw the Markov process chain and the one-step transition probability matrix.

Rick makes a series of experiments by exposing a healthy cell culture to the virus and checking a sample every 10 hours. After 10 hours, 40% of the cells are infected, but none are dead. After 20 hours, 24% of the cells are dead and 36% are infected.

**b** Consider that the result of the 10 hours sample is the probability vector  $V_t(t=1)$  obtained from the one-step transition probability matrix above and a vector  $V_t(t=0)$  representing the healthy cell culture population. Calculate the transition probabilities from  $s_1$ .

**c** Consider now the result of the 20 hours sample as the probability vector  $V_t(t=2)$ . Can you determine the transition probabilities from  $s_2$ ?

## 6.4.3 Problem definition

Tina Belcher, brand-new manager of the new Bob's burger parlour downtown San Francisco, wants to analyse the impact on the market and the loyalty of the customers of Bob's Burger and Pesto's, an Italian restaurant nearby. It is assumed that one customer dines once a week at Bob's or at Pesto's, but not at both. Tina has collected information from 100 consumers over a 10-week period. When checking the data, she discovered that of all the customers who consumed at Bob's in a given week, 90 % returned to the restaurant following week, while 10 % switched to Pesto's. Of all the consumers who consumed at Pesto's in a given week, 80 % returned to the same place the following week, but 20 % switched to Bob's. Assuming that the transition probabilities are the same for every customer and that they not change over time, we can get insights modelling customer fidelity with Markov chains.

Model customer fidelity as a Markov process and answer the following questions:

- a)** If at week 0, a customer dines at Bob's, what is the probability of the same customer dining at Bob's in week 1?
- b)** What is the probability of a customer going back to Bob's two weeks running?
- c)** What is the probability of a customer changing to Pesto's after dining at Bob's and then going back to Bob's
- d)** Consider the customer of section a) what is the probability that this customer comes back to Bob's in two weeks time?
- e)** What is the expected market share of customers between the two restaurants?

## 6.4.4 Distribution of Operators in Workshop Areas

### Problem Definition

Rick & Morty Inc. is a small workshop that specialises in manufacturing special parts for the metallurgical industry. The firms' personnel comprises the Director Rick (an Industrial Organisation Engineer), the Designer Morty (an Industrial Design Engineer) and three highly specialised Operators. The workshop has two very different areas (A and B) given the type of machinery in them and the kind of works that can be done with it. The three Operators can move between the two areas as they prefer because they can all work on any of the workshop machines. The Director has noticed that the Operators tend to gather in the same area because their performance is enhanced if they help each other. Nevertheless, leaving an area without Operators for several days is not a good option because the tasks in this area are considerably delayed. By analysing last year's manufacturing reports, the Director has calculated that the two areas were not occupied

by the Operators for 30 % of the days. However, the most worrying aspect is that one area can be void of Operators for several days running. Therefore, the Director has decided to set up the following work plan: at the beginning of the working day, the three Operators will enter a draw to see which of them changes area so no area will be empty for more than 1 day running, or so the Director believes.

a) Consider the problem according to a Markov chain

b) For what percentage of time will one of the areas be void of Operators?

c) On average, how much time passes from there being no Operator in one zone to there being one Operator in it? And the time from an area with no Operator until this same situation occurs again?

[ ]:

### 6.4.5 Replacing a car

#### Problem definition

At the beginning of each year, a car is in good, regular or bad state:

- There is a 90% probability that a good car will be in a good state at the beginning of next year
- There is a 5% probability that it will be in a regular state
- and a 5 % probability that it will be in bad state.

If a car is in a regular state, then:

- There is a 70 % probability that the car is in again in regular state at the beginning of next year,
- and a 30 % probability that it will be in a bad state.

A bad car has no sale value and must be immediately replaced with a good one.

It costs €12,000 to buy a good car, but a regular one can be found for €5,000. Maintenance costs are €1,000/year for a good car and €2,000 for a regular car, and must be paid upfront at the beginning of the year.

Must I replace my car as soon as it is in a regular state, or must I wait until it is in a bad state?

### 6.4.6 Management of Computer virus threats

#### Problem definition

The IT department of your company has detected a threat in a software virus that affects the Operating System (OS) of the PCs used in the company. When the virus infects a computer, it sends emails to a set of corporate emails. Once the virus is installed in a computer, there is a procedure that can successfully remove the virus and restore the normal operation of the computer. However, in some situations, IT might not be able to restore the OS and the computer might become non-operative.

IT wants to use Markov Chains to model the status of a computer. After collecting data, they have determined that the probability that a computer is infected by the virus given that it was operating normally the day before is 0.05. The probability that the computer is non-operative for other reasons given that it was operating normally the day before is 0.1. Once a computer is infected, they always perform the virus removal procedure in the same day. They estimate that the probability that the procedure is successful is 70%, while in 30% of the cases the procedure renders the computer non-operative. When a computer is in non-operative state, the equipment supplier always replaces it the next business day.

a Define the one-step probability matrix of the Markov chain

**b** The maintenance cost of a computer is 3€/day when it operates normally. The cost of the virus removal procedure is 30€ and the cost of replacing a computer is 1200€. Calculate the long term average costs in computer maintenance for the IT department.

## 6.5 Markov Chain Solved Exercises

In this section you have the solution to the different exercises.

### 6.5.1 Consultancy Firm

#### Problem Definition

ABC Consulting is a consultancy firm in which employees move through three levels: Junior Consultant, Senior Consultant and Manager. Changes of level are determined by the Human Resources (HR) Director based on an annual evaluation done for each employee. Only one change of level can be done per evaluation and employees never go down one level. After doing a statistical analysis, the HR Director has determined that the Junior Consultant has a 20 % probability of moving up one level after the evaluation, whereas this probability lowers to 5 % for a Senior Consultant. He has also established that there is a probability of 15, 10 and 5 % that a Junior Consultant, a Senior Consultant or a Manager, respectively, leaves the firm (to work for another firm or to retire) after knowing the result of his or her annual evaluation. For every employee who leaves the firm, a new Junior Consultant is contracted.

**a)** Model employee carriers as a Markov Chain

We can define a Markov chain with one state per level:

- $s_1$ : Employee at Junior consultant level
- $s_2$ : Employee at Senior consultant level
- $s_3$ : Employee at Manager level
- $s_4$ : Employee leaves the firm

The time period between state changes is equal to the evaluation period (not specified).

Using the information collected by the HR Director, we can note the one-step probability matrix as:

$$P^{(1)} = \begin{bmatrix} 0.65 & 0.2 & 0.0 & 0.15 \\ 0.00 & 0.85 & 0.05 & 0.10 \\ 0.0 & 0.0 & .95 & 0.05 \\ 1.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$p_{41}$  is equal to 1 since every employee that leaves the firm, a new junior consultant is hired.

**b)** According to the model, calculate the stationary probabilities to determine What percentage of employees at each level will the company have in the long term

$$\pi_1 = 0.65 \cdot \pi_1 + \pi_4$$

$$\pi_2 = 0.2 \cdot \pi_1 + 0.85 \cdot \pi_2$$

$$\pi_3 = 0.05 \cdot \pi_2 + 0.95 \cdot \pi_3$$

$$\pi_4 = 0.15 \cdot \pi_1 + 0.1 \cdot \pi_2 + 0.05 \cdot \pi_3$$

$$\pi_1 + \pi_2 + \pi_3 + \pi_4 = 1$$

This yields:

$$0.35 \cdot \pi_1 = \pi_4$$

$$0.2 \cdot \pi_1 = 0.15 \cdot \pi_2$$

$$0.04 \pi_3 = 0.04 \cdot \pi_2$$

$$\pi_1 = 0.249$$

$$\pi_2 = 0.332$$

$$\pi_3 = 0.332$$

$$\pi_4 = 0.08715$$

## 6.5.2 Alien Virus

### Problem Definition

Rick is a re-known scientist that is currently analysing the effects and mortality of an alien virus. Rick considers that, under the effect of the virus, a cell can be in three different states  $s_1$ =healthy,  $s_2$ =infected,  $s_3$ =dead.

**a** Rick thinks that the effect of the virus can be modelled as a Markov Process. Consider the transition probabilities as unknowns, taking into account that infected cells may heal but dead cells cannot be alive again. Draw the Markov process chain and the one-step transition probability matrix.

The one-step probability matrix is given by:

$$P^{(1)} = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ 0 & 0 & 1 \end{bmatrix}$$

Since dead cells cannot be healthy or infected again, the probability of a dead cell remaining cell is 1, the rest of the transition probabilities are unknowns.

Rick makes a series of experiments by exposing a healthy cell culture to the virus and checking a sample every 10 hours. After 10 hours, 40% of the cells are infected, but none are dead. After 20 hours, 24% of the cells are dead and 36% are infected.

**b** Consider that the result of the 10 hours sample is the probability vector  $V_t(t=1)$  obtained from the one-step transition probability matrix above and a vector  $V_t(t=0)$  representing the healthy cell culture population. Calculate the transition probabilities from  $s_1$ . After 10 ours, our vector is:

$$V^1 = [0.6 \quad 0.4 \quad 0] = [1 \quad 0 \quad 0] * P^{(1)} = [p_{11} \quad p_{12} \quad p_{13}]$$

$$p_{11} = 0.6$$

$$p_{12} = 0.4$$

$$p_{13} = 0$$

**c** Consider now the result of the 20 hours sample as the probability vector  $V_t(t=2)$ . Can you determine the transition probabilities from  $s_2$ ?

We can obtain the missing values using the same procedure:

$$V^2 = [0.4 \quad 0.36 \quad 0.24] = [0.6 \quad 0.4 \quad 0] * P^{(1)}$$

$$0.4 = 0.6 * p_{11} + 0.4 * p_{21} \rightarrow p_{21} = \frac{0.4 - 0.36}{0.4} = 0.1$$

$$0.36 = 0.6 * p_{12} + 0.4 * p_{22} \rightarrow p_{22} = \frac{0.36 - 0.24}{0.4} = 0.3$$

$$0.24 = 0.6 * p_{13} + 0.4 * p_{23} \rightarrow p_{23} = \frac{0.24}{0.4} = 0.6$$

### 6.5.3 Problem definition

Tina Belcher, brand-new manager of the new Bob's burger parlour downtown San Francisco, wants to analyse the impact on the market and the loyalty of the customers of Bob's Burger and Pesto's, an Italian restaurant nearby. It is assumed that one customer dines once a week at Bob's or at Pesto's, but not at both. Tina has collected information from 100 consumers over a 10-week period. When checking the data, she discovered that of all the customers who consumed at Bob's in a given week, 90 % returned to the restaurant following week, while 10 % switched to Pesto's. Of all the consumers who consumed at Pesto's in a given week, 80 % returned to the same place the following week, but 20 % switched to Bob's. Assuming that the transition probabilities are the same for every customer and that they not change over time, we can get insights modelling customer fidelity with Markov chains.

#### Model

We model the decision of customers as a Markov chain, assuming two possible states every week (which is the discrete event period of our model). The states are:

- $s_1$ : represents the customers decision to go to Bob's Burger.
- $s_2$ : represents the customers decision to go to Pesto's.

Now, with the data collected by Tina, we can estimate the transition probability Matrix.

- $p_{11}$ : It is the probability that a customer goes back to Bob's Burger after they have been to Bob's Burger. Tina discovered that this probability is 0.9 or 90%.
- $p_{12}$ : It is the probability that a customer goes to Pesto's after going to Bob's Burger. Tina estimates that this probability is 10%. Clearly, it has to be  $(1 - p_{11} = 0.1)$  since we only consider two possible states and at any given time, the system has to be in one particular state, so the transition probabilities need to add up to 1.
- $p_{21}$ : It is the probability that a customer goes back to Bob's Burger after they have been to Pesto's. Tina discovered that this probability is 0.2.
- $p_{22}$ : It is the probability that a customer goes to Pesto's for two consecutive week, 0.8 according to Tina's data.

With this, the transition probability matrix is:

$$P^{(1)} = \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix}$$

We can use Graphviz to represent the Markov chain:

```
[ ]: from graphviz import Digraph
# Create Digraph object
din = Digraph(format='png')

# Add Bob's Burger node
din.node('s1', shape='circle', style="filled")

# Add Pesto's node
din.node('s2', shape='circle', style="filled")

# Add edges
din.edge('s1', 's1', '0.9')
din.edge('s1', 's2', '0.1')

din.edge('s2', 's1', '0.2')
din.edge('s2', 's2', '0.8')
```

(continues on next page)

din

Now, let us answer some questions:

**a)** If at week 0, a customer dines at Bob's, what is the probability of the same customer dining at Bob's in week 1?

Tina estimated that this is 90%, it is also possible to calculate it using the initial status and the probability Matrix, just in case we want to get more insights about this scenario. Since we are analysing a customer that goes to Bob's at time 0, the probability vector is:

$$V_0 = [p_1, p_2] = [1, 0]$$

The, at t=1, the probability vector becomes:

$$V_1 = V_0 \cdot P^{(1)} = [1, 0] \cdot \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix} = [0.9, 0.1]$$

**b)** What is the probability of a customer going back to Bob's two weeks running?

Now, in a Markov system, the possibility of a sequence of states, not taking into consideration any initial state and considering that the probability only depends on the previous state, is the product of the corresponding transition probabilities. In this example, we are analysing a sequence  $s_1, s_1, s_1$ , where the customer goes back to Bob's two consecutive weeks (or in other words, the customer dines at Bobs for three consecutive weeks). If we do not consider any initial state, the probability is:

$$p_{11} \cdot p_{11} = 0.9 \cdot 0.9 = 0.81$$

**c)** What is the probability of a customer changing to Pesto's after dining at Bob's and then going back to Bob's

Again, if we do not consider any initial state, we are analysing a sequence  $s_1, s_2, s_1$ , and the probability of this sequence is:

$$p_{12} \cdot p_{21} = 0.1 \cdot 0.2 = 0.02$$

**d)** Consider the customer of section a) what is the probability that this customer comes back to Bob's in two weeks time?

Now, coming back to the first example, we have an initial state, and now we want to consider the two-step transition probability matrix:

$$V_2 = V_0 \cdot P^{(2)} = [1, 0] \cdot \begin{bmatrix} 0.83 & 0.17 \\ 0.34 & 0.66 \end{bmatrix} = [0.83, 0.17]$$

**e)** What is the expected market share of customers between the two restaurants?

We can calculate the stationary probabilities representing the probability that a customer is in a restaurant when k tends to infinity. In this case, we need to solve the following linear system of equations:

$$\pi_1 = 0.9 \cdot \pi_1 + 0.2 \cdot \pi_2$$

$$\pi_2 = 0.1 \cdot \pi_1 + 0.8 \cdot \pi_2$$

$$\pi_1 + \pi_2 = 1$$

This yields:

$$0.1 \cdot \pi_1 = 0.2 \cdot \pi_2$$

$$\pi_2 = 1/3$$

$$\pi_1 = 2/3$$

That is, in the long run, it is expected that 66.67% of customers are at state 1 (Bob's Burger) and 33.33% of customers at state 2 (Pesto's).

## 6.5.4 Distribution of Operators in Workshop Areas

### Problem Definition

Rick & Morty Inc. is a small workshop that specialises in manufacturing special parts for the metallurgical industry. The firms' personnel comprises the Director Rick (an Industrial Organisation Engineer), the Designer Morty (an Industrial Design Engineer) and three highly specialised Operators. The workshop has two very different areas (A and B) given the type of machinery in them and the kind of works that can be done with it. The three Operators can move between the two areas as they prefer because they can all work on any of the workshop machines. The Director has noticed that the Operators tend to gather in the same area because their performance is enhanced if they help each other. Nevertheless, leaving an area without Operators for several days is not a good option because the tasks in this area are considerably delayed. By analysing last year's manufacturing reports, the Director has calculated that the two areas were not occupied by the Operators for 30 % of the days. However, the most worrying aspect is that one area can be void of Operators for several days running. Therefore, the Director has decided to set up the following work plan: at the beginning of the working day, the three Operators will enter a draw to see which of them changes area so no area will be empty for more than 1 day running, or so the Director believes.

a) Consider the problem according to a Markov chain We can model the problem as a Markov chain with the following states:

- $s_1$ : 0 operators in cell A and 3 operators in cell B
- $s_2$ : 1 operator in cell A and 2 operators in cell B
- $s_3$ : 2 operators in cell A and 1 operator in cell B
- $s_4$ : 3 operators in cell A and 0 operators in cell B

The period between states is one day.

Now, at the beginning of each day, the operators will enter a draw to see who must change area. Assuming that each one has a probability of  $P_c = 1/3$  to be selected in the draw and change cell, we can calculate the transition probability matrix. For instance, the probability of going from state  $s_1$  to state  $s_2$  is 1, since there are 3 operators in cell B and one of them must change area. The probability of going from state  $s_2$  to state  $s_1$  is  $1/3$ , since one of the operators in cell A must change area:

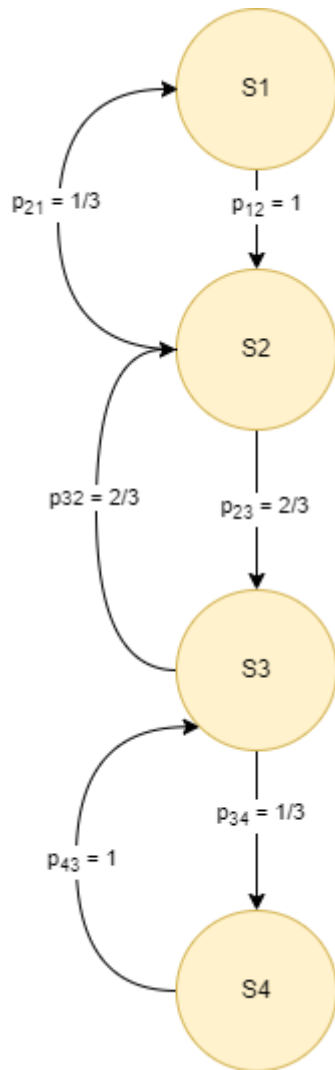
$$3 * P_c = 1$$

The probability of going from state  $s_2$  to state  $s_3$  is  $2/3$ , since the probability of the two operators in cell B to change area is  $2 * P_c = 2/3$ , and the probability of going from state  $s_2$  to state  $s_1$  is  $1/3$ , since the probability of the operator in cell A to change area is  $P_c = 1/3$ .

Following this reasoning yields a transition probability matrix equal to:

$$P^{(1)} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1/3 & 0 & 2/3 & 0 \\ 0 & 2/3 & 0 & 1/3 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

The following diagram shows the Markov chain:



b) For what percentage of time will one of the areas be void of Operators?

We can calculate the stationary probabilities since they will tell us which is the probability that the system is at any given state.

$$\pi_1 = 1/3 \cdot \pi_2$$

$$\pi_2 = \pi_1 + 2/3 \cdot \pi_3$$

$$\pi_3 = 2/3 \cdot \pi_2 + \pi_4$$

$$\pi_4 = 1/3 \cdot \pi_3$$

$$\pi_1 + \pi_2 + \pi_3 + \pi_4 = 1$$

This yields:

$$\pi_1 = 1/3 \cdot \pi_2$$

$$2/3 \cdot \pi_2 = 2/3 \cdot \pi_3$$

$$\pi_4 = 1/3 \cdot \pi_3$$

$$\pi_1 + 3 \cdot \pi_1 + 3 \pi_1 + \pi_1 = 1$$

$$\pi_1 = 1/8 = 0.125$$



$$\pi_2 = 3/8 = 0.375$$

$$\pi_3 = 3/8 = 0.375$$

$$\pi_4 = 1/8 = 0.125$$

A will be void 12.5% of the time and B 12.5% of the time, so one of the areas is going to be void 25% of the time. Rick is not a genius after all.

c) On average, how much time passes from there being no Operator in one zone to there being one Operator in it? And the time from an area with no Operator until this same situation occurs again?

This is left for future work!

## 6.5.5 Replacing a Car

### Problem definition

At the beginning of each year, a car is in good, regular or bad state:

- There is a 90% probability that a good car will be in a good state at the beginning of next year
- There is a 5% probability that it will be in a regular state
- and a 5 % probability that it will be in bad state.

If a car is in a regular state, then:

- There is a 70 % probability that the car is in again in regular state at the beginning of next year,
- and a 30 % probability that it will be in a bad state.

A bad car has no sale value and must be immediately replaced with a good one.

It costs €12,000 to buy a good car, but a regular one can be found for €5,000. Maintenance costs are €1,000/year for a good car and €2,000 for a regular car, and must be paid upfront at the beginning of the year.

Must I replace my car as soon as it is in a regular state, or must I wait until it is in a bad state?

### Solution

We can model the status of the car at the beginning of the next year as a Markov system, with three possible states:

- $s_1$ : represents the car being in good state.
- $s_2$ : represents the car being in regular state.
- $s_3$ : represents the car being in bad state.

With this model, we have to evaluate two possible scenarios. If we let the car transition to the regular state, the one-step transition probability matrix is:

$$P^{(1)} = \begin{bmatrix} 0.9 & 0.05 & 0.05 \\ 0 & 0.7 & 0.3 \\ 0.9 & 0.05 & 0.05 \end{bmatrix}$$

Since a bad car needs to be replaced immediately, even if the event occurs in the last day of the year, a car in a bad status is always replaced by a new one before the beginning of the next year, and therefore the transition probabilities are the same as for a new car.

Now, if we replace the car immediately as soon as it is in a regular state, the one-step probability matrix becomes:

$$P^{(1)} = \begin{bmatrix} 0.9 & 0.05 & 0.05 \\ 0.9 & 0.05 & 0.05 \\ 0.9 & 0.05 & 0.05 \end{bmatrix}$$

That is, as soon as the car is in a regular state, it is replaced by a new one. We have thus two different scenarios, and each step has different associated in each scenario.

We can calculate the stationary probabilities and corresponding long term average costs. In scenario 1, the long term probabilities are:

$$\pi_1 = 0.771429$$

$$\pi_2 = 0.142857$$

$$\pi_3 = 0.0857143$$

In this scenario the costs are:

- $C_1$ : 1000€ maintenance cost of a good car.
- $C_2$ : 2000€ maintenance cost of a regular car.
- $C_3$ : 13000€ cost of purchasing a new car plus year maintenance costs.

The long term average cost is:

$$C = \sum_i C_i \cdot \pi_i = 2,171$$

In the second scenario:

$$\pi_1 = 0.9$$

$$\pi_2 = 0.05$$

$$\pi_3 = 0.05$$

In this scenario the costs are: -  $C_1$ : 1000€ maintenance cost of a good car. -  $C_2$ : 8000€ cost of purchasing a new car plus year maintenance cost minus costs of selling a regular car. -  $C_3$ : 13000€ cost of purchasing a new car plus year maintenance costs.

We can calculate the long term average cost again as:

$$C = \sum_i C_i \cdot \pi_i = 1950$$

So, under these conditions, the second scenario provides a lower long term average cost

## 6.5.6 Machine Control System

### Problem definition

The control system of a certain machine comprises two independent controls, A and B. This system is activated every hour the machine operates to check that the machine works according to suitable parameters. The controls have a limited life span. Indeed, control A has been replaced 52 times in the last 1,300 h of operation, whereas control B has been substituted 39 times in the same time. Every time the system is activated, apart from controlling the machine, it warns that one (or both) of the controls has failed. The faulty control is replaced before the system is next activated. When one of the two controls is replaced, its reliability is lower because it has not been verified. Specifically, it is considered that when the system is next activated, the reliability of the substituted control is half its normal value. Replacing control A entails a cost of €70 and of €50 for replacing control B.

- a) Consider the problem according to a Markov chain. Calculate the failure probability of each control according to its historic of failures. Calculate the reliability
- b) What is the hourly cost of the control system?

We can define the following states:

- s1: None of the controls are replaced
- s2: control B is replaced
- s3: control A is replaced
- s4: Both controls are replaced.

This is the one step transition matrix:

states	s1	s2	s3	s4
s1	0.9312	0.0288	0.0388	0.0012
s2	0.4656	0.4944	0.0194	0.0206
s3	0.4656	0.0144	0.5044	0.0156
s4	0.2328	0.2472	0.2522	0.2678

Since control A has been replaced 52 times in 1300h, if we consider a sampling period of 1 hour, the fault probability is 0.04. For control B, fault probability is 0.03. The reliability is then 0.96 for control A and 0.97 for control B.

The probability that none of the controls is replaced is  $0.96 \cdot 0.97 = 0.9312$ , so this is the probability that we remain in state 1. The probability that only control A is replaced is  $0.04 \cdot 0.97$  (control A is faulty and control B is still operating normally). The probability that only control B is replaced is  $0.96 \cdot 0.03$ , and the probability that both remain operational is  $0.03 \cdot 0.04$ . This is how we obtain the first row.

The problem states that if one control is replaced, the reliability of the replaced component is half the normal value. Therefore, if control B is replaced, the reliability goes down from 0.97 to 0.485. Now, the probability that both controls remain operational is  $0.485 \cdot 0.96 = 0.4656$ . The probability that only control B is replaced is  $0.515 \cdot 0.96 = 0.4944$ . The probability that only control A is replaced is  $0.485 \cdot 0.04 = 0,0194$ . And finally, the probability that both controls are replaced is  $0.515 \cdot 0.04$ . This is how we obtain the transition probabilities from state two.

We can obtain the transition probabilities from state 3 in a similar manner. In this case, the reliability of control A goes down from 0.96 to 0.48, and therefore the fault probability goes up to 0.52. The reliability of control B in this case is the same as in state 1. Taking this into account we can derive the probabilities of the four possible state transitions. In state four, the reliability of both controllers is down to half their nominal values (0.48 and 0.485) and using these values we can obtain the transition probabilities and the final row.

Once we have obtained the one step transition matrix, we can calculate the stationary probability vector and the average cost.

## 6.5.7 Management of Computer virus threats

### Problem definition

The IT department of your company has detected a threat in a software virus that affects the Operating System (OS) of the PCs used in the company. When the virus infects a computer, it sends emails to a set of corporate emails. Once the virus is installed in a computer, there is a procedure that can successfully remove the virus and restore the normal operation of the computer. However, in some situations, IT might not be able to restore the OS and the computer might become non-operative.

IT wants to use Markov Chains to model the status of a computer. After collecting data, they have determined that the probability that a computer is infected by the virus given that it was operating normally the day before is 0.05. The probability that the computer is non-operative for other reasons given that it was operating normally the day before is 0.1. Once a computer is infected, they always perform the virus removal procedure in the same day. They estimate that the probability that the procedure is successful is 70%, while in 30% of the cases the procedure renders the computer

non-operative. When a computer is in non-operative state, the equipment supplier always replaces it the next business day.

**a** Define the one-step probability matrix of the Markov chain. Let us identify the following states of a computer:

$s_1$  : The computer is in normal operation

$s_2$  : The computer is infected by the virus

$s_3$  : The computer is non-operative for any other reason

Given these states, the one-step transition probability matrix can be expressed as:

$$P^{(1)} = \begin{bmatrix} 0.85 & 0.05 & 0.1 \\ 0.7 & 0 & 0.3 \\ 1 & 0 & 0 \end{bmatrix}$$

Note that in the matrix the value of  $p_{ij}$  is the probability that a computer is in state  $j$  in the next day given that it is in state  $i$ . Since the provider always replaces the computer the next day,  $p_{31} = 1$ .

**b** The maintenance cost of a computer is 3€/day when it operates normally. The cost of the virus removal procedure is 30€ and the cost of replacing a computer is 1200€. Calculate the long term average costs in computer maintenance for the IT department.

Let us first define the system of equations to obtain the long term probabilities of the Markov system:

$$\pi_1 = 0.85 * \pi_1 + 0.7 * \pi_2 + \pi_3$$

$$\pi_2 = 0.05 * \pi_1$$

$$\pi_3 = 0.1 * \pi_1 + 0.3 * \pi_2$$

$$\pi_1 + \pi_2 + \pi_3 = 1$$

With this, we can calculate:

$$\pi_1 = \frac{1}{1+0.05+0.115} = 0,858$$

$$\pi_2 = 0.05 * \pi_1 = 0,0429$$

$$\pi_3 = 0.015 * \pi_1 = 0,0987$$

And once that the values of the long term probabilities are obtained, long term costs are calculated as:

$$C = 3 * \pi_1 + 30 * \pi_2 + 1200 * \pi_3$$

$$C = 122.3\text{€}$$

## BIBLIOGRAPHY

The following references have been used throughout this interactive book:

- European Union. (n.d.). Artificial intelligence. Retrieved from <https://ec.europa.eu/>
- Hagberg, A., Schult, D., & Swart, P. (2008). Networkx: A Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. Retrieved from <https://networkx.org/>
- Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., & others. (2020). Array programming with NumPy. *Nature*, 585(7825), 357-362. Retrieved from <https://numpy.org/>
- Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3), 90-95. Retrieved from <https://matplotlib.org/>
- Keen, B. A. (2018). Introduction to Operations Research with Python. Retrieved from <https://github.com/benalex-keen/Introduction-to-linear-programming>>
- Jones, E., Oliphant, T., & Peterson, P. (2001). SciPy: Open source scientific tools for Python. Retrieved from <https://scipy.org/>
- Mitchell, L. (2011). PuLP: A Linear Programming Toolkit for Python. Retrieved from <https://coin-or.github.io/pulp/>
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., & others. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825-2830. Retrieved from <https://scikit-learn.org/>
- Poler, R., Mula, J., & Díaz-Madroñero, M. (2014). *Operations Research Problems Statements and Solutions*. Springer.
- Python Software Foundation. (n.d.). Python. Retrieved from <https://www.python.org/>
- The Pandas Development Team. (2020). Pandas: Powerful data structures for data analysis in Python. Retrieved from <https://pandas.pydata.org/>



## CALL FOR CONTRIBUTIONS

If you are a teacher and would like to contribute to this online book, please do not hesitate to contact me and I will be glad to discuss how to integrate your valuable input. We are always eager to:

- **Add new exercises:** There are just not enough exercises to master the field, so if you think you can contribute with exercises to any of the sections, please reach out! We will gladly see how to incorporate them into the book.
- **Add new tutorials:** Interactive tutorials and dynamics are at the heart of this book, if you have any activity in mind that uses Python to illustrate key concepts we will be very happy to go through them with you.
- **Add new chapters:** Operations research is a vast field of applied mathematics and many techniques and methods are not covered in this book, yet. Let us work together to prepare new chapters and extend the scope of the book.