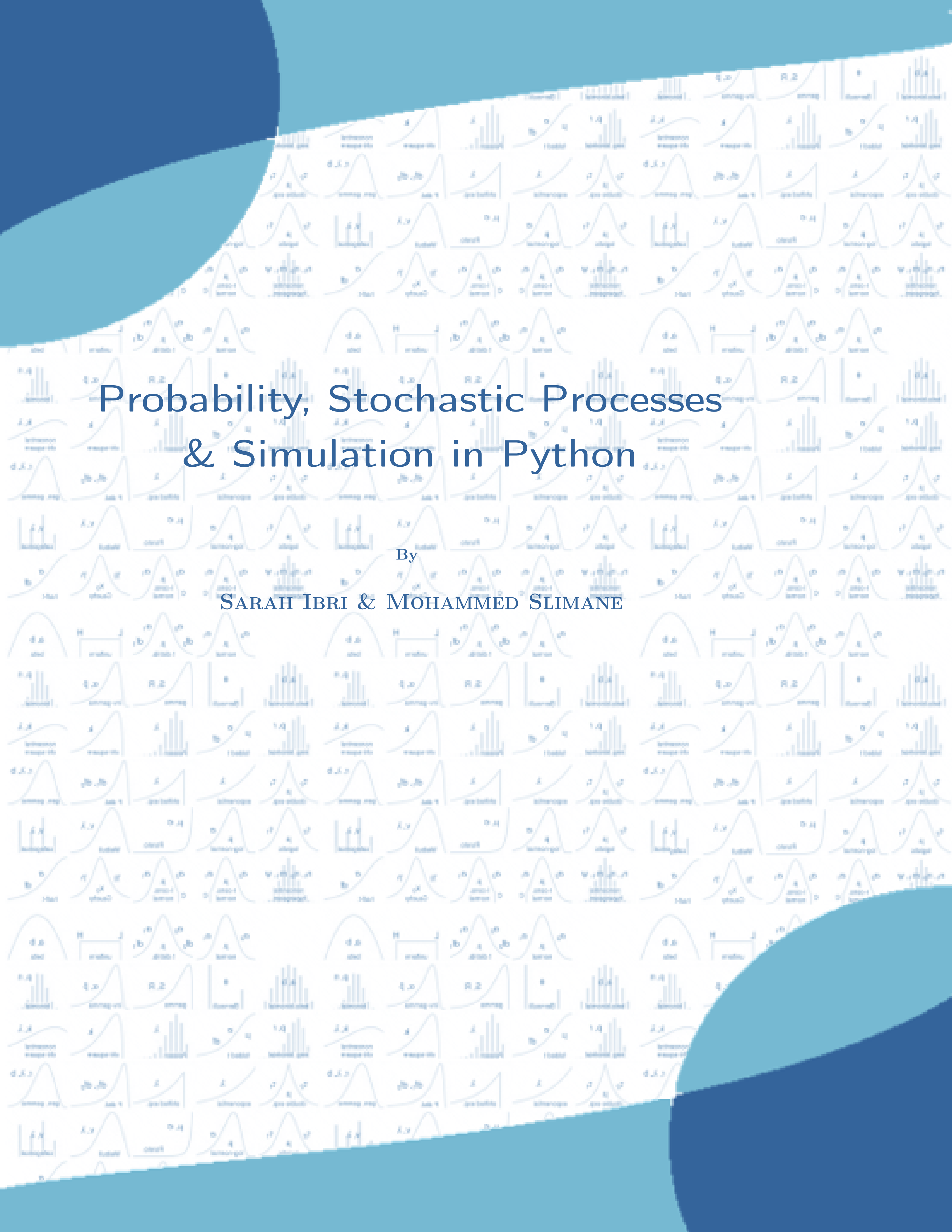
Probability Stochastic Processes and Simulation In Python en

Book · May 2022

2 authors:

Sarah Ibri
University of Science and Technology Houari Boumediene
16 PUBLICATIONS **95** CITATIONS
SEE PROFILE

Mohammed Slimane
University of Oran
3 PUBLICATIONS **5** CITATIONS
SEE PROFILE



Probability, Stochastic Processes & Simulation in Python

By

Sarah Ibri & Mohammed Slimane

Probability, Stochastic Processes & Simulation in Python

Essential Tools For Stochastic Modeling

Written By
Sarah Ibri & Mohammed Slimane

2022 Edition

Praise is to Allah by Whose grace good deeds are completed

To my parents, my younger sister

To all cancer patients may Allah heal all of them, Amen.

To people of Palestine.

Mohammed

To my parents

Sarah

Contents

Preface

In the age of artificial intelligence, robotics, intelligent systems and machine learning, which are increasingly taking up space in practically all areas; and since the emergence of intelligent applications on which most of the major IT players rely; in this great technological whirlwind, we ask the computer scientist to know, create and innovate more and more. However, the development of the necessary skills to master these recent technologies cannot be achieved without having the essential fundamentals in probability, which is one of their fundamental pillars. Mastering the foundations of this pillar has thus become essential. It is with this in mind that we have proposed this book on probability, stochastic processes and simulation, and the challenge of which is to simplify the basic concepts by the example and by the practice. This book is intended primarily for computer scientists, but can obviously be used by anyone wishing to learn the elementary concepts of probability with their practical side in order to subsequently be able to tackle other more advanced subjects such as machine learning and artificial intelligence.

The particularity of this work lies in the large number of included examples, exercises and codes meant to facilitate the assimilation of the various presented concepts. The codes are written in Python programming language and allow the reader to expose the most common and efficient APIs and libraries for probability, stochastic processes and simulation. Python being one of the most popular programming languages that beginners can learn quickly because of the simplicity of its syntax and the readability of its code. It is versatile and its packages are among the most mature of computer languages.

The first part of this work is devoted to the basics of probabilities: in the first chapter the definitions of the most elementary concepts in probabilities are presented. Chapter 2 focuses on the notion of discrete and continuous random variables by presenting the most well-known probability distributions. In the chapters of the second part, we deal respectively with stochastic processes, discrete and continuous Markov chains, as well as their applications to queuing systems. The last part is devoted to simulation. We first present the random numbers and variables' generators, then the different modules offered by Python for the simulation of more elaborated systems.

Throughout the book, the reader can find a plethora of Python code. The written codes show how to implement the different concepts on machine using the available libraries in Python, at the same time, they contribute to their improvement with new classes and functionalities. The codes are available on the following link: https://github.com/ibslim/probabook. Finally, and to help the reader better understand the usefulness of theoretical concepts, we have added several applications with their implementations at the end of the chapters.

Several individuals have provided valuable reviews of this book. We want to thank Rachid Bechar for his input, comments and corrections. We are indebted to Mohammed Tayebi for his thorough review and the precious time he spent in detecting several errors. May Allah reward you for your efforts. Of course, all errors remain the authors' responsibility.

For any comments or suggestions, the reader can contact the authors at the following address: prosimpython@gmail.com.

Part I

Probability and Random Variables

Chapter 1

Introduction to Probability

1.1 Introduction

When we are faced with uncertain events, i.e events that cannot be predicted with total certainty, we try to know the chances (how likely) they are to happen. The probability theory gives us the tools to quantify these chances, to model and to study complex uncertain situations in order to be able to make proper decisions. For instance, in artificial intelligence (AI), robots have to evaluate their environment which is mainly dynamic and uncertain before taking (the most appropriate) actions that lead them to their objectives.
In this chapter the basics of the probability theory are presented along with many examples, exercises and codes.

1.2 Random experiment

An *experiment* is a *process* or a *procedure* associated to conditions (natural or artificial) that can be repeated infinitely. It has a well defined set of possible outcomes that can be produced when it is carried out under these conditions and resulting in one and only one outcome. The outcome of the experiment is defined by the observer/experimenter (what is he interested in? what is the subject of the observation?). In case the experiment is achieved in one step, it is called *simple*, otherwise (requires many steps) it is said *compound*.
An experiment is *deterministic (predictable)* if the conditions determine uniquely the outcome of the procedure (the same conditions result always in the same outcomes with certainty). When repeating the process under the same conditions do not result, necessarily, in the same outcomes the experiment is then unpredictable and it is said *random* (the outcome depends on chance).
Modeling this type of experiments is the main subject of the probability theory. A probability model is a mathematical representation of the random aspect of experiments. It consists in a complete description of possible outcomes and a quantification of the indeterminism attribute of these phenomenons.

> **Definition 1.**
> *A random experiment is an experiment that satisfies the following two conditions:*
> *- We can absolutely determine, in advance, all the possible outcomes (there are at least two).*
> *- We cannot predict, in advance, which one of the outcomes will be obtained even if we know the experimental conditions.*

Each repetition of the experiment is called a *trial*. When the experiment is compound, a trial might combine many simple consecutive or simultaneous trials. In the remaining of this book, random experiments will be noted **R** indexed by a number.

> **Definition 2.**
> *Let $\boldsymbol{R}$ be a random experiment,*
> *The sample space of $\boldsymbol{R}$ is the set of the possible outcomes when $\boldsymbol{R}$ is performed, noted Ω.*

The outcome of the random experiment $\mathbf{R}$ will be noted ω (an element of Ω). If $\mathbf{R}$ is a random experiment *compound* of random experiments $(\mathbf{R}_i)_{1..n}$ then, its sample space Ω is the Cartesian product of the sample spaces of $\mathbf{R}_i$ (Ω_i) in the same order of their realisation.

$$\Omega = \prod_{i=1}^{n} \Omega_i$$

Each trial of $\mathbf{R}$ is an ordered tuple $(\omega_1, \omega_2, \cdots, \omega_i, \cdots, \omega_n)$.

1.2.1 Discrete and continuous sample spaces

The set of natural numbers (positive integers) is an infinite set and is a prototype of countable infinite sets. In general, an infinite set is said infinite countable if there is a one to one function (bijection, we can label/enumerate its elements with natural integers) that assigns the elements of this set to the set of natural numbers. In other words, each element of the set can be assigned to a unique natural number and reciprocally, each natural number matches a unique element of this set. For example, the set of perfect squares: $1, 4, 9, 25, ...$ is a countable infinite set. Sets that are either finite or countable infinite are said *countable*.

The set of points on a straight line and the set of real numbers between 0 and 1 are examples of infinite uncountable sets. Sets that are neither finite or countable infinite are said *uncountable*.

Like any other set, Ω can be *countable*. In this case it is called *discrete*; otherwise (if Ω is *uncountable*), it is said *continuous*.

Example 1. *Consider the following experiments:*
- $\boldsymbol{R}_1$ *: Tossing a coin.*
- $\boldsymbol{R}_1$ *is random because we cannot predict with certainty the outcome before the experiment is done.*
- *Possible outcomes of* $\boldsymbol{R}_1$ *are* ***head*** *and* ***tail*** *noted H and T respectively, $\Omega_1 = \{H, T\}$ is finite (discrete).*

- $\boldsymbol{R}_2$ *: Throwing vertically upward a ball having a mass m with an initial speed s and observing the height h where its velocity becomes zero.*
- $\boldsymbol{R}_2$ *is not random, it is deterministic because we can predict, with certainty, its outcome before its realisation by applying physics laws. (This type of experiments is useful when we have a non random phenomenon that we do not know its law, and want to confirm experimentally some hypothesis about it)*

- $\boldsymbol{R}_3$ *: Rolling a die.*
- $\boldsymbol{R}_3$ *is random because we cannot predict with certainty the outcome before the experiment is done.*
- *The possible outcomes of* $\boldsymbol{R}_3$ *are the faces of the die, $\Omega_3 = \{1, 2, 3, 4, 5, 6\}$ which is finite (discrete).*

- $\boldsymbol{R}_4$ *: Choosing randomly a letter out of the word 'statistics'.*
- $\boldsymbol{R}_4$ *is random because we cannot predict with certainty the outcome before the experiment is done.*
- *The possible outcomes of* $\boldsymbol{R}_4$ *are the letters of this word, $\Omega_4 = \{s, t, a, i, c\}$ is finite (discrete).*

- $\boldsymbol{R}_5$ *: Tossing a coin followed by rolling a die.*
- $\boldsymbol{R}_5$ *is a compound experiment of two simple random experiments* $\boldsymbol{R}_1$ *and* $\boldsymbol{R}_3$*, since we cannot predict the outcome of the component random experiments, then we cannot predict the one of* $\boldsymbol{R}_5$*, so it is random.*

- A possible outcome of $\boldsymbol{R}_5$ is a tuple that contains the outcome of $\boldsymbol{R}_1$ followed by the one of $\boldsymbol{R}_3$. The sample space Ω_5 is the Cartesian product of Ω_1 and Ω_3 which is finite (discrete).

$$\Omega_5 = \Omega_1 \times \Omega_3 = \{T, H\} \times \{1, 2, 3, 4, 5, 6\} = \{(T, 1), (T, 2), \cdots, (H, 5), (H, 6)\}$$

- $\boldsymbol{R}_6$: We toss a coin three times.
- $\boldsymbol{R}_6$ is a compound experiment of three repetitions of the same simple random experiment $\boldsymbol{R}_1$, since we cannot predict the outcome of this experiment, we cannot do it for $\boldsymbol{R}_6$, so it is random.
- A possible outcome of $\boldsymbol{R}_6$ is a tuple compound of the outcome of each repetition of $\boldsymbol{R}_1$ in order. The sample space Ω_6 is the Cartesian product of Ω_1 by itself three times which is finite (discrete).

$$\Omega_6 = \Omega_1^3 = \{T, H\}^3 = \{(T, T, T), (T, T, H), \cdots, (H, H, T), (H, H, H)\}$$

- $\boldsymbol{R}_7$: In a square C on the plane Q, we choose randomly a point inside C.
- $\boldsymbol{R}_7$ is random hypothetically.
- The possible outcomes of $\boldsymbol{R}_7$ are the points inside the square C, so $\Omega_7 = \{(x, y) \in Q | (x, y) \in C\}$. In this case, the sample space Ω_7 is infinite uncountable because $\Omega_7 \subset \mathbb{R}^2$ (continuous).

- $\boldsymbol{R}_8$: Rolling a die until getting number 6, the sample space is the set of integer numbers $\Omega_8 = \mathbb{N}$, the outcome k means that 6 is obtained at the k^{th} roll. This set is discrete (infinite countable).

- $\boldsymbol{R}_9$: measuring the elapsed time until the first occurrence of a given event, the sample space is the interval $\Omega_9 = [0, \infty[$ of real positive numbers that represent the arrival time of the event. Ω_9 is continuous (infinite uncountable).

N01 : SymPy is a Python library for symbolic computation, it offers the possibility of handling algebraic expressions.
N02 : *itertools* contains a set of tools for implementing iterative building blocks (textit iterators). It contains functions of combinatorial generators *product()*, *permutations()*, *combinations()*.

Let’s code!

The following code models some of the random experiments given in the example using *sympy*^{N01} and *itertools*^{N02} and importing *get_Omega*, *set_Product* and *set_Power* functions.

```
# Code101.py

from sympy.stats import Coin, Die
import sys;sys.path.append('../lib')
from utils import get_Omega,set_Product,set_Power

"""
In this section package sympy.stats is used
class Coin models the coin tossing experiment
class Die models the die rolling experiment
"""

Omega1 = {'P','F'}                          ;print("Omega 1 = ",Omega1)
Omega3 = set(range(1,7))                    ;print("Omega 3 = ",Omega3)
Omega5 = set_Product([Omega1,Omega3])       ;print("Omega 5 = ",Omega5)
Omega6 = set_Power(Omega1,3)                ;print("Omega 6 = ",Omega6)

print("\n Using sympy.....................................\n")
#R1
coin = Coin('Coin')
coin_omega=get_Omega(coin)
print("- R1 Toss a coin omega : ",coin_omega)

#R3
die = Die('Die')
die_omega=get_Omega(die)
print("- R3 Roll die omega : ",die_omega)

#R6
threeCoins_omega=set_Power(coin_omega,3)
print("- R6 Toss coins omega : ", threeCoins_omega)

#______________________________   Output   ______________________________________
# Omega 1 =  {’P’, ’F’}
# Omega 3 =  {1, 2, 3, 4, 5, 6}
# Omega 5 =  {(’F’, 5), (’P’, 1), (’F’, 1), (’P’, 2), (’F’, 2), (’P’, 3),
# (’F’, 3), (’F’, 6), (’P’, 4), (’F’, 4), (’P’, 5), (’P’, 6)}
# Omega 6 =  {(’P’, ’F’, ’P’), (’P’, ’P’, ’P’), (’F’, ’P’, ’P’), (’P’, ’P’, ’F’),
# (’P’, ’F’, ’F’), (’F’, ’P’, ’F’), (’F’, ’F’, ’P’), (’F’, ’F’, ’F’)}

#  Using sympy.....................................

# - R1 Toss a coin omega :  {H, T}
# - R3 Roll die omega :  {1, 2, 3, 4, 5, 6}
# - R6 Toss coins omega :  {(T, T, T), (H, T, T), (H, H, H), (T, H, T),
#                           (T, T, H), (H, T, H), (T, H, H), (H, H, T)}
```

Let’s code!

The following code *utils* is a custom helper module (that can be found in the package "lib"). It contains many functions shared with the following codes. The finite maps are modeled in Python with *dict* data structure which is a set of (key,value) pairs such that the key is the antecedent element and the value is its image. "utils.py" is in a folder called "lib" and will be imported using the following statement: "import sys;sys.path.append('../lib')" as it is not a standard or installed module in Python.

```
# utils.py

import matplotlib.pyplot as plt
from sympy.stats import density
from itertools import accumulate,product
from fractions import Fraction

#--------------------------------------------------------------------------------
# Fraction is a class that respresents rational numbers
Fracstr = lambda p,q : str(Fraction(p,q))

# To format the dictionary values to 2 decimal floats
get_round_dic = lambda dic:{k:round(float(v),2) for k,v in dic.items()}

# Returns the sample space (SS) of the random experiment (RE)
get_Omega = lambda re: set(density(re).dict.keys())

# power: returns cartesian product of a set with itself n times
set_Power = lambda omega,n: set(product(omega,repeat=n))
```

```
# product: returns the cartesian product of the given sets
set_Product = lambda omegas: set(product(*omegas))

# filter: selects elements from a set based on some criterion
set_Filter = lambda predicate,collection:set(filter(predicate,collection))

# Pe: returns the probabbility of an event from equally likely SS
Pe  = lambda Omega, Event : Fraction(len(Event), len(Omega))

# Pde: returns the probabbility of an elementary event from equally likely SS
Pde = lambda Omega : { omega : Fracstr(1,len(Omega)) for omega in Omega}

# Pgiven: returns conditional probability of A given B
Pgiven = lambda EventB,EventA :\
    Fraction(len(set(EventA) & set(EventB)), len(EventA))
```

1.2.2 Measurable space and events

Definition 3.
Let $\boldsymbol{R}$ *be a random experiment having the sample space* Ω*,*
1- A σ-Algebra $\mathcal{F}$ *of* Ω *is a collection of subsets of* Ω *having the following properties :*
i- $\Omega \in \mathcal{F}$
ii- $\mathcal{F}$ *is closed under the complement :* $\forall E \in \mathcal{F} \Rightarrow E^c \in \mathcal{F}$
iii- $\mathcal{F}$ *is closed under the countable union of subsets of* Ω *:*

$$\{E_i\}_{i\geq 0} \subset \mathcal{F} \Rightarrow \bigcup_{i\geq 0} E_i \in \mathcal{F}$$

- The couple $(\Omega, \mathcal{F})$ *is called the measurable space associated with* $\boldsymbol{R}$*.*

Let $(\Omega, \mathcal{F})$ *be a measurable space associated with the random experiment* $\boldsymbol{R}$*.*
2- An event E *is an element of* $\mathcal{F}$ *(subset of the sample space* Ω*).*

$$E \text{ is an event} \iff E \in \mathcal{F}, (\ E \subset \Omega\ )$$

3- A simple event E *is an event that contains one element of* Ω *(singleton set).*

$$E \text{ is a simple event} \iff E \in \mathcal{F} \text{ and } |E| = 1(\ \exists \omega \in \Omega : E = \{\omega\}\ )$$

Let Ω be a sample space, its smallest σ-algebra contains only Ω and $\varnothing$ (check that it satisfies the proprieties i,ii and iii), and is called *trivial* σ-algebra.

2^Ω is the power set of Ω (the set of all subsets of Ω), it satisfies the properties of a σ-algebra, and is called *maximal σ-algebra*.

We say that an event E *occurs* when the result of the experiment belongs to E.

Example 2. *Let consider the random experiments of example 1.1 :*
1- For all $\boldsymbol{R}_i$ *and* Ω_i*, for* $i \in \{1, 3, 4, 5, 6\}$*, we have :*
a- $(\Omega_i, 2^{\Omega_i})$ *is a measurable space (maximal σ-algebra, all the subsets of* Ω *are events).*
b- $(\Omega_i, \{\varnothing, \Omega_i\})$ *is a measurable space (trivial σ-algebra, only these two subsets of* Ω *are events).*

2- For $\boldsymbol{R}_1$ *and* $\Omega_1 = \{H, T\}$*, we have :*
a- $(\Omega_1, \{\varnothing, \{H\}, \{T\}, \{H, T\}\})$ *(maximal σ-algebra).*
b- $(\Omega_1, \{\varnothing, \{H, T\}\})$ *(trivial σ-algebra).*
c- $(\Omega_1, \{\varnothing, \{H\}, \{H, T\}\})$ *is not a measurable space because the complement of* $\{H\}$ $(\{T\}) \notin \mathcal{F}$*.*

3- For $\boldsymbol{R}_3$ *and* $\Omega_3 = \{1, 2, 3, 4, 5, 6\}$*, we have :*
a- $(\Omega_3, \{\varnothing, \{1\}, \{2, 3, 4, 5, 6\}, \{1, 2, 3, 4, 5, 6\}\})$ *is a measurable space.*

b- $(\Omega_3, \{\varnothing, \{1\}, \{2,3,4,5,6\}, \{1,5\}, \{2,3,5\}, \{1,2,3,4,5,6\}\})$ *is not a measurable space because the property ii is not satisfied* $(\{1,5\} \cup \{2,3,5\}$ *is not element of* $\mathcal{F})$*.*

4- For $\boldsymbol{R}_6$ *and* $\Omega_6 = \{H,T\}^3$*, we have :*
a- $(\Omega_6, \{\varnothing, \{(H,H,H)\}, \Omega_6/\{(H,H,H)\}, \Omega_6\})$ *is a measurable space.*

Like any set, in addition to its explicit form (when enumerating its elements), events are described by a property that characterize them implicitly. This description is usually a linguistic expression that is made explicit to allow doing calculation in the context of the probability theory. As an example, for $\mathbf{R}_3$ (a die rolling), the event E described by the following property: "the obtained outcome is an even number" is explicitly described by $E = \{2,4,6\}$. The σ-algebra of the events is a set of subsets, so all the sets operations are possible on the σ-algebra. Within the scope of this theory, we use a specific terminology (probabilistic interpretation) of these operations.

Set operation	Notation	Operation	occurs iff
Complement	E^c	negation	E does not occur
Union	$E \cup F$	or	E or F occurs
Intersection	$E \cap F$	and	E and F occur simultaneously
Difference	$E - F$	only of	E occurs, but not F
Symmetric difference	$E \Delta E$	not simultaneously	E or F doesn't occur simultaneously

To handle events, we use set algebra (associative and distributive properties, Morgan relations, etc).

If $E \cap F = \varnothing$ then the two events E and F are *disjoints*.

Two events E and F are *mutually exclusives* if the occurrence of one excludes the occurrence of the other. As a generalization of the previous result, events $E_1, E_2, \cdots, E_n$ are *mutually exclusives* if the occurrence of one excludes the occurrence of the others

Let's code!

The following code presents events' operations.

```
#Code102.py

from sympy.stats import Die
import sys;sys.path.append('../lib')
from utils import get_Omega,set_Filter

print("Events manipulation (operations)")

abc_Omega = {1,2,3,4,5,6}    ;print("Sample space Omega      : ",abc_Omega)
A = {2,5}                    ;print("Event A                 : ", A)
B = {3,4,5}                  ;print("Event B                 : ",B)
isEvent = A < abc_Omega      ;print("is A event of Omega     : ", isEvent)
a_bar = abc_Omega - A        ;print("Complement of A         : ", a_bar)
C = A & B                    ;print("Union of A and B        : ", C)
D = A | B                    ;print("Intersection of A and B : ", D)
E = A - B                    ;print("Difference of A and B   : ", E)
F = A ^ B                    ;print("Symetric Diff A and B   : ", F)

# isEvenNumber: predicate to check if a number is even
isEvenNumber = lambda nb : nb % 2 == 0

die_omega=get_Omega(Die('Die'))
evenNb =set_Filter(isEvenNumber,die_omega)
print("Event A even numbers  : ", evenNb)

evenNbComplement = set(die_omega) - set(evenNb)
print("Event Complement of A : ", evenNbComplement)

#______________________________   Output   ______________________________________
# Events manipulation (operations)
# Sample space Omega      :  {1, 2, 3, 4, 5, 6}
# Event A                 :  {2, 5}
# Event B                 :  {3, 4, 5}
# is A event of Omega      :  True
# Complement of A          :  {1, 3, 4, 6}
# Union of A and B         :  {5}
# Intersection of A and B :  {2, 3, 4, 5}
# Difference of A and B    :  {2}
# Symetric Diff A and B    :  {2, 3, 4}
# Event A even numbers     :  {2, 4, 6}
# Event Complement of A    :  {1, 3, 5}
```

Let's code!

The following code implements R_6 with $E = (T,T,H),(T,H,T),(H,T,T)$ is the event of getting two tails.

```
#Code103.py

from sympy import Symbol
from sympy.stats import Coin
import sys;sys.path.append('../lib')
from utils import get_Omega,set_Power,set_Filter

# class Symbol sets symbols for algebric expressions. "T" for coin tail
# twoTails: checks if the outcome has two tails
twoTails = lambda triple : triple.count(Symbol("T") )==2

coin_omega=get_Omega(Coin('Coin'))
threeCoins_omega = set_Power(coin_omega,3)
has2Tails = set_Filter(twoTails,threeCoins_omega)
print("Event with two tails only : ", has2Tails)

#______________________________   Output  _______________________________________
# Event with two tails only :  {(T, T, H), (H, T, T), (T, H, T)}
```

1.3 Probability

Probability is a function that assigns a real number between 0 and 1 (included) to a given event. This number represents the chance that this event occurs. The probability axioms are mathematical rules that the probability function has to satisfy.

> **Definition 4.**
> *Let $(\Omega, \mathcal{F})$ be the measurable space associated to a random experiment R.*
> *A probability P is a function of $\mathcal{F}$ in the interval $[0,1]$.*
>
> $$P : \mathcal{F} \mapsto [0,1]$$
>
> *that satisfies the following axioms:*
> - ***normality** : $P(\Omega) = 1$ (a norm on Ω)*
> - *σ-**Infinite-additivity** : for all family of events mutually exclusive, the probability of their union is the sum of the probabilities of each of them.*
>
> $$\{E_i\}_{i>0} \text{ and } \forall i \neq j, E_i \cap E_j = \varnothing \implies P(\bigcup_{i>0} E_i) = \sum_{i>0} P(E_i)$$
>
> *We call the triplet $(\Omega, \mathcal{F}, P)$ a probability measurable space.*

In the case of countable spaces, probabilities can be assigned to each subset of the space. However, in uncountable spaces, "uncommon" subsets can be built up and for which we cannot assign a probability. So, for some purely mathematical reasons, a probability can be given only for well defined subsets when the space is uncountable (the axiom σ-**Infinite-additivity** is not always satisfied, see [11]). For this reason the probability is defined for σ-algebra on a sample space with a particular structure that allows the σ-additivity to be satisfied.

Assigning probability to events is actually a transition from a concrete context to a mathematical model. For a useful choice of the probabilities given to events, the assignment has to result in an "appropriate" model for real situations. Two main approaches exist: the *frequentist approach* and the *subjective approach (Bayesian)*. In the *frequentist approach*, probabilities are assigned to the results of a physical experiment that can be repeated many times under identical conditions. This is the case, for example, of a die rolling, these probabilities can be determined experimentally. In the *subjective approach (Bayesian)*, the word probability is synonym of *plausibility*, the probability is defined as the belief degree that someone has about the occurrence of some event. This is the case, for example, of the chances that someone gives for a team to win a game. This approach is limited to experiments that cannot be repeated.

1.3.1 Probability properties

Proposition 1.
Let $(\Omega, \mathcal{F}, P)$ be the probability measurable space, the following properties are satisfied:

*1- **Finite-Additivity** : let $\{E_i\}_{i\in\{1..n\}}$ be a finite sequence of mutually exclusive events, then :*

$$P(\bigcup_{i=1}^{n} E_i) = \sum_{i=1}^{n} P(E_i)$$

*2- **Complement rule** : $\forall E \in \mathcal{F},\ P(E^c) = 1 - P(E)$.*

*3- **Union rule** : $\forall E, F \in \mathcal{F},\ P(E \cup F) = P(E) + P(F) - P(E \cap F)$*
*4- **Inclusion-exclusion formula** : let $\{E_i\}_{i\in\{1..n\}}$ be a finite sequence of events,*

$$P(\cup_{i=1}^{n} E_i) = \sum_{i=1}^{n} P(E_i) - \sum_{i<j\le n} P(E_i \cap E_j) + \sum_{i<j<k\le n} P(E_i \cap E_j \cap E_k) ... (-1)^{n-1} P(E_1 \cap ... \cap E_n)$$

*5- **Monotonicity** : $\forall E, F \in \mathcal{F},\ E \subset F \Rightarrow P(E) \le P(F)$.*

1- To demonstrate $P(\varnothing) = 0$, we use the infinite-additivity axiom with $E_i = \varnothing, \forall i$. We have $P(\varnothing) = \sum_{i=1}^{\infty} a_i$ such that $a_i = P(\varnothing), \forall i$, this implies $P(\varnothing) = 0$.

Let $\{E_i\}_{i\in\{1..n\}}$ be a finite sequence of mutually exclusive events. We consider the following infinite sequence :

$\{A_i\}_{i>0}$ s.t $A_i = E_i$, for $i \le n$ and $A_i = \varnothing$, for $i > n$, so:

$$\bigcup_{i=1}^{\infty} A_i = \bigcup_{i=1}^{n} E_i$$

$$P(\bigcup_{i=1}^{n} E_i) = P(\bigcup_{i=1}^{\infty} A_i) = \sum_{i=1}^{n} P(A_i) + \sum_{i=n+1}^{\infty} P(A_i) = \sum_{i=1}^{n} P(E_i)$$

2- The second property results from the property of the **σ-finite-Additivity** :
Let E be an event, and E^c its complement.
Since E and E^c are mutually exclusives.
And since $E \cup E^c = \Omega$
we have $1 = P(\Omega) = P(E \cup E^c) = P(E) + P(E^c)$,
So $P(E^c) = 1 - P(E)$.

4- By induction, we assume that the property is true. Let : $R_n = \cup_{i=1}^{n} E_i$

$$P(R_n) = \sum_{i\le n} P(E_i) - \sum_{i<j\le n} P(E_i \cap E_j) + \sum_{i<j<k\le n} P(E_i \cap E_j \cap E_k) + ... + (-1)^{n-1} P(E_1 \cap ... \cap E_n)$$

We show that the property is true for R_{n+1}:

$$P(R_{n+1}) = P(\cup_{i=1}^{n+1} E_i) = P(E_{n+1} \cup (\cup_{i=1}^{n} E_i)) = P(E_{n+1}) + P(R_n) - P(E_{n+1} \cap R_n)$$

$E_{n+1} \cap R_n = \cup_{i=1}^{n}(E_{n+1} \cap E_i)$

$$P(E_{n+1} \cap R_n) = \sum_{i<n+1} P(E_i \cap E_{n+1}) - \sum_{i<j<n+1} P(E_i \cap E_j \cap E_{n+1}) + ... + (-1)^{n-1} P(E_1 \cap ... \cap E_{n+1})$$

$$P(R_{n+1}) = \sum_{i \leq n+1} P(E_i) - \sum_{i<j \leq n+1} P(E_i \cap E_j) + \sum_{i<j<k \leq n+1} P(E_i \cap E_j \cap E_k) + ... + (-1)^n P(E_1 \cap ... \cap E_{n+1})$$

1.3.2 Equiprobability

Let $(\Omega, \mathcal{F}, P)$ be a probability measurable space : if Ω is finite ($|\Omega| = n < \infty$) then, $\Omega = \{\omega_1, \omega_2, \cdots \omega_n\}$ where ω_i is a simple event for all $1 \leq i \leq n$ in this case, we take $\mathcal{F} = 2^{\Omega}$ the discrete σ-algebra of Ω and the probability P is defined by its restriction p on Ω, $p : \Omega \to [0, 1]$ such that $p(\omega_i) = P(\{\omega_i\})$ noted p_i. For the event $E = \{\omega_{i_1}, \omega_{i_2}, \cdots, \omega_{i_h}\}$, we have :

$$P(\{\omega_{i_1}, \omega_{i_2}, \cdots, \omega_{i_h}\}) = \sum_{k=1}^{h} p_{i_k}$$

Definition 5. ***Equiprobability***
Let $(\Omega, \mathcal{F}, P)$ be a probability measurable space :
Simple events are said equally likely if Ω is finite and they have the same probability.

$$\forall \omega_i \in \Omega, p(\omega_i) = \frac{1}{|\Omega|}$$

The probability of an event E in this space is :

$$\forall E \in \mathcal{F}, P(E) = \frac{|E|}{|\Omega|}$$

In this case, the calculation of probabilities comes down to an enumeration (see appendix A). In random experiments with a finite probability space, it is natural to assume the outcomes to be equally likely (based on belief). This model is called *the classic probability model* or *Laplace model*.

Let's code!

We use Pe from Lib to compute the equally likely probability of an event.

```
#Code104.py

from functools import partial
import sys;sys.path.append('../lib')
from utils import Pe, Pde

# partial: returns a new function from the given one by setting the specified parameters
Omega = {1,2,3,4,5,6}          ;print("Omega                : ", Omega)
A     = {2,5}                  ;print("Event A              : ", A)
Pe    = partial(Pe,Omega)
pe_A  = Pe(A)                  ;print("Probability of A     : ", pe_A)
Pde   = Pde(Omega)             ;print("Probability App      : ", Pde)

#_______________________________  Output  _______________________________________
# Omega               :  {1, 2, 3, 4, 5, 6}
# Event A             :  {2, 5}
# Probability of A    : 1/3
# Probability App     : {1: '1/6', 2: '1/6', 3: '1/6', 4: '1/6', 5: '1/6', 6: '1/6'}
```

Example 3. *Three regular dice are rolled. What is the probability that the outcome of the 3^{rd} die is equal to the sum of the outcomes of the two others.*

Solution
The measurable space is:
1- $\Omega = \{1, 2, 3, 4, 5, 6\}^3 = \{(i, j, k) : i, j, k = 1..6\}$ such that i is the outcome of the 1^{st} die, j the

outcome of the 2^{nd}, and k is the outcome of the 3^{rd} one. $|\Omega| = 6 \times 6 \times 6 = 216$
2- The σ-algebra is 2^Ω
3- if we assume the outcomes are equally likely then, $\forall \omega_i \in \Omega, p(\omega_i) = \frac{1}{216}$.

For this probability measurable space $(\Omega, 2^\Omega, P)$, the event E that 'the outcome of the 3^{rd} die is equal to the sum of the outcomes of the two others' has the following outcomes:

$$\begin{aligned}E =&\{(1,1,2),(1,2,3),(2,1,3),(1,3,4),(3,1,4),(1,4,5),(4,1,5),(1,5,6),(5,1,6),(2,2,4),\\ &(2,3,5),(3,2,5),(2,4,6),(4,2,6),(3,3,6)\}\end{aligned}$$

$$P(E) = \frac{|E|}{|\Omega|} = \frac{15}{216} = \frac{5}{72}$$

Let's code!

The code of the previous example.

```
#Code105.py

from sympy.stats import Die
import sys;sys.path.append('../lib')
from utils import get_Omega, Pe, Pde, set_Filter, set_Power

die_Omega=get_Omega(Die('Die'))
threeDices_Omega = set_Power(die_Omega , 3)

print("- Drop 3 Dices Omega : ", list(threeDices_Omega)[:4])
print("- Omega length : "      , len(threeDices_Omega))

P_3Dices_Omega = Pde(threeDices_Omega)
print("- Proba mapping : "     , list(P_3Dices_Omega.items())[:4])

# ev_Property: checks if the sum of the two first elements equals the 3rd one
ev_Property = lambda omega : omega[0] + omega[1] == omega[2]
E   = set_Filter(ev_Property, threeDices_Omega) ; print("- E Length: ",len(E))
p_E = Pe(threeDices_Omega, E)                   ; print("- Probability of E : ",p_E)

#______________________________    Output  ______________________________________
# - Drop 3 Dices Omega :  [(4, 2, 2), (1, 4, 4), (2, 2, 4), (5, 5, 1)]
# - Omega length :  216
# - Proba mapping :  [((4, 2, 2), '1/216'), ((1, 4, 4), '1/216'), ((2, 2, 4), '1/216'), ((5, 5, 1)
    , '1/216')]
# - E Length:  15
# - Probability of E :  5/72
```

Example 4. *In a café, customers pay a cup of tea 30DA, 20DA or 10DA. To know how much a customer has to pay, he rolls a die. If the outcome is 1, he pays 10DA, if it is 2,3,4 or 5 he pays 20DA otherwise, he pays 30 DA. Two friends go to this café, what is the probability that the sum of money paid by them together does not exceed 30DA?*

Solution
1- The sample space of this experiment is $\Omega = \{1,2,3,4,5,6\}^2$
2- Since Ω is finite, then we choose 2^Ω as σ-algebra
3- We assume that the elements of the sample space are equally likely, so $\forall \omega_i \in \Omega, p(\omega_i) = 1/|\Omega| = 1/36$.
For this sample space $(\Omega, 2^\Omega, P)$, let consider the following events :
$E_1 = \{$ *the first client pays 10 and the second pays 10* $\} = \{(i,j)|i = 1$ *and* $j = 1\}$
$E_2 = \{$ *the first client pays 10 and the second pays 20* $\} = \{(i,j)|i = 1$ *and* $j \in \{2,3,4,5\}\}$
$E_3 = \{$ *the first client pays 20 and the second pays 10* $\} = \{(i,j)|i \in \{2,3,4,5\}$ *et* $j = 1\}$
The event E that the sum of money paid by the two friends together does not exceed 30DA can

be written as the union of these three disjoint events:

$$\begin{aligned} E &= E_1 \cup E_2 \cup E_3 \\ P(E) &= P(E_1 \cup E_2 \cup E_3) \\ &= P(E_1) + P(E_2) + P(E_3) \\ &= \frac{|E_1|}{|\Omega|} + \frac{|E_2|}{|\Omega|} + \frac{|E_3|}{|\Omega|} \\ &= \frac{1}{36} + \frac{4}{36} + \frac{4}{36} = \frac{9}{36} = \frac{1}{4} \end{aligned}$$

Example 5. *Let consider the experiment of tossing two coins. E is the event that the first one gives 'head' and F the event that the second gives 'head'. What is the probability that the first or the second gives 'head'?*

Solution
1- The sample space is : $\Omega = \{H, T\}^2 = \{(H, H), (H, T), (T, H), (T, T)\}$
2- The σ-algebra: $\mathcal{F} = 2^\Omega$
3- Elements of Ω equally likely.
For this measurable space $(\Omega, 2^\Omega, P)$, we consider the following events :

$$\begin{aligned} E =& \{ \textit{ the first coin gives head } \} \\ F =& \{ \textit{ the second coin gives head } \} \\ K =& \{ \textit{ the first or the second coin give head } \} \\ H : head, T : tail. E =& \{(H, H), (H, T)\}, \quad P(E) = \frac{1}{4} + \frac{1}{4} = \frac{1}{2} \\ F =& \{(H, H), (T, H)\}, \quad P(F) = \frac{1}{4} + \frac{1}{4} = \frac{1}{2} \\ K =& E \cup F, \quad P(K) = P(E \cup F) = P(E) + P(F) - P(E \cap F) \\ & \textit{Since } P(E \cap F) = P(\{(H, H)\}) = \frac{1}{4} \\ P(K) =& \frac{1}{2} + \frac{1}{2} - \frac{1}{4} = \frac{3}{4} \end{aligned}$$

Let’s code!

The code of the previous example.

```python
#Code106.py

from sympy.stats import Coin
from sympy import Symbol
from functools import partial
import sys;sys.path.append(’../lib’)
from utils import get_Omega, set_Power, set_Filter, Pde, Pe

coin_Omega=get_Omega(Coin(’Coin’))
twoCoins_Omega = set_Power(coin_Omega, 2)
print("- Toss 2 Coins Omega : ",twoCoins_Omega)

p_2Coins_Omega = Pde(twoCoins_Omega)
print("- Proba map : ",p_2Coins_Omega)

# ev_Property: checks if the ith element of the outcome equals val
ev_Property = lambda i, val, omega : omega[i] == val

# set_Filter: filters the given SS using ev_Property with the specified parameters
E   = set_Filter(partial(ev_Property,0, Symbol("T")) , twoCoins_Omega)
p_E = Pe(twoCoins_Omega, E)
print("- Event E : ",E, " Probability : ",p_E)

F   = set_Filter(partial(ev_Property,1, Symbol("T")) , twoCoins_Omega)
p_F = Pe(twoCoins_Omega, F)
print("- Event F : ",F, " Probabulity : ",p_F)
```

```
EF   = E & F
p_EF = Pe(twoCoins_Omega, EF)
print("- Event E et F : ",EF, " Probability : ", p_EF)

K   = E | F
p_K = p_E + p_F - p_EF
print("- Event K (E or F) : ", K, " Probability : ", p_K)

#______________________________   Output  ______________________________________
# - Toss 2 Coins Omega :  {(H, T), (T, T), (T, H), (H, H)}
# - Proba map :  {(H, T): ’1/4’, (T, T): ’1/4’, (T, H): ’1/4’, (H, H): ’1/4’}
# - Event E :  {(T, H), (T, T)}  Probability :  1/2
# - Event F :  {(T, T), (H, T)}  Probabulity :  1/2
# - Event E et F :  {(T, T)}  Probability :  1/4
# - Event K (E or F) :  {(H, T), (T, H), (T, T)}  Probability :  3/4
```

Example 6. *We want to know the probability that in a class of n kids (not twins) at least two kids have the same birthday. We consider that a year has 365 days (we ignore the leap years) and that all days are equally likely.*

Solution
A class of n non twin kids is represented as n-tuple of birthdays of these kids (d_i in the set $Y = \{d_1, d_2, \cdots d_{365}\}$ is the birthday of the i^{th} kid). The random experiment consists in choosing a list of n elements of Y (n-tuple) as an outcome that represents a class.

1- The sample space : $\Omega = Y^n = \{(\omega_{i_1}, \omega_{i_2}, \cdots, \omega_{i_n})\}_{1 \leq i_k \leq 365}$ the cardinality is $|\Omega| = 365^n$

2- The σ-algebra $\mathcal{F} = 2^\Omega$

3- The elements of Ω are equally likely.

For this measurable space $(\Omega, 2^\Omega, P)$, we have the event :
E : In a class of n kids (not twins) at least two kids have the same birthday. (there is a birthday that recurs more than once in the n-tuple).
E^c : is the complement of E , having a class of n kids all, with different birthdays; its cardinality is the number arrangements (see appendix A) of n elements out of 365.

$$P(E) = 1 - P(E^c) = 1 - \frac{|E^c|}{|\Omega|} = 1 - \frac{A^n_{365}}{365^n} = 1 - \frac{365!}{(365-n)!365^n} = f(n)$$

Figure 1.4 shows the probability function of n. We can see that for $n = 23$, $P(E) = 0.50$, for a class of 23 kids, there is 50% chance that at least two kids have the same birthday which is much higher than our intuition. With $n = 41$, $p = 90\%$ and reaches $p = 99\%$ for $n = 57$.

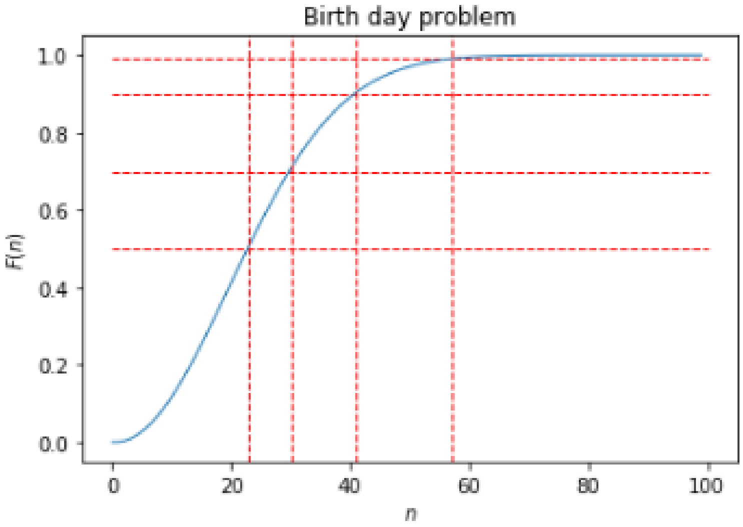


Figure 1.1: Probability function of n

N01 : Numpy is a Python library for numerical computation offering the operations to manipulate matrices, vectors, and the usual numerical methods.

Let's code!

The code of the previous example.

```
#Code107.py

# function prod of numpy is used to compute the product of the vector's elements
def get_birth(n, L):
    import numpy as np

    F = lambda n : 1 - np.prod([(365-i)/365 for i in range(n)])
    E = [F(i) for i in range(n)]
    for i in L :print("F(",i,"):",F(i))
    return E

# plotter: plots the function associated with the birthday probability example
def plotter(n,E):
    import matplotlib.pyplot as plt

    plt.plot(range(n), E, linewidth=1)
    xpos = [23, 30, 41, 57]
    for xc in xpos: plt.axvline(x=xc, color='r', linewidth=1, linestyle='--')

    ypos = [0.5, 0.7, 0.9, 0.99]
    for yc in ypos: plt.hlines(y=yc, xmin=0, xmax=100, linewidth=1, color='r',
                               linestyle='--')

    plt.xlabel('$n$');  plt.ylabel(r'$F(n)$'); plt.title('Birth day problem')
    plt.show()

n = 100
L = [23,30,41,57]
plotter(n,get_birth(n,L))
#______________________________   Output   ______________________________________
# F(23): 0.5072972343239857
# F(30): 0.7063162427192688
# F(41): 0.9031516114817354
# F(57): 0.9901224593411699
```

Application

Digital file identifier Practical applications of this problem include methods that associate digital identifiers with files. So if we have $n$ files, we have to take K (the size of the set of identifiers) large enough that the probability that two files have the same identifier is very low.

The cryptographic hash function In cryptography, the hash function h is used in conjunction with the electronic signature to generate a digital fingerprint of information that will be signed and attached to this information to prove its authenticity and to guarantee the objective of integrity and non-repudiation. It associates with any message m of arbitrary size a key message k of fixed size identifying m. h is defined:

$$h : \{0,1\}^{\infty} \to \{0,1\}^{n}$$

Based on the definition of h, the birthday attack exploits probabilistic properties to violate the principle of integrity by changing the message m associated with a key k. As the message size is arbitrary and $n$ is fixed then it is possible to find two different messages which have the same key by h. We say that a collision has been found and the associated problem is called the collision problem. By giving h, the goal of this attack is to find two messages m_1 and m_2 such that $m_1 \neq m_2$ and $h(m_1) = h(m_2)$. This pair is called the collision.

1.4 Conditional Probabilities

Let A be an event of a random experiment R. $P(A)$ is the chance of occurrence of A before the experiment happens. $P(A)$ is called the unconditional probability (or "a priori"). Assuming that we know that in this experiment an event B has occurred, but we don't know if A has occurred or not. Knowing that B has occurred, the sample space Ω is replaced by the sample space of B. The probability of occurrence of A given that B has occurred is called the the conditional probability.

> **Definition 6.** ***Conditional Probability***
> *Let $(\Omega, \mathcal{F}, P)$ be the probability measurable space associated with a random experiment R and the event B having non null probability.*
> *The application P_B defined from $(\Omega, \mathcal{F})$ to the interval $[0,1]$ as:*
>
> $$\forall A \in \mathcal{F}, \quad P_B(A) = \frac{P(A \cap B)}{P(B)}$$
>
> *is a probability under the σ-algebra of B. (satisfies the probability axioms).*
> *We call P_B the conditional probability given B and noted $P(.|B)$. If Ω is finite, then*
>
> $$\forall A \in \mathcal{F}, \quad P(A|B) = \frac{|A \cap B|}{|B|}$$

> **Proposition 2.** ***Conditional probability properties***
> *Since $P(.|A)$ is a probability on the restriction of the sample space knowing the occurrence of A, then it has the same properties as a regular probability.*

Example 7. *We roll two dice such that each of the possible 36 outcomes is equally likely. We notice that the first die gave 4; knowing that, what is the probability that the sum of the two dice equals 6?*

Solution
Knowing the information that '4 is the outcome of the first die', the set of possible outcomes of this experiment (the new sample space) is :
$\Omega = \{(4,1),(4,2),(4,3),(4,4),(4,5),(4,6)\}$ *associated to its uniform probability.*
Let E be the event 'the sum of the two dice is 6' and F the event that 'the first die is 4'.
$E \cap F = \{(4,2)\}$

$$P(E|F) = \frac{P(E \cap F)}{P(F)} = \frac{|E \cap F|}{|F|} = \frac{1/36}{6/36} = \frac{1}{6}$$

This is due to the fact that if F occurs, then E will occur if the outcome is in E and F: $E \cap F$. And, because we know that F has occurred, then F becomes the new sample space, so the probability that $E \cap F$ occurs equals the probability of $E \cap F$ in relation to the probability of F

Let's code!

The code of the previous example.

```
#Code108.py

import sys;sys.path.append('../lib')
from utils import set_Power,set_Filter, Pe, Pgiven

Omega = set_Power(range(1,7), 2)

# set_Filter: filters the outcomes having the first element equals 4.
F= set_Filter(lambda a: a[0] == 4,Omega)
print("F given Event : ", F, ", P(F)=", Pe(Omega,F))

# set_Filter: filters the outcomes having the sum of the 1st and the 2nd elements equals 6.
E= set_Filter(lambda a : a[0] + a[1] == 6,Omega)
print("E event : ",E, ", P(E)=", Pe(Omega,E))

EF = E & F
print("E and F event : ",EF, ", P(E & F)=", Pe(Omega,EF))

p_given_F = Pgiven(E,F)
print("Probability of E given F : ", p_given_F)

#______________________________   Output  ______________________________________
# F given Event :  {(4, 6), (4, 5), (4, 4), (4, 3), (4, 2), (4, 1)} , P(F)= 1/6
# E event :  {(5, 1), (3, 3), (1, 5), (4, 2), (2, 4)} , P(E)= 5/36
# E and F event :  {(4, 2)} , P(E & F)= 1/36
# Probability of E given F :  1/6
```

Example 8. *Let U be a box that contains 4 red balls R and 6 green ones G. Two balls are drawn from U in a blind way one after another and without replacement. What is the probability that the second drawn ball is red given that the first drawn one is red?*

Solution
$R = \{r_1, r_2, r_3, r_4\}$
$G = \{g_1, g_2, g_3, g_4, g_5, g_6\}$
$U = R \cup G$
$\Omega = \{(b_1, b_2) \in U^2 \ and \ b_1 \neq b_2\}, \ |\Omega| = |U|(|U|-1) = 10 \times 9 = 90$
$A = \{ \ 1^{th} \ red \ ball \ \} = \{(b_1, b_2) \in \Omega \ and \ b_1 \in R\}, \ |A| = |R|(|U|-1) = 4 \times 9 = 36$
$P(A) = \frac{|A|}{|\Omega|} = \frac{4\times 9}{10\times 9} = \frac{4}{10}$
$B = \{ \ 2^{\,nd} \ red \ ball \ \} = \{(b_1, b_2) \in \Omega \ and \ b_2 \in R\}$

$$A \cap B = \{(b_1, b_2) \in \Omega, b_1 \in R \ and \ b_2 \in R\}, \ |A \cap B| = |R|(|R|-1) = 4 \times 3 = 12$$

$P(B \cap A) = \frac{|A \cap B|}{|\Omega|} = \frac{12}{90}$
Given that A has occurred, we have a new sample space :
$\Omega_A = A$*, and the associated σ-algebra* 2^{Ω_A}.
In this sample space, consider the event B_A 'the second ball is red' :
$B_A = \{(r_i, r_j) \in \Omega_A, r_j \in R\} = A \cap B, \ |B_A| = |A \cap B|$

$$P(B|A) = \frac{|B_A|}{|\Omega_A|} = \frac{|A \cap B|}{|A|} = \frac{|\Omega|}{|A|} \times \frac{|A \cap B|}{|\Omega|} = \frac{P(A \cap B)}{P(A)} = \frac{12}{36} = \frac{1}{3}$$

Let's code!
The code of the previous example.

```
#Code109.py

from functools import partial
import sys;sys.path.append('../lib')
from utils import Pe, Pgiven, set_Filter

# get_diff_pairs: returns cartesian product of U discarding the diagonal elements
def get_diff_pairs(U):
    return [(i,j) for i in U for j in U if i!=j]

R = { 'r1' , 'r2', 'r3',  'r4' }
B = { 'b1', 'b2', 'b3', 'b4', 'b5', 'b6' }

Omega=get_diff_pairs(R | B)
P=partial(Pe,Omega)

# selects the outcomes having the 1st element starting with 'r'
A=set_Filter(lambda X:X[0][0]=='r',Omega)
print("Probability of A:",P(A))

B=set_Filter(lambda X:X[1][0]=='r',Omega)
print("Probability of B:",P(B))

AB=set_Filter(lambda X:X[0][0]=='r' and X[1][0]=='r',Omega)
print("Probability of AB:",P(AB))

print("Probability of B given A:",Pgiven(B,A))

#______________________________    Output  ______________________________________
# Probability of A: 2/5
# Probability of B: 2/5
# Probability of AB: 2/15
# Probability of B given A: 1/3
```

1.4.1 Compound probability

A box contains two blue balls and three red balls, what is the probability to draw (without replacement) a red ball, and then a blue ball.
The common sense is that 3/5 is the probability that the first ball is red (unconditional), and then $2/4 = 1/2$ is the probability that the second drawn ball is blue because in the second draw, we know that it remains only 4 balls in the box (conditional). So, the probability that the first

is red and the second is blue is $3/5 \times 1/2 = 3/10$: if R is the event to draw a red ball first and B the event to draw a blue ball second, then $P(R \cap B) = P(R) \times P(B|R)$, this is called *the chain rule* and can be extended to n successive events as follows:

> **Proposition 3.**
> *Let $(E_i)_{i \in I}$ be a collection of events*
>
> *If I is finite and $\forall i \in I, P(E_i) \neq 0$ then :*
>
> $$P(\bigcap E_i) = P(E_1)P(E_2|E_1) \cdots P(E_n|E_1 \cap \cdots \cap E_{n-1})$$

1.4.2 Total probability

We consider the experiment that consists in rolling a die, the outcome is in {1,2,3,4,5,6}, and then a coin is tossed a number of times equal to the outcome of the rolled die. The coin is fake so that P(T)=3/4 and P(H)=1/4, and we want to determine the probability of not obtaining a Tail in this experiment.

Ω of this experiment is a partition, each of its element correspond to a value k obtained by the die rolling. For example for $k = 2$, the associated set is $\{(T, T), (T, H), (H, T), (H, H)\}$. For each value of k, the probability of event E_k (not getting Tail in k tosses) is $P(E_k) = (1/4)^k$, and because the events corresponding to different values of k are mutually exclusive, then the probability of not obtaining a Tail is:

$$\frac{1}{4}\frac{1}{6} + (\frac{1}{4})^2\frac{1}{6} + (\frac{1}{4})^3\frac{1}{6} + (\frac{1}{4})^4\frac{1}{6} + (\frac{1}{4})^5\frac{1}{6} + (\frac{1}{4})^6\frac{1}{6} = 0.1641$$

> **Proposition 4.**
> *Let $(E_i)_{i \in I}$ be a partition of Ω (mutually exclusive events) and A an event, then :*
>
> $$P(A) = \sum_{i \in I} P(A|E_i)P(E_i)$$

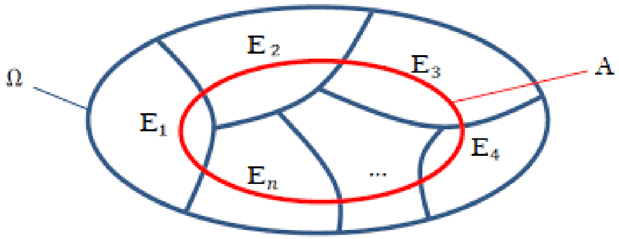


Figure 1.2: Total probability diagram

Consider the partitions A and A^c ($\bar{A}$) of Ω, we can visualize the total probability formula by the conditional probability diagram whose branches attached to the root represent this partition with its associated probability to which other branches are attached and which represent the events B and $\bar{B}$ conditioned by A and $\bar{A}$. To calculate the total probability of B, we have just to sum the probabilities of the branches leading to B.

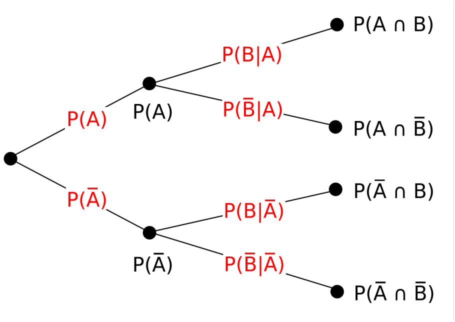


Figure 1.3: Conditional probabilities diagram

1.4.3 Bayes Formula

Proposition 5. ***Bayes Formula***
Let (E_i) *be a partition of* Ω *and* A *an event, then :*

$$P(E_i|A) = \frac{P(A|E_i)P(E_i)}{\sum_{j\in I} P(A|E_j)P(E_j)}$$

This results from the fact that:

$$P(E_i|A) = \frac{P(E_i \cap A)}{P(A)} = \frac{P(A\cap E_i)}{P(A)} = \frac{P(A|E_i)P(E_i)}{\sum_{j\in I} P(A|E_j)P(E_j)}$$

Another approach to present Bayes rule, is using *probability rate (odds)*. The probability rate of an event A is defined by:

$$O_A = \frac{P(A)}{P(A^c)} = \frac{P(A)}{1-P(A)} \text{ , and the probability of } A,\ P(A) = \frac{O_A}{1+O_A}$$

Let H and E be two events which represent respectively a hypothesis and a proof of a given case.

$$O_{H|E} = \frac{P(H|E)}{1-P(H|E)} = \frac{P(H|E)}{P(H^c|E)} = \frac{P(E|H)P(H)}{P(E)} \times \frac{P(E)}{P(E|H^c)P(H^c)} = \frac{P(H)}{P(H^c)} \times \frac{P(E|H)}{P(E|H^c)}$$

$O_H = \frac{P(H)}{P(H^c)}$ is the *"a priori" probability rate* of H (before knowing E).
$\frac{P(E|H)}{P(E|H^c)}$ is called the *plausibility rate* or the *Bayes factor*.

$$O_{H|E} = O_H \times \frac{P(E|H)}{P(E|H^c)}$$

This rule updates the "a priori" probability rate of $H|E$ by multiplying the "a priori" probability rate of H by the Bayes factor, which gives us an indication on how much the new evidence (E) changes the hypothesis belief.

Example 9. *[10] A team of scuba divers searches for a ship wreck. They believe that it is in the search area with a probability* 0.4*. A search in this zone is successful with a probability* 0.9 *if it exists. What is the probability that the wreck is in this zone if the search fails?*

Solution
H: the wreck exists
E: the search fails
We want to calculate $P(H|E)$*.*
Method1.

$$O_{H|E} = \frac{P(H|E)}{P(H^c|E)} = \frac{P(H)}{P(H^c)} \times \frac{P(E|H)}{P(E|H^c)} = \frac{0.4}{0.6} \times \frac{0.1}{1} = \frac{1}{15}$$

$$P(H|E) = \frac{O_{H|E}}{O_{H|E}+1} = \frac{1}{16}$$

Method2.

$$P(H|E) = \frac{P(E|H) \times P(H)}{P(E|H) \times P(H) + P(E|H^c) \times P(H^c)} = \frac{0.1 \times 0.4}{0.1 \times 0.4 + 1 \times 0.6} = \frac{1}{16}$$

Example 10. *What is the probability of drawing at least 5 cards (5 or more) from a deck of 52 cards before we get the first ace.*

Solution
The first method to solve this problem is to find the size of the sample space Ω *that contains* 52! *elements, and to compute the number of elements in* E*, the event of drawing at least 5 cards before we get the first ace.* $|E| = 48 \times 47 \times ... \times 44 \times 47!$*. We have:*

$$P(E) = \frac{48 \times 47 \times ... \times 44 \times 47!}{52!} = 0.6588$$

A simpler method consists in applying the chain rule by considering the successive steps of the experiment. Let E_i *be the event that the* i^{th} *draw is not an ace, then:*

$$\begin{aligned} P(E_1 \cap E_2 \cap ... \cap E_5) =& P(E_1)P(E_2|E_1) \cdots P(E_5|E_4, E_3, ..., E_1) \\ =& \frac{48}{52} \times \frac{47}{51} \times ... \times \frac{44}{48} \\ =& 0.6588 \end{aligned}$$

In the second method, we need to find the probabilities of some well chosen events without specifying explicitly the sample space.

Example 11. *In an exam, the students have to answer one question with multiple choices. Some student knows the correct answer with a probability* p *or guesses it with a probability* $1-p$*. Having* n *answers to choose from with only one correct answer. What is the probability that this student knows the right answer given that he chose the correct one?*

Solution
Let E *be the event that the student knows the right answer, and* F *the event that his answer is correct. Using Bayes formula, we have :*

$$\begin{aligned} P(E|F) =& \frac{P(E \cap F)}{P(F)} \\ =& \frac{P(F|E) \times P(E)}{P(F|E) \times P(E) + P(F|E^c) \times P(E^c)} \\ =& \frac{1 \times p}{1 \times p + 1/n \times (1-p)} \\ =& \frac{n \times p}{1 + (n-1) \times p} \end{aligned}$$

Let’s code!

The code of the previous example.

```
#Code110.py

from itertools import product

N, p = 4, 0.4

# omega’s outcomes are represented as tuples (X,i,j) s.t :
# X is Y if the student knows the answer and N otherwise
# i is the correct answer and j is the given answer
E = set(product({’N’},set(range(1,N+1)),set(range(1,N+1))))
F = set([(’Y’,i,i) for i in range(1,N+1)])
Omega = F.union(E);
```

```
Dist =  { o : (1-p)/(N*N) if o[0]==’N’ else p/N  for o in Omega }
print(Dist)

PF = 0; PEF=0
for i in Omega:
    if i[1] == i[2]: PF  += Dist[i]
    if i[0] == ’Y’ : PEF += Dist[i]

print("P(E|F) = %0.5f" % (PEF/PF))
print("Bayes rule: ",N*p/(1+(N-1)*p))

#______________________________  Output  ______________________________________
# {(’N’, 3, 3): 0.0375, (’Y’, 4, 4): 0.1, (’Y’, 2, 2): 0.1, (’N’, 4, 3): 0.0375,
#  (’N’, 3, 2): 0.0375, (’N’, 1, 4): 0.0375, (’N’, 2, 2): 0.0375, (’N’, 4, 4): 0.0375,
#  (’Y’, 1, 1): 0.1, (’N’, 3, 4): 0.0375, (’N’, 1, 3): 0.0375, (’N’, 2, 3): 0.0375,
#  (’N’, 1, 2): 0.0375, (’N’, 1, 1): 0.0375, (’N’, 2, 1): 0.0375, (’Y’, 3, 3): 0.1,
#  (’N’, 3, 1): 0.0375, (’N’, 2, 4): 0.0375, (’N’, 4, 1): 0.0375, (’N’, 4, 2): 0.0375}
# P(E|F) = 0.72727
# Regle de Bayes:  0.7272727272727273
```

Example 12. *Consider three boxes, each containing 100 balls.*

- Box 1 contains 75 red and 25 green balls.
- Box 2 contains 60 red and 40 green balls.
- Box 3 contains 45 red and 55 green balls.

We choose randomly a box, and then we randomly draw a ball from this box.
1- What is the probability that the chosen ball is red?
2- Assuming that we draw a red ball, what is the probability that it was drawn from Box 1 ?

Solution
- Let E be the event that the drawn ball is red.
- Let F_i be the event that the i^{th} Box is chosen.
1- Using the total probability theorem, we have:

$$\begin{aligned} P(E) =& P(E|F_1)P(F_1) + P(E|F_2)P(F_2) + P(E|F_3)P(F_3) \\ =& 75/100 \times 1/3 + 60/100 \times 1/3 + 45/100 \times 1/3 \\ =& 3/5 \end{aligned}$$

2- Using Bayes rule, we have:

$$\begin{aligned} P(F_1|E) =& \frac{P(E|F_1)P(F_1)}{P(E)} \\ =& (75/100 \times 1/3)/(3/5) \\ =& 5/12 \end{aligned}$$

Using the conditional probability diagram, we can find the probability of the branches that end with E. Just add its values to find the probability of E (the total probability).

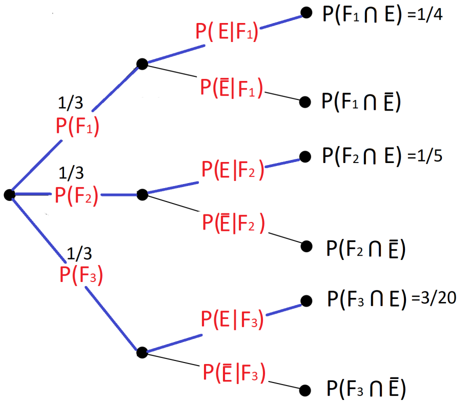


Figure 1.4: Conditional probabilities diagram

Example 13. *[9]*
A disease affects one out of 10000 people. There is a test to check if someone is affected.
The probability that the test's result is positive when the person don't have the disease is 2%.
The probability that the test's result is negative whereas the person has the disease is 1%.
Someone was tested positive to the disease, what is the probability that this person has indeed the disease?

Solution
Let M be the event that the person has the disease, and T the event that the test is positive.

$$\begin{aligned} P(M) =& \frac{1}{10000} \\ P(T|M^c) =& 0.02 \\ P(T^c|M) =& 0.01 \\ P(T|M) =& 1 - 0.01 = 0.99 \end{aligned}$$

Using Bayes rule:

$$\begin{aligned} P(M|T) =& \frac{P(T|M)P(M)}{P(T|M)P(M) + P(T|M^c)P(M^c)} \\ =& \frac{0.99 \times 0.0001}{0.99 \times 0.0001 + 0.02 \times (1 - 0.0001)} = 0.0049 \end{aligned}$$

1.5 Independence

We should note that in *example 1.8* if we return the balls after each draw, the probabilities do not change, this is due to the fact that the initial environment of the experiment does not change (as if the same experiment is repeated one more time).
Events are *independents* when each event is not affected by the others. This is the case of coin tosses where each toss is an independent event, i.e the previous tosses do not affect the current toss and the probability is 1/2 at each toss.

Definition 7.
Two events E and F are independent (notation: $E \perp F$) if:

$$P(E \cap F) = P(E)P(F)$$

This definition implies that: $P(E|F) = P(E)$, this means that E and F are independent if knowing that F has occurred does not affect the probability that E occurs.
Sometimes the independence of two events is clear because there are no physical interactions between them; but when this is not the case, one must check whether the condition of independence is satisfied.

Example 14. *Suppose two dice are rolled. Let A be the event that the result of the first die is equal to 3, B is the event that the sum of the two is 9 and C the event that the sum of the two is equal to 7. Are A and B independent? Same question for A and C.*

Solution
The experiment has 36 equally likely results $(i, j) : 1 \leq i, j \leq 6$ such that i is the result of the

first die and j the result of the second one.

$$P(A) = P(\{(i,j)|i = 3, 1 \leq j \leq 6\}) = \frac{6}{36} = \frac{1}{6}$$
$$P(B) = P(\{(i,j)|i + j = 9, 1 \leq i, j \leq 6\}) = \frac{4}{36} = \frac{1}{9}$$
$$P(A \cap B) = P(\{(3,6)\}) = \frac{1}{36}$$
$$P(A \cap B) \neq P(A)P(B)$$

A and B are therefore dependent.

$$P(C) = P(\{(i,j)|i + j = 7, 1 \leq i, j \leq 6\}) = \frac{6}{36} = \frac{1}{6}$$
$$P(A \cap C) = P(\{(3,4)\}) = \frac{1}{36}$$
$$P(A \cap C) = P(A)P(C)$$

A and C are therefore independent.

We can explain this result by the fact that event C can occur for any value obtained by the first die, so it is independent of this one. This is not the case for event B because we cannot have a sum of 9 if for example the first die gives 1 or 2, so the occurrence of B depends on the result of the first die.

In the case of countable sample spaces, one should not confuse disjoint and independent events. Indeed, if E and F are disjoint then they are not independent because if $P(E) \neq 0$ and $P(F) \neq 0$ then, $P(E \cap F) = 0$ because $E \cap F = \varnothing$ and $P(E) \times P(F) \neq 0$ because both are not zero which means that E and F are not independent.

1.5.1 Conditional independence

The concept of independence can be extended to conditionally independent events.

Definition 8.
Two events A and B are conditionally independent given another event C such that $P(C) > 0$ if :

$$P(A \cap B|C) = P(A|C) \times P(B|C)$$

If A and B are conditionally independent, then we have $P(A|B, C) = P(A|C)$. Indeed:

$$P(A|B, C) = \frac{P(A \cap B|C)}{P(B|C)} = \frac{P(A|C) \times P(B|C)}{P(B|C)} = P(A|C)$$

Example 15. *In a box there are two coins, the first is fair and the second is biased so that $p(H) = 1$. We choose a coin at random and toss it twice. Let define the following events:*

- *A: the event that the first toss gives head (H).*
- *B: the event that the second toss gives head (H).*
- *C: the event that the first coin is chosen.*

Find $P(A|C), P(B|C), P(A \cap B|C), P(A), P(B), P(A \cap B)$

Solution
$P(A|C) = P(B|C) = \frac{1}{2}$
$P(A \cap B|C) = \frac{1}{4} = \frac{1}{2} \times \frac{1}{2} = P(A|C)P(B|C)$
Therefore, A and B are conditionally independent.

Using the total probability rule, we get:

$$\begin{aligned}
P(A) &= P(A|C)P(C) + P(A|C^c)P(C^c) = \frac{1}{2} \times \frac{1}{2} + 1 \times \frac{1}{2} = \frac{3}{4} \\
P(B) &= P(B|C)P(C) + P(B|C^c)P(C^c) = \frac{1}{2} \times \frac{1}{2} + 1 \times \frac{1}{2} = \frac{3}{4} \\
P(A \cap B) &= P(A \cap B|C)P(C) + P(A \cap B|C^c)P(C^c) \\
&= P(A|C)P(B|C)P(C) + P(A|C^c)P(B|C^c)P(C^c) \\
&= \frac{1}{2} \times \frac{1}{2} \times \frac{1}{2} + 1 \times 1 \times \frac{1}{2} = \frac{5}{8}
\end{aligned}$$

Here, we can notice that A and B are not independent because :

$$P(A \cap B) = \frac{5}{8} \neq P(A)P(B) = \frac{9}{16}$$

Let’s code!

The code of the previous example.

```
#Code111.py

from sympy.stats import Coin, density, given, FiniteRV, P
from sympy import Symbol, Eq
import sys;sys.path.append(’../lib’)
from utils import get_Omega,set_Product

#pmf_i: returns the probability of getting tail in the ith toss of coin omega[0]
pmf_i = (lambda omega, i :
    (density(cf).dict[omega[i]] if omega[0]==1 else density(cu).dict[omega[i]]))

# The outcome is encoded as follows (H=1, T=2):
# chosen coin * 100 + 1st toss * 10 + 2nd toss
encode_omega = (lambda o:
               o[0]*100+(1 if o[1]==H else 2)*10+(1 if o[2] == H else 2))
get_proba = lambda o : PC[o[0]] * pmf_i(o, 1) * pmf_i(o, 2)

H=Symbol(’H’)

# PC: chosen coin, cu: unfair coin, cf: fair coin
PC, cu, cf  = { 1:1/2, 2:1/2}, Coin(’CU’, 1)  , Coin(’CF’)
omegaPC , omegaC = PC.keys(), get_Omega(cf)

# generates the SS of the experiment (X,Y,Z):
# X is the chosen coin, Y, Z is the outcome of resp, the 1st and 2nd  toss
Omega = set_Product([omegaPC, omegaC, omegaC])
dist = { o: PC[o[0]]* pmf_i(o, 1) * pmf_i(o, 2)  for o in Omega }
print("Distribution : ", dist)

# Encodes the distribution’s outcomes as numbers to make it easy for events handling
dist_encoded = { encode_omega(o) : get_proba(o) for o in Omega }
print("Encoded distribution : ",dist_encoded)

X    = FiniteRV(’X’, dist_encoded)
A    = X % 100 < 20   ; print("Probability of getting T in the first toss: %0.2f" % P(A))
B    = X % 10 < 2     ; print("Probability of getting T in the second toss: %0.2f" % P(B))
C    = X < 200        ; print("Probability of choosing the first coin: %0.2f" % P(C))
AB   = Eq(X % 100,11) ; print("Probability of A and B: %0.2f" % P(AB))

AGC  = given(A,C)     ; print("Probability of A|C: %0.2f" % P(AGC))
BGC  = given(B,C)     ; print("Probability of B|C: %0.2f" % P(BGC))
ABGC = given(AB,C)    ; print("Probability of A et B | C: %0.2f" % P(ABGC))

#______________________________  Output  ______________________________________
# Distribution:  {(2, T, T): 0, (2, H, H): 0.5, (2, T, H): 0, (2, H, T): 0,
# (1, T, T): 0.125, (1, H, T): 0.125, (1, T, H): 0.125, (1, H, H): 0.125}
# Encoded distribution:  {222: 0, 211: 0.5, 221: 0, 212: 0, 122: 0.125,
# 112: 0.125, 121: 0.125, 111: 0.125}
# Probability of getting T in the first toss: 0.75
# Probability of getting T in the second toss: 0.75
# Probability of choosing the first coin: 0.50
# Probability of A and B: 0.62
# Probability of A|C: 0.50
# Probability of B|C: 0.50
# Probability of A et B | C: 0.25
```

1.6 Exercises

Exercise 1.
Consider the following experiments :
- $\boldsymbol{R}_1$ *: The gender of the delegate chosen from a class of 20 students.*
- $\boldsymbol{R}_2$ *: The distance between the dart and the center of the target in the game of darts (the target has 30cm diameter).*
- $\boldsymbol{R}_3$ *: The IP address of the first website visited after a Google search (IP addresses are of the form a.b.c.d such that a, b, c, d are numbers between 0 and 255).*
- $\boldsymbol{R}_4$ *: The numbers of the balls obtained from a drawing without replacement of two balls (one after another) from an urn containing five numbered balls.*
- $\boldsymbol{R}_5$ *: The total time required to empty a water tank of volume V equipped with a valve of diameter D.*
- $\boldsymbol{R}_6$ *: The choice of a class among 6 classes and the choice of the math teacher among 3 teachers for this class.*
- $\boldsymbol{R}_7$ *: Number of births in a hospital per day*
- $\boldsymbol{R}_8$ *: The maximum duration of the delay of the students who arrive at a course of 60 mns.*
- $\boldsymbol{R}_9$ *: Throwing a bar B inside a circle C, then observing the angle between the horizontal diameter of C and B.*
- $\boldsymbol{R}_{10}$ *: Throwing a bar B of length l on a plane in which a circle C of diameter D is drawn and observing if B falls inside C.*

1- For each experiment determine if it is random or not, if the answer is yes, give its nature (simple / compound), its sample space and specify its type (discrete / continuous).
2- Implement experiments 1, 4, 6, 9 and 10 in Python.

Exercise 2.
1- Give the measurable space corresponding to the trivial σ-algebra and that of the maximal σ-algebra of REs 1,4, 6, 9, 11 of exercise 01.
2- Consider the following sets of parts of $\Omega = \{1, 2, 3\}$:
A- $F = \{\{1, 2\}, \{1, 2, 3\}\}$
B- $F = \{\{\}, \{1\}, \{1, 2, 3\}, \{2, 3\}\}$
C- $F = \{\{\}, \{1\}, \{2\}, \{2, 3\}\}$
D- $F = \{\{\}, \{1\}, \{2\}, \{3\}, \{1, 3\}, \{2, 3\}, \{1, 2\}, \{1, 2, 3\}\}$
Which ones are σ-algebra.

Exercise 3.
Give a measurable space that corresponds to a RE with a non-trivial and non-maximal σ-algebra.

Exercise 4.
For the measurable spaces of the REs of exercise 01, give :
1- Examples of events
2- Examples of disjoints events.

Exercise 5. *Coding*
Write the Python code that :
1- Generates the trivial and maximal σ-algebra of a given finite set.
2- Checks if a set of part of a given finite set is a σ-algebra.
3- Checks if a set is an event of a given RE.
4- Returns the union and the intersection of a collection of parts of Ω

Exercise 6. *Coding*
Using the class Die with 4 faces of the module sympy.stat (see chapter7)
1- Model the RE of tossing two Dice. Print its sample space.
2- Write a lambda expression of the event E_1 "the sum of the outcome is prime".
3- Write a lambda expression of the event E_2 " the product of outcomes is greater than k" (k given as argument).
4- Print event E_1 with its complement.
5- Print event E_2, its complement, its union and its intersection with E_1.

Exercise 7. *Coding*
Using the class Die with 8 faces of module sympy.stat (see chapter 7):
1- Print its sample space.
2- Write a lambda expression of the event "the outcome is multiple of k" (k given as argument).
3- Write a lambda expression of the event "the outcome is less than k"(k given as argument).
4- Print event A "the outcome is multiple of 2" and its complement.
5- Print event B "the outcome is multiple of 7", its complement, its difference and its symmetric difference with A.

Exercise 8.
1- Find c so that the following applications are probabilities :
i- $f(k) = c/k!$ *such that* $k \in \mathbb{N}$
ii- $f(k) = cx^k$ *such that* $0 < x < 1$ *and* $k \in \mathbb{N}$

Exercise 9. *Coding*
1- Write a Python function that returns the non equally likely distribution of a finite sample space Ω *of* n *elementary events, where* e_i *has the probability* $p_i = i/\sum_{k=1}^{n} k$.
2- Print the probability of events $A = \{i \text{ est pair }\}$ *and* $B = \{i < n/2\}$.

Exercise 10.
Consider a random experiment that consists in tossing three fair coins.
1- What is the probability that the outcome of the third coin is different from the outcomes of the two first ones ?
2- ***Coding****. Write the Python code that simulates this experiment.*

Exercise 11.
Consider a family of events $\{E_i\}_{\{1\leq i\leq m\}}$ *of the measurable space* $(\Omega, \mathcal{F}, P)$. *Let* $\{A_i\}_{\{1\leq i\leq k\}}$ *be a sub collection of events from this family satisfying the following conditions :*
i. $P(A_i) = \frac{1}{m}, \forall 1 \leq i \leq k$
ii. $P(A_k|A_1, A_2, ..., A_{k-1}) = \frac{1}{m-k+1}, \forall 1 \leq k \leq m$

1- Demonstrate the following property

$$P(A_k, A_{k-1}, ..., A_2, A_1) = \frac{(m-k)!}{m!}$$

2- For m large enough, demonstrate that:

$$P(A_m \cup A_{m-1} \cup ... \cup A_2 \cup A_1) \approx 1 - \frac{1}{e}$$

** For question 2, use the following formula :* $\sum_{n=0}^{\infty} \frac{x^n}{n!} = e^x$

Exercise 12.
1. Prove property 3 (the rule of union) of proposition 1.
2. Prove property 5 (the monotonicity) of proposition 1.

Exercise 13.
A fair coin is tossed 80 times. What is the probability of having exactly 40 tails?

Exercise 14.
What is the probability that two different faces each appear twice in the roll of 4 dice?
Coding*. Implement the solution of this exercise by:*
1. Generating the elements of Ω*, the associated events and calculating the requested probability.*
2. Calculating the same probability using combinatorial analysis.

Exercise 15.
Two dice are rolled. What is the conditional probability that the sum of the two is equal to 6 given that the two obtained values are different.
Coding*. Implement the solution of this exercise by:*
1. Generating the elements of Ω*, the associated events and calculating the requested probability.*
2. Calculating the same probability using combinatorial analysis.

Exercise 16.
Consider an urn containing 8 balls, 2 red and 6 of other colors different from red. What is the probability of having to draw at least 3 balls before getting the first red ball.
Coding. *Write the Python code that calculates the requested probability using: 1. combinatorial analysis and 2. the chain rule.*

Exercise 17.
Prove proposition 2 of the conditional probability.

Exercise 18.
Consider a committee that contains 8 men and 6 women. Two people are chosen to be responsible of this committee.
1- What is the probability that the second chosen person is a woman given that the first one is a man?
2- Write the Python code of this experiment.

Exercise 19.
We have three classes, each has 20 students.
- Class 1 contains 16 students from district A and 4 from district B
- Class 2 contains 14 students from district A and 6 from district B.
- Class 3 contains 12 students from district A and 8 from district B.
We choose a class randomly, then a random student is chosen from that class.
1- What is the probability that the chosen student is from district B?
2- Suppose we have chosen a student from district A, what is the probability that he is from class 1.
3- Write the Python code of this experiment.

Exercise 20. *Two players play a game that consists of, in turn, drawing a question at random and answering it, the first who gives a false answer loses the game. The first player has a probability p_1 of answering correctly and the second has a probability p_2 to answer correctly. What is the probability that the first player loses the game.*

Exercise 21. *[10]*
In a murder case, two fugitives X and Y are suspected. After an initial investigation, it turns out that both of them have the same probability of being the killer, that the real killer has group A blood, and 10% of the population have that blood group. Another investigation reveals that X has type A blood group, but Y's blood type is unknown. What is the probability that X is the killer?

Exercise 22.
Two dice are rolled. Let E be the event that the first gives an even number and F the event that the sum of the two is odd. Are E and F independent?

Exercise 23.
A computer application contains 4 errors: e_1, e_2, e_3, e_4. In the verification phase, the application is tested by several testers independently from each another. Each tester has a $1/3$ probability of finding and correcting each error. It is assumed that the errors are independent of each other and so are the tests.
1. What is the probability that error e_1 will not be corrected at the end of the n^{th} test?
2. What is the probability that all errors will be corrected at the end of the n^{th} test?
3. How many testers do we need to have a probability greater than or equal to 0.9 to correct all the errors?

Exercise 24.
A person participates in a game. He is in front of three boxes, one of the three contains an amount of money and the other two are empty. The participant must choose a box. He either knows the winning box with probability p or guesses it (with probability $1 - p$). What is the probability that the participant already knew the winning box (cheated) given that he has chosen the one that contains the amount of money.

Exercise 25.
We have two boxes, the first B_1 contains 2 white balls and 7 black balls and the second B_2 contains 5 white balls and 6 black balls. We toss a coin, if we get a Head we choose B_1 otherwise B_2 is chosen, and then we draw a ball from the chosen box and put it back in the other box. We repeat this experiment m times, we are interested in the color of the last drawn ball.
1. For $m = 2$, find the probability that the coin toss sequence is heads and tails given that the last ball drawn is white.
2. Write the Python code that models this experiment.

1.7 Solutions

Solution 1.
- $\boldsymbol{R}_1$ *: Yes, Simple,* $\Omega_1 = \{M, F\}$ *Discrete.*
- $\boldsymbol{R}_2$ *: Yes, Simple,* $\Omega_2 = [0, 15]$*, Continuous*
- $\boldsymbol{R}_3$ *: Yes, Simple,* $\Omega_3 = \{a.b.c.d|a, b, c, d \in \{0, ..., 255\}\}$*, Discrete*
- $\boldsymbol{R}_4$ *: Yes, Compound,* $\Omega_4 = \{(b_1, b_2) \in \{1, ..., 5\}^2|b_1 \neq b_2\}$*, Discrete*
- $\boldsymbol{R}_5$ *: No, The time required to empty the tank can be calculated as a function of V and D.*
- $\boldsymbol{R}_6$ *: Yes, Compound,* $\Omega_6 = \{(C, E)|C \in \{1, ..., 6\}, E \in \{1, 2, 3\}\}$*, Discrete*
- $\boldsymbol{R}_7$ *: Yes, Simple,* $\Omega_7 = \{0, 1, 2, 3, ...\} = \mathbb{N}$*, Discrete*
- $\boldsymbol{R}_8$ *: Yes, Simple,* $\Omega_8 = [0, 60]$*, Continuous*
- $\boldsymbol{R}_9$ *: Yes, Simple,* $\Omega_9 = [0, 2\pi]$*, Continuous*
- $\boldsymbol{R}_{10}$ *: Yes, Simple,* $\Omega_{10} = \{I, O\}$*, Discrete*

Solution 2.
1- Trivial and maximal σ-algebra:
a- $\boldsymbol{R}_1$, measurable space with trivial σ-algebra :
$(\{M, F\}, \{\{\}, \{M, F\}\})$ *and maximal σ-algebra :* $(\{M, F\}, \{\{\}, \{M\}, \{F\}, \{M, F\}\})$.
b- $\boldsymbol{R}_4$, measurable space with trivial σ-algebra : $(\Omega_4, \{\{\}, \Omega_4\})$ *and maximal σ-algebra :* $(\Omega_4, 2^{\Omega_4})$.
c- $\boldsymbol{R}_6$, measurable space with trivial σ-algebra : $(\Omega_6, \{\{\}, \Omega_6\})$ *and maximal σ-algebra :* $(\Omega_6, 2^{\Omega_6})$.
d- $\boldsymbol{R}_9$, measurable space with trivial σ-algebra : $(\Omega_9, \{\{\}, \Omega_9\})$ *and maximal σ-algebra :* $(\Omega_9, 2^{\Omega_9})$.
e- $\boldsymbol{R}_{11}$, measurable space with trivial σ-algebra : $(\Omega_{11}, \{\{\}, \Omega_{11}\})$ *and maximal σ-algebra :* $(\Omega_{11}, 2^{\Omega_{11}})$.
2- σ-algebra:
A- No, it does not contain the empty set.
B- Yes, it satisfies all the properties of the σ-algebra on Ω.
C- No, no sample space.
D- Yes, maximal σ-algebra.

Solution 3.
Consider the RE that consists of rolling a die with 4 faces, so its random space is $\Omega = \{1, ..., 4\}$, we choose the set $F = \{\{\}, \{2, 3, 4\}, \{1\}, \Omega\}$ that satisfies the properties of the σ-algebra on Ω. $(\Omega, \mathcal{F})$ is a measurable space $\mathcal{F}$ non-trivial and non-maximal.

Solution 4.
- $\boldsymbol{R}_1$ *:* $\{M\}$, $\{M\} \cap \{F\} = \varnothing$
- $\boldsymbol{R}_2$ *:* $\{x \leq 10\}$, $\{x \leq 5\} \cap \{10 \leq x \leq 15\} = \varnothing$
- $\boldsymbol{R}_3$ *:* $\{x|x$ *starts with 192*$\}$, $\{x|x$ *starts with 192*$\} \cap \{x|x$ *starts with 178*$\} = \varnothing$
- $\boldsymbol{R}_4$ *:* $\{(b_1, b_2), (b_3, b_4)\}$, $\{(b_1, b_2), (b_3, b_4)\} \cap \{(b_2, b_1), (b_4, b_3)\} = \varnothing$
- $\boldsymbol{R}_6$ *:* $\{(C_1, E_2), (C_3, E_1)\}$, $\{(C_1, E_2), (C_3, E_1)\} \cap \{(C_2, E_3), (C_4, E_2)\} = \varnothing$
- $\boldsymbol{R}_7$ *:* $\{x \geq 10\}$, $\{x < 10\} \cap \{10 \leq x \leq 20\} = \varnothing$
- $\boldsymbol{R}_8$ *:* $\{x \in [0, 15]\}$, $\{x \in [0, 10]\} \cap \{x \in]10, 15]\} = \varnothing$
- $\boldsymbol{R}_9$ *:* $\{a \in [0, \pi/2]\}$, $\{a \in [0, \pi/4]\} \cap \{a \in [\pi/3, \pi/2]\} = \varnothing$
- $\boldsymbol{R}_{10}$ *:* $\{I\}$, $\{I\} \cap \{O\} = \varnothing$

Solution 5. *codage*

Solution 6. *codage*

Solution 7. *codage*

Solution 8.
1- $f(k) = c/k!$ for $k \in \mathbb{N}$
f must satisfy the three conditions of probability : positivity, normality and additivity.
i- Positivity : c must be positive $c \geq 0$
ii- Additivity : f satisfies this condition since it is defined on a countable set.
iii- Normality : f must satisfy $\sum_{k=0}^{\infty} f(k) = 1$ then, $\sum_{k=0}^{\infty} c/k! = c \sum_{k=0}^{\infty} 1/k! = ce = 1$
So for $c = 1/e$, f is a probability.

2- $f(k) = cx^k$ for $k \in \mathbb{N}$
f must satisfy the three conditions of probability : positivity, normality and additivity.
i- Positivity : c must be positive $c \geq 0$
ii- Additivity : f satisfies this condition since it is defined on a countable set.
iii- Normality : f must satisfy $\sum_{k=0}^{\infty} f(k) = 1$ then, $\sum_{k=0}^{\infty} cx^k = c \sum_{k=0}^{\infty} x^k = c/(1-x) = 1$ puisque $0 < x < 1$.
So for $c = 1 - x$, f is a probability.

Solution 9. *coding*

Solution 10.
1. $\Omega = \{h, t\}^3, \quad E : \{(i, j, k) \in \Omega | k \neq i \text{ and } k \neq j\}, \; P(E) = 1/4$

Solution 11.
Consider a sub collection of events $\{A_i\}_{\{1 \leq i \leq k\}}$ from of $\{E_i\}_{\{1 \leq i \leq m\}}$ that satisfies (i) and (ii) $(1 \leq k \leq m)$.

1- We demonstrate (1) by induction on k.
For $k = 1$, this case is verified using hypothesis (i) we have : $P(A_k) = \frac{1}{m}$
We assume that the property is true for k and we demonstrate it for $k + 1 \leq m$:

$$\begin{aligned}
P(A_{k+1}, A_k, ..., A_2, A_1) &= P(A_{k+1} | A_k, ..., A_2, A_1) P(A_k, ..., A_1) \\
&= \frac{1}{(m-k+1)} \frac{(m-k)!}{m!} \\
&= \frac{(m-(k+1))!}{m!}
\end{aligned}$$

So the property is true.
2- Let m be a large number, we apply the inclusion-exclusion rule :

$$\begin{aligned}
P(A_m \cup ... \cup A_1) &= \sum_{k=1}^{m} P(A_k) - \sum_{i \leq j \leq m} P(A_i, A_i) + ... + (-1)^{m-1} (P(A_m, ..., A_2, A_1) \\
&= C_m^1 \frac{1}{m} - C_m^2 \frac{(m-2)!}{m!} + C_m^3 \frac{(m-3)!}{m!} + ... + (-1)^{m-1} C_m^0 \frac{1}{m!} \\
&= \frac{1}{1!} - \frac{1}{2!} + \frac{1}{3!} + ... + (-1)^{m-1} \frac{1}{m!} \\
&= \sum_{k=1}^{m} \frac{(-1)^{k-1}}{k!} \approx 1 - e^{-1} \quad \text{for a large } m
\end{aligned}$$

Solution 12.
*3- **Union**: $P(E \cup F) = P(E) + P(F) - P(E \cap F)$*

$$\begin{aligned}
E \cup F &= (E - F) \cup (F - E) \cup (E \cap F) \\
&\quad (E - F), (F - E) \text{ and } (E \cap F) \text{ are disjoints} \\
P(E \cup F) &= P(E - F) + P(F - E) + P(E \cap F) \\
&= P(E) - P(E \cap F) + P(F) - P(E \cap F) + P(E \cap F) \\
&= P(E) + P(F) - P(E \cap F)
\end{aligned}$$

5- ***Monotonicity:*** $\forall E, F \in \mathcal{F},\ E \subset F \Rightarrow P(E) \leq P(F)$.

$$P(F) = P(E \cup E_F^c) = P(E) + P(E_F^c) - P(E \cap E_F^c) = P(E) + P(E_F^c) \geq P(E)$$

Such that E_F^c is the complement of E in F

Solution 13.

In this experiment, the sample space is made up of all the heads and tails sequences of length 80. Its size is therefore 2^{80} sequences. The number of sequences having exactly 40 tails is equal to C_{80}^{40} (see appendix A), so the probability of having exactly 40 tails is: $\frac{C_{80}^{40}}{2^{80}}$.
In order to calculate this value, we use the following Stirling approximation: $n! \approx \sqrt{2\pi n}(\frac{n}{e})^n$ for a sufficiently large value of n $(n \geq 10)$ and $e = 2.718$. Which gives $\frac{1}{2\sqrt{\pi 10}} = 0.089$.

Solution 14.

$\Omega = \{1, .., 6\}^4$
E is the event of having two different heads each appearing twice.
We have C_6^2 possibilities of the two different heads that appear and for each of the possibilities, there are $\frac{4!}{2!2!} = 6$ of possible arrangements (permutations with repetition).
So $P(E) = \frac{6 \times \frac{6!}{2!4!}}{|\Omega|} = \frac{60}{6^4}$.

Solution 15.

The set of all possible outcomes is made up of 36 equally likely elements. Let A be the event that the sum of the two is equal to 6, and B the event that the outcomes are different. In all, there are 30 outcomes with different values and 4 of them have the sum equal to 6. Then $P(A \cap B) = 4$ and $P(B) = 30$. So, $P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{2}{15}$

Solution 16.

Let E_i be the event that the drawn ball from the i^{th} draw is not red.
Let E be the event of having to draw at least three balls before having the first red ball.
$P(E) = P(E_1 \cap E_2 \cap E_3)$ using the chain rule:
$P(E) = P(E_1)P(E_2|E_1)P(E_3|E_1, E_2) = \frac{6}{8} \times \frac{5}{7} \times \frac{4}{6} = \frac{5}{14}$

Solution 17.

We demonstrate that $P_{.|A}$ satisfies the properties of a probability on the measurable space $(A, 2^A)$:
i- $P_{.|A}(A) = \frac{P(A \cap A)}{P(A)} = 1$
ii- Consider a family of mutually exclusive events $\{E_i\}_{i>0}$:
$P_{.|A}(\bigcup_{i>0} E_i) = \frac{P((\bigcup_{i>0} E_i) \cap A)}{P(A)} = \frac{P(\bigcup_{i>0}(E_i \cap A))}{P(A)} = \sum_{i>0} \frac{P(E_i \cap A)}{P(A)} = \sum_{i>0} P_{.|A}(E_i)$

Solution 18.

1.The fact that the first chosen person is a man, reduces the sample space to 13 persons of which 6 women and 7 men. So the probability that the second chosen person is a woman is $6/13$ (using the new sample space).
2. Coding.

Solution 19.

1. Let E_A, E_B be the event that the student is from district A, B respectively.
Let C_i the event that the student is from class i.
Using the rule of total probability:

$$\begin{aligned} P(E_B) &= P(E_B \cap C_1) + P(E_B \cap C_2) + P(E_B \cap C_3) \\ &= P(E_B|C_1)P(C_1) + P(E_B|C_2)P(C_2) + P(E_B|C_3)P(C_3) \\ &= 4/20 \times 1/3 + 6/20 \times 1/3 + 8/20 \times 1/3 = 3/10. \end{aligned}$$

2. The probability that the student is in class 1 given that he is from district A. By applying

Bayes rule:

$$P(C_1|E_A) = \frac{P(C_1 \cap E_A)}{P(E_A)}$$
$$P(C_1 \cap E_A) = P(E_A \cap C_1) = P(E_A|C_1)P(C_1) = \frac{16}{20} \times \frac{1}{3}$$
$$P(C_1|E_A) = \frac{8}{30} \times \frac{10}{7} = \frac{8}{21}$$

Solution 20. *Let define events: $E =$ "player1 loses" and E_k: "player1 loses on the k^{th} turn".*

$$P(E_k) = (p_1p_2)^{k-1}(1-p_1) \text{ so}$$
$$P(E) = P(E_1 \cup E_2 \cup E_3....)$$

Since the E_i are disjoint events then:

$$\begin{aligned} P(E) &= P(E_1) + P(E_2) + P(E_3) + \cdots \\ &= (1-p_1)[1 + p_1p_2 + (p_1p_2)^2 + (p_1p_2)^3...] \\ &= \frac{1-p_1}{1-p_1p_2} \end{aligned}$$

Solution 21.
Consider the two events: H : X is the real killer and E : X has blood type A.
If we use the ODDs method :

$$O_{H|E} = \frac{P(H|E)}{P(H^c|E)} \text{ and } P(H|E) = \frac{O_{H|E}}{O_{H|E}+1}$$
$$\frac{P(H|E)}{P(H^c|E)} = \frac{P(H)}{P(H^c)} \times \frac{P(E|H)}{E|H^c} = \frac{1/2}{1/2} \times \frac{1}{0.1} = 10$$

So, $P(H|E) = \frac{10}{11}$.

If we apply directly the Bayes rule :

$$P(H|E) = \frac{P(E|H) \times P(H)}{P(E|H) \times P(H) + P(E|H^c) \times P(H^c)} = \frac{1 \times 1/2}{1 \times 1/2 + 0.1 \times 1/2} = \frac{0.5}{0.55} = \frac{10}{11}$$

Solution 22.
$E = \{(i,j)|i \in \{2,4,6\}, 0 \leq j \leq 6\}$
$P(E) = 18/36 = 1/2$
$F = \{(i,j)|i+j \in \{3,5,7,9,11\} \text{ and } 0 \leq i,j \leq 6\}$
$P(F) = 18/36 = 1/2$
$E \cap F = \{(i,j)|i+j \in \{3,5,7,9,11\}, i \in \{2,4,6\}, 0 \leq j \leq 6\}$
$P(E \cap F) = 9/36 = 1/4 = P(E)P(F)$
So E and F are independent.

Solution 23.
1. Let A_n be the event that error e_1 is not corrected at the end of the n^{th} test, and let E_i be the event that error e_1 is not corrected by the i^{th} tester. $P(E_i) = 2/3$ and the events E_i are independent.
$P(A_n) = P(E_1 \cap E_2 \cap ... \cap E_n) = \prod_{i=1}^{n} 2/3 = (2/3)^n$

2. Let B_n be the event that all the errors are corrected at the end of n^{th} test and F_i the event that error e_i is not corrected at the end of n^{th} test.
$P(F_i) = (2/3)^n \ \forall i = 1, \ldots, 4.$
$P(B_n) = P(F_1^c \cap F_2^c \cap F_3^c \cap F_4^c) = \prod_{k=1}^{4}(1-(2/3)^n) = (1-(2/3)^n)^4$

3.

$$\begin{aligned} P(B_n) \geq 0.9 &\iff (1-(2/3)^n)^4 \geq 0.9 \\ &\iff n\log(2/3) \leq log(1-0.9^{1/4}) \\ &\iff n \geq \frac{\log(1-0.9)^{1/4}}{\log(2/3)} \\ &\iff n \geq 10. \end{aligned}$$

Solution 24.

K is the event that the person knows the box that contains the money.
S the event that the selected box contains the amount of money.

$$P(K|S) = \frac{P(S|K)P(K)}{P(S|K)P(K) + P(S|K^c)P(K^c)} = \frac{p}{p+(1/3)(1-p)}$$

Solution 25.

HT is the event of getting Head then Tail when the coin is flipped twice.
B is the event that the second moved ball is white.

$$P(HT|B) = \frac{P(B|HT)P(HT)}{P(B|HH)P(HH) + P(B|HT)P(HT) + P(B|TH)P(TH) + P(B|TT)P(TT)}$$

At each stage, each box has $1/2$ probability of being chosen, then the probability of choosing a white ball in the second draw depends on both the chosen box and the first drawn ball.
$P(B|FF) = 2/9 \times 1/8 + 7/9 \times 1/4$
$P(B|HT) = 1/2 \times 2/9 + 7/9 \times 5/12$
$P(B|TH) = 5/11 \times 3/10 + 6/11 \times 1/5$
$P(B|TT) = 5/11 \times 2/5 + 6/11 \times 1/2$

$P(HT|B) = 0.32$

Let’s code!

2. Coding

```
#exo1_24.py

from itertools import product
import copy


def prob_urne_boule(u,i):
    return u[i[2]][i[3]]/(u[i[2]][’B’] + u[i[2]][’N’])

u0={ ’F’:{’B’:2,’N’:7}, ’P’:{ ’B’:5, ’N’:6}}
omega=list(product([’F’,’P’],[’B’,’N’],[’F’,’P’],[’B’,’N’]))
PB = 0; PBFP=0
for i in omega:
    u = copy.deepcopy(u0)
    i0c = ’F’ if i[0] == ’P’ else ’P’

    # tirage et transfert de boule
    u[i[0]][i[1]] = u[i[0]][i[1]] - 1; u[i0c ][i[1]] = u[i0c ][i[1]] + 1

    P1 = prob_urne_boule(u0, (None,None, i[0],i[1]))
    P = 0.25 * P1 * prob_urne_boule(u,i)

    if i[3] == ’B’: PB = PB + P
    if i[0] == ’F’ and i[2]==’P’ and i[3]==’B’: PBFP += P
    print(i[0],"   ",i[1],"   %0.2f" %P, ’ \t ’, u)

print(’P(B) = %0.2f ’%PB, ’ \t P(F,P,B) = %0.2f’ % PBFP, ’ \t P(FP|B) = %0.2f’ % (PBFP/PB) )

#________________________________   Output  ________________________________________
# F     B    0.01     {’F’: {’B’: 1, ’N’: 7}, ’P’: {’B’: 6, ’N’: 6}}
# F     B    0.05     {’F’: {’B’: 1, ’N’: 7}, ’P’: {’B’: 6, ’N’: 6}}
# .....
# P     N    0.07     {’F’: {’B’: 2, ’N’: 8}, ’P’: {’B’: 5, ’N’: 5}}
# P     N    0.07     {’F’: {’B’: 2, ’N’: 8}, ’P’: {’B’: 5, ’N’: 5}}
# P(B) = 0.34      P(F,P,B) = 0.11     P(FP|B) = 0.32
```

Chapter 2

Random Variables

2.1 Introduction

When we have an experiment, such as coin tossing for example, and we assign a value to each elementary outcome, then the set of the obtained values defines a random variable. For instance, if we give value 10 to heads and value 20 to tails (suppose that these are the amount in Dinar that we are going to win by getting heads or tails), then we will have defined a random variable whose set of values is $\{10, 20\}$.

More formally, a random variable X is a real-valued function that assigns a value to each possible outcome ω of a random experiment in the probability space $(\Omega, \mathcal{F}, P)$.

Let $\mathcal{B}$ be the set of real subsets resulting from the finite union and intersection of the intervals of $\mathbb{R}$ called the set of *Borelians* which is the σ-*algebra* generated by the intervals of $\mathbb{R}$.

Definition 1.

Consider a random experiment whose probability space is $(\Omega, \mathcal{F}, P)$.

A random variable r.v is the function $X : \Omega \longrightarrow \mathbb{R}$ *such that for any Borelian* $B \in \mathcal{B}(\mathbb{R})$, *the image of the set* B $(X^{-1}(B) = \{\omega \in \Omega | X(\omega) \in B\}$ *noted* $\{X \in B\})$ *is an event of* Ω (*element of the* $\mathcal{F}$ σ-*algebra of* Ω).

$$\begin{aligned} X: \quad & \Omega \longrightarrow \mathbb{R} \\ & \omega \longmapsto x = X(\omega) \end{aligned}$$

$$X \textit{is a r.v} \implies \forall B \in \mathcal{B}(\mathbb{R}) : X^{-1}(B) \in \mathcal{F}$$

The domain of a r.v X *is the sample space of the random experiment* (Ω)

The rank R_X *of a r.v* X $(X(\Omega))$ *is the set of the possible values that it can take in* $\mathbb{R}$.

The event $\{X \in \{x\}\}$ will be noted $\{X = x\}$ (respectively $\{X \in]-\infty, x]\} = \{X \leq x\}$, $\{X \in]-\infty, x[\} = \{X < x\}$, $\{X \in [x, +\infty[\} = \{X \geq x\}$, $\{X \in]x, +\infty[\} = \{X > x\}$, $\{X \in]x_1, x_2]\} = \{x_1 < X \leq x_2\}$)

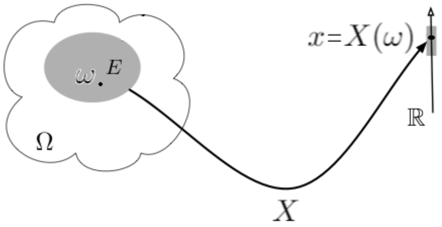


Figure 2.1: Random variable definition

Definition 2.
Let X be a r.v, we call the probability distribution of X (the law of X, noted P_X) the application $P_X : \mathcal{B}(\mathbb{R}) \longrightarrow [0,1]$ which associates to any Borelian B, the probability of the event $\{X \in B\}$

$$\begin{aligned} P_X : \mathcal{B}(\mathbb{R}) &\longrightarrow [0,1] \\ B &\longmapsto P_X(B) = P(\{X \in B\}) = P(X \in B) \end{aligned}$$

The diagram below shows the composition of the application X and P to define the law of X (P_X)

$$\begin{tikzcd} \mathcal{F}(\Omega) \arrow[r, "P"] \arrow[d, "X"'] & {[0,1]} \\ \mathcal{B}(\mathbb{R}) \arrow[ur, "P_X"] & \end{tikzcd}$$

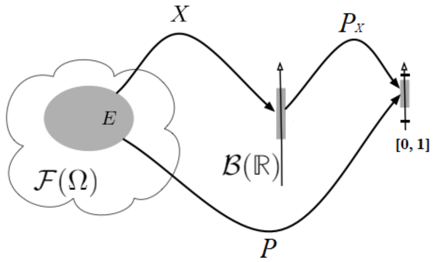


Figure 2.2: Random variable probability distribution (law)

Example 1. *Suppose our experiment consists in flipping two coins. Let X be the function that associates the number of the obtained heads with each result of this experiment, the values taken by this function are $0, 1$ and 2. For all $i \in \{0,1,2\}$, we denote by B_i the class of Borelians which contain only i. Any other Borelian is written as a union of the disjoint Borelians of these three classes. The inverse image of a borelian B_i is the inverse image of $\{i\}$ $(X^{-1}(B_i) = X^{-1}(\{i\}))$ which is in $\mathcal{F}$ and the inverse image of any other Borelian is the union of the inverse images of the disjoint Borelians of classes B_i which compose the latter, this image is in $\mathcal{F}$ so X is a r.v. Find P_X.*

Solution
X is a r.v $(X : \Omega \longrightarrow \mathbb{R})$ its domain is $\Omega = \{H,T\}^2$, its rank is $R_X = \{0,1,2\}$ and its distribution is P_X:

$$\begin{aligned} P_X(B_0) &= P(X \in B_0) = P(X^{-1}(B_0)) = P(X^{-1}(\{0\})) = P\{(H,H)\} = 1/4 \\ P_X(B_1) &= P(X \in B_1) = P(X^{-1}(B_1)) = P(X^{-1}(\{1\})) = P\{(H,T),(T,H)\} = 2/4 \\ P_X(B_2) &= P(X \in B_2) = P(X^{-1}(B_2)) = P(X^{-1}(\{2\})) = P\{(T,T)\} = 1/4 \end{aligned}$$

For the other Borelians, we apply the additivity rule of the probability P.

Definition 3.
Let X be a r.v, the distribution function of X (denoted by F_X) is the function defined from $\mathbb{R}$ to $[0,1]$ which for all $x \in \mathbb{R}$ associates the probability of the Borelian $]-\infty, x]$ (also noted $X \leq x$):

$$\begin{aligned} F_X : \mathbb{R} &\longrightarrow [0,1] \\ x &\longmapsto F_X(x) = P_X(]-\infty,x]) = P(X \in]-\infty,x]) = P(X \leq x) \end{aligned}$$

> *Also called the cumulative distribution function (cdf)*

The diagram below shows the composition of the application P_X and U that associates $x \in \mathbb{R}$ with its semi-interval $]-\infty, x]$ to define the cdf of X (F_X)

$$\begin{tikzcd}
\mathcal{B}(\mathbb{R}) \arrow[r, "P_X"] & {[0,1]} \\
\mathbb{R} \arrow[u, "U"] \arrow[ur, "F_X"] &
\end{tikzcd}$$

Example 2. *Following the previous example, we have:*

$$P_X(x) = \begin{cases} 1/4 & \textit{if } x = 0 \textit{ or } x = 2 \\ 2/4 & \textit{if } x = 1 \end{cases}$$

and which will be noted:

$$P_X = \frac{1}{4}\mathbb{1}_{\{0,2\}} + \frac{2}{4}\mathbb{1}_{\{1\}}$$

Find F_X the distribution function of X (cdf).

Solution

We have $F_X(x) = P(X \leq x) = P(X \in]-\infty, x])$

$$\begin{aligned}
&F_X(x) = 0, \quad x < 0 \\
&F_X(x) = P_X(B_0) = 1/4, \quad 0 \leq x < 1 \\
&F_X(x) = P(X \leq 0) + P(0 < X \leq x) = F_X(0) + P_X(B_1) = 1/4 + 2/4 = 3/4, \quad 1 \leq x < 2 \\
&F_X(x) = P(X \leq 1) + P(1 < X \leq x) = F_X(1) + P_X(B_2) = 3/4 + 1/4 = 1, \quad x \geq 2
\end{aligned}$$

$$F_X = \frac{1}{4}\mathbb{1}_{[0,1[} + \frac{3}{4}\mathbb{1}_{[1,2[} + \mathbb{1}_{[2,\infty[}$$

In order to define an application on Ω in Python, we must first create a function or a *lambda expression* that associates a value to each elementary event ω of Ω, then we apply this association to Ω using the *map(function, iterator)* function and the result will be matched to the *zip()* function. Finally, we create a dictionary for the final result with the *dict()* function.
From *utils* we import the functions: *createFiniteRV* which creates the dictionary of a finite r.v according to the steps described in the first paragraph, *getCDF()* allows to define the cdf from a dictionary of the distribution of a finite r.v and *getInversedFiniteRV()* returns the inverse image of this r.v. The function *plot_function()* plots the cdf using the *matplotlib* library and its *pyplot* module. To plot the probability distribution we use the function *bar()* and *step()* for the cdf.

Let's code!

```
# utils.py (Continuation)

# sort_PDF: returns a key sorted distribution of the argument
def sort_PDF(prob_RV):
    return {key:prob_RV[key] for key in sorted(prob_RV.keys())}

# zip : create an iterator that aggregate a collection's elements
# map : applies a function to the elements of a list.
# create_FiniteRV: Create a distribution by mapping a function on Omega's elements
def create_FiniteRV(Omega, map_X):
    return dict(zip(Omega, map(map_X, Omega)))

# get_InversedFiniteRV: inverses a finite RV distribution (value:key)
def get_InversedFiniteRV(finiteRV):
    return {v:{i for i in finiteRV.keys() if finiteRV[i] == v } for k,v in finiteRV.items()}

# get_PMF: returns the probability distribution P_X of X
def get_PMF(finiteRV, probability_Omega):
    inv_X =  get_InversedFiniteRV(finiteRV)
    prob_values = list(map(sum,
        [[probability_Omega[omega] for omega in event] for event in inv_X.values()]))
    return dict(zip(inv_X.keys(), prob_values))
```

```
# accumulate : returns the reduced result by applying the given operation on a set
# get_CDF: cumulative distribution function CDF of X
def get_CDF(prob_RV):
    sprob_RV =sort_PDF(prob_RV)# {key:prob_RV[key] for key in sorted(prob_RV.keys())}
    return sprob_RV,dict(zip(sprob_RV.keys(), list(accumulate(sprob_RV.values()))))

# plot_Pdf_Cdf: plots PDF and CDF of RV X
def plot_Pdf_Cdf(pdf0, cdf0, choice =None):
    first  = list(pdf0.keys())[0]
    last   = list(pdf0.keys())[-1]
    keys   = [first-1] + list(pdf0.keys()) + [last+1]
    pvalues = [0.0]+ list(pdf0.values())

    fig = plt.figure()
    ncols0 =  2 if choice == None else 1
    axes = fig.subplots(nrows=1, ncols=ncols0)
    axes[0].bar(keys, pvalues + [0.0], width=0.05)

    if not choice:
        cvalues = [0.0]+ list(cdf0.values())
        axes[1].step(keys, [0.0]+ cvalues)

    plt.show()
```

py: *sympy.stats.cdf (RExpr, ..)* gives the cdf of the r.v RExpr
RExpr: is a random symbolic expression that represents a r.v function of another r.v.
sympy.stats.Density (RExpr, ..) returns an object that represents the pmf of a d.r.v. Its dict property gives the corresponding dictionary.
sympy.stats.FiniteRV (name, Density, ..) returns an object which represents finite d.r.v giving its pmf in a dictionary form.

Let’s code!

```
#Code201.py

import sys;sys.path.append(’../lib’)
from utils import (set_Power, create_FiniteRV, get_InversedFiniteRV,get_PMF,
                    get_CDF,plot_Pdf_Cdf)

# Random Variable Implementation : Encapsulate the logic of RV X. p:Tail, f:head
distribution = {(’f’,’p’):1/4, (’p’,’f’):1/4, (’f’,’f’):1/4, (’p’,’p’):1/4}

Omega = set_Power({’p’,’f’}, 2)           ;print(’Omega         : ’,Omega)
prob_Omega = distribution                 ;print(’ProbaOmega    : ’,prob_Omega)

#map_X: counts the number of tails
map_X = lambda a : a.count(’p’)
rv_X  = create_FiniteRV(Omega, map_X)     ;print(’RV dictionary : ’,rv_X)
rng_X = set(rv_X.values())                ;print(’RV Range      : ’,rng_X)
inv_X = get_InversedFiniteRV(rv_X)        ;print(’inversed RV   : ’,inv_X)
pdf_X   = get_PMF(rv_X, prob_Omega)       ;print(’P_X RVProbLaw : ’,pdf_X)
pdf_X, cdf_X = get_CDF(pdf_X)             ;print(’CDF of X      : ’,cdf_X)

plot_Pdf_Cdf(pdf_X, cdf_X)

#______________________________   Output   ______________________________________
# Omega         :  {(’f’, ’p’), (’p’, ’p’), (’f’, ’f’), (’p’, ’f’)}
# ProbaOmega    :  {(’f’, ’p’): 0.25, (’p’, ’f’): 0.25, (’f’, ’f’): 0.25, (’p’, ’p’): 0.25}
# RV dictionary :  {(’f’, ’p’): 1, (’p’, ’p’): 2, (’f’, ’f’): 0, (’p’, ’f’): 1}
# RV Range      :  {0, 1, 2}
# inversed RV   :  {1: {(’p’, ’f’), (’f’, ’p’)}, 2: {(’p’, ’p’)}, 0: {(’f’, ’f’)}}
# P_X RVProbLaw :  {1: 0.5, 2: 0.25, 0: 0.25}
# CDF of X      :  {0: 0.25, 1: 0.75, 2: 1.0}
```

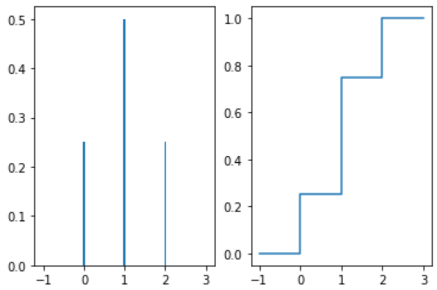


Figure 2.3: Distribution and cdf curve (code201.py)

Proposition 1. *The properties of a cdf are:*
*1- **Positive** and **bounded** on the unit:* $0 \leq F_X(x) \leq 1$
*2- **Monotone non-decreasing**:* $x_1 < x_2 \implies F_X(x_1) < F_X(x_2)$
*3- **Limits**:* $lim_{x \to +\infty} F_X(x) = F_X(\infty) = 1, \quad lim_{x \to -\infty} F_X(x) = F_X(-\infty) = 0$
*4- **Right continuous**:* $lim_{x \to a^+} F_X(x) = F_X(a^+) = F_X(a)$
5- $P_X(a < x \leq b) = F_X(b) - F_X(a), \quad P_X(X > a) = 1 - F_X(a), \quad F_X(b^-) = P_X(X < b)$

2.2 Discrete random variables (d.r.v)

In this section we deal with the case where the rank of the r.v is countable. In this case it is said to be *discrete*.

Definition 4.
A r.v X is discrete (d.r.v) if its rank R_X is a finite or infinite countable set. i.e. $R_X = \{x_i \in \mathbb{R}\}_{i \in I}$ and I a finite or infinite countable part of $\mathbb{N}$. In this case the law of X: P_X is a discrete law.

In addition to its probability law and its distribution function, it can be defined by the individual probabilities of the elementary events.

Definition 5.
Let E be a set, we call function $f : E \longrightarrow \mathbb{R}$ a mass function if its domain of definition D_f is finite or infinite countable and it satisfies the following properties:
- $\forall e \in D_f, 0 \leq f(e) \leq 1$
- $\sum_{e_i \in D_f} f(e_i) = 1$

Let X be a d.r.v, the function p_X defined from R_X to $[0,1]$ which to all $x_i \in R_X$ associates the probability of $\{X = x_i\}$:

$$\begin{aligned} p_X : R_X &\longrightarrow [0,1] \\ x_i &\longmapsto p_X(x_i) = P_X(X = x_i) = p_i \end{aligned}$$

is called the probability mass function (pmf) of X and denoted p_X.

The diagram below shows the composition of the application X and p to define the law of X (p_X) in discrete case.

$$\begin{tikzcd} \Omega \arrow[r, "p"] \arrow[d, "X"'] & {[0,1]} \\ R_X \arrow[ur, "p_X"] & \end{tikzcd}$$

If X is a d.r.v, then we can calculate:

$$\begin{aligned} P_X(\{x_i\}) &= P(\{X = x_i\}) = P(X = x_i) = p_X(x_i), \qquad \forall x_i \in R_X \\ P_X(B) &= P(X \in B) = P\left(\bigcup_{x_i \in B \cap R_X} \{X = x_i\} \right) \\ &= \sum_{x_i \in B \cap R_X} P(\{X = x_i\}) = \sum_{x_i \in B \cap R_X} P(\{X = x_i\}) = \sum_{x_i \in B \cap R_X} p_X(x_i) \\ F_X(x) &= P(X \leq x) = \sum_{x_i \in R_X \& x_i \leq x} P(X = x_i) = \sum_{x_i \in R_X \& x_i \leq x} p_X(x_i) \end{aligned}$$

In case X is finite such that $R_X = \{x_0, x_1, \cdots, x_n\}$ and $x_0 \leq x_1 \leq \cdots \leq x_n$, we have :

$$\begin{aligned}
\forall x \in [-\infty, x_0[, &\quad F_X(x) = 0 \\
\forall x \in [x_i, x_{i+1}[, &\quad F_X(x) = F_X(x_{i-1}) + p_X(x_i) = F_X(x_{i-1}) + p_X(x_i)\mathbb{1}_{[x_i, x_{i+1}[}(x) \\
\forall x \in [x_n, \infty[, &\quad F_X(x) = 1
\end{aligned}$$

Example 3. *To prepare an exam, a teacher randomly chooses two exercises out of 10, four of them are difficult to solve. Let the r.v X be the number of difficult exercises chosen for the exam. What is the mass function of X ?*

Solution
Let E (resp. E^c) be the set of easy (respectively difficult) exercises. The experiment consists in choosing two exercises among $E \cup E^c$. Since the result is unordered and without repetition, then the space Ω is:

$$\Omega = \{\{e_1, e_2\} | (e_1, e_2) \in (E \cup E^c)^2 \text{ and } e_1 \neq e_2\}$$

the number of possibilities is a combination of 2 among 10 (C_{10}^2). The r.v X is defined by:

$$\begin{aligned}
X : \Omega &\longrightarrow \mathbb{R} \\
\{e_1, e_2\} &\longmapsto x = X(\{e_1, e_2\}) = \text{Nb\_Difficult\_Exercises}(\{e_1, e_2\}) = \mathbb{1}_{E^c}(e_1) + \mathbb{1}_{E^c}(e_2)
\end{aligned}$$

$$\begin{aligned}
R_X &= \{0, 1, 2\} \\
p_X(k) &= P(X = k), \quad \forall k \in R_X \\
p_X(0) &= P(\{\{e_1, e_2\} | (e_1, e_2) \in E^2 \text{ and } e_1 \neq e_2\}) = \frac{C_6^2}{C_{10}^2} = \frac{1}{3} \\
p_X(1) &= P(\{\{e_1, e_2\} | ((e_1 \in E \text{ and } e_2 \in E^c) \text{ or } (e_2 \in E \text{ and } e_1 \in E^c)) \text{ and } e_1 \neq e_2\}) = \frac{C_6^1 C_4^1}{C_{10}^2} = \frac{8}{15} \\
p_X(2) &= P(\{\{e_1, e_2\} | e_1 \in E^c \text{ and } e_2 \in E^c \text{ and } e_1 \neq e_2\}) = \frac{C_4^2}{C_{10}^2} = \frac{2}{15} \\
p_X(x) &= 0 \text{ , } \forall x \notin R_X
\end{aligned}$$

$$p_X = \frac{1}{3}\mathbb{1}_{\{0\}} + \frac{8}{15}\mathbb{1}_{\{1\}} + \frac{2}{15}\mathbb{1}_{\{2\}}$$

Example 4. *Let Ω be the sample space of a given random experiment and X an associated d.r.v whose rank is $R_X = \{1, 2, 3\}$ and its pmf:*

$$p_X = \frac{1}{2}\mathbb{1}_{\{1\}} + \frac{1}{3}\mathbb{1}_{\{2\}} + \frac{1}{6}\mathbb{1}_{\{3\}}$$

1- Give its cumulative distribution function F_X.
2- Write the Python code which models X and plot its pmf and cdf functions.

Solution
1- The cdf is given by: $F_X = \frac{1}{2}\mathbb{1}_{[1,2[} + \frac{5}{6}\mathbb{1}_{[2,3[} + \mathbb{1}_{[3,\infty[}$
2- The following code models X, its pmf and its cdf.

Let’s code!

```
#Code202.py

import sys;sys.path.append('../lib')
from utils import get_CDF, plot_Pdf_Cdf, get_round_dic

#dic1: defines a r.v on Omega={o1.o2.o3}, dic2: defines the associated probabilities
dic1,dic2={'o1':1,'o2':2,'o3':3 }, {1:0.5,2:0.33,3:0.17 }

rv_X  = dict(dic1)                      ;print('RV map        : ',rv_X)
rng_X = set(rv_X.values())              ;print('RV Range      : ',rng_X)
```

```
pdf_X = get_round_dic(dict(dic2))        ;print('P_X RVProbLaw : ',pdf_X)
pdf_X, cdf_X = get_CDF(pdf_X)            ;print('Cdf of X      : ',get_round_dic(cdf_X))

plot_Pdf_Cdf(pdf_X, cdf_X)

#______________________________   Output  ______________________________________
# RV map        :  {'o1': 1, 'o2': 2, 'o3': 3}
# RV Range      :  {1, 2, 3}
# P_X RVProbLaw :  {1: 0.5, 2: 0.33, 3: 0.17}
# Cdf of X      :  {1: 0.5, 2: 0.83, 3: 1.0}
```

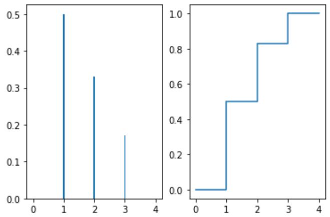


Figure 2.4: Curve of the pmf and cdf (code202.py)

The following code uses the predefined functions in *sympy.stats: FiniteRV, density, cdf* for the r.v whose rank is $\{0, 1, 2, 3\}$ and the following pmf:

$$p_X = \frac{1}{10}\mathbb{1}_{\{0\}} + \frac{2}{10}\mathbb{1}_{\{1\}} + \frac{3}{10}\mathbb{1}_{\{2\}} + \frac{4}{10}\mathbb{1}_{\{3\}}$$

Let's code!

```
#Code203,py

from sympy.stats import FiniteRV, density, cdf
import sys;sys.path.append('../lib')
from utils import get_CDF,plot_Pdf_Cdf, get_round_dic

# distribution of r.v X
pmf = {0:.1,1:0.2,2:.3,3:.4}

rv_X = FiniteRV('X',pmf)                     ;print('RV Range : ',set(pmf.keys()))
pdf_X = get_round_dic(density(rv_X).dict)    ;print('pdf_X    : ',pdf_X)
cdf_X = get_round_dic(cdf(rv_X))             ;print('cdf_X    : ',cdf_X )
spdf_X, cdf_X = get_CDF(pdf_X)               ;print('Cdf of X : ',get_round_dic(cdf_X))

plot_Pdf_Cdf(spdf_X, cdf_X)

#______________________________   Output  ______________________________________
# RV Range :  {0, 1, 2, 3}
# pdf_X    :  {0: 0.1, 1: 0.2, 2: 0.3, 3: 0.4}
# cdf_X    :  {0: 0.1, 1: 0.3, 2: 0.6, 3: 1.0}
# Cdf of X :  {0: 0.1, 1: 0.3, 2: 0.6, 3: 1.0}
```

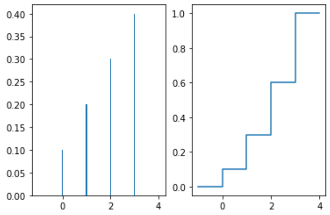


Figure 2.5: Curve of the pmf and cdf (code203.py)

2.3 Continuous random variables (c.r.v)

Unlike a discrete r.v, the rank of a continuous r.v is uncountable. This is the case for example of a battery lifetime or the time until the arrival of the next earthquake or the amount of

precipitation during a given period.
For the continuous case, we speak of the probability that the r.v takes a value in a given interval instead of the probability of being equal to a given value. So for a c.r.v $P(X = x) = 0\ \forall x \in R$ because by taking finer and finer units of measurement, the probability that the r.v is exactly equal to x becomes zero.
For this reason, we cannot speak of a probability mass function in the continuous case, but rather of a probability density function which is the limit (when it exists) of the probability that X takes a value in the interval $]x, x + \Delta x]$ divided by the length of this interval (which is Δx) when $\Delta x \longrightarrow 0$:

$$f_X(x) = \lim_{\Delta x \to 0} \frac{P_X(X \in]x, x + \Delta x])}{\Delta x} = \lim_{\Delta x \to 0} \frac{F_X(x + \Delta x) - F_X(x)}{\Delta x} = \frac{dF_X(x)}{dx} = F'_X(x)$$

if $F_X(x)$ is differentiable in x.
$f(x)$ in fact measures the probability that X is close to x and not equal to x .

Definition 6.
Let X be a r.v and its corresponding distribution function F_X. X is continuous if its rank R_X is an uncountable set.
A r.v X admits a density if the following limit exists:

$$f_X(x) = \lim_{\Delta x \to 0} \frac{P_X(X \in]x, x + \Delta x])}{\Delta x} = \lim_{\Delta x \to 0} \frac{F_X(x + \Delta x) - F_X(x)}{\Delta x}$$

In this case f_X is called the probability density function of X and F_X is differentiable and its derivative is f_X ($f_X(x) = F'_X(x) = \frac{dF_X(x)}{dx}$).
The probability distribution of X is defined for any set B of real numbers, by the following relationship:

$$P_X(X \in B) = \int_B dP = \int_{x \in B} f_X(x)dx$$

If $B = [a, b]$ then $P(X \in B) = \int_a^b f_X(x)dx$.
If $a = b$, then $P(X \in B) = \int_a^a f_X(x)dx = 0$.
This explains that: $P(X = x) = 0\ \forall x \in R$.
f_X satisfies the following properties:

- Its domain of definition D_f is **uncountable**.
- **Positive**: $\forall x \in D_f, f_X(x) \geq 0$
- **Integrable** and **normed**: $\int_{D_f} f_X(x)dx = 1$

Example 5. *Let X be a c.r.v and f is a function defined by:*
$f_X(x) = ce^{-x/2}$ for $x \geq 0$ and 0 otherwise. c is a positive constant ($f_X(x) = ce^{-x/2}\mathbb{1}_{x\geq 0}$).

1- Find the value of c so that f is a density function.
2- Find the cumulative distribution function F_X.
3- Find $P_X(2 < X < 4)$.

Solution

1. The function f is a density function if it verifies :
   a. $D_f = \mathbb{R}^+$
   b. $f(x) \geq 0, \forall c > 0$.
   c. Integrable and normed over D_f if :
   $\int_{-\infty}^{+\infty} f(x)dx = 1$ so :

$$\int_{-\infty}^{+\infty} f(x)dx = \int_0^{\infty} ce^{-x/2}dx = 2c = 1 \quad \text{so } c = 1/2$$

2. The cumulative distribution function F_X:

$$F_X(x) = \int_{-\infty}^{x} f(x)dx = \int_0^x \frac{1}{2}e^{-x/2}dx = \frac{1}{2}(1 - e^{-x/2})$$

3. The probability of $2 < X < 4$:

$$P_X(2 < X < 4) = F(4) - F(2) = \frac{e^{-1} - e^{-2}}{2}$$

2.4 Expectation and Variance

2.4.1 Expectation

In the case of a d.r.v, the expectation of X is the weighted average of the possible values that X can take; each value is weighted by the probability that X takes that value. On the other hand, if X is a c.r.v, the expectation is the area delimited by the curve of X times its density.

Definition 7.
Let X be a r.v, the expectation of X denoted by $\mathbb{E}(X)$ is defined in the general case by:

$$\mathbb{E}(X) = \int_B X dP$$

1- If X is a d.r.v and p_X its pmf, then:

$$\mathbb{E}(X) = \sum_{x_i \in R_X} x_i p_X(x_i)$$

2- If X is a c.r.v and f_X its pdf, then:

$$\mathbb{E}(X) = \int_{R_X} x f_X(x) dx$$

Proposition 2. *Properties of Expectation*
*1- **Constant** : if X is constant (its value is c), then $\mathbb{E}(X) = c$.*
*2- **Indicator**: $\forall A \in \mathcal{F}, \mathbb{E}(\mathbb{1}_A) = P(A)$*
*3- **Linearity**: $\mathbb{E}(aX + bY) = a\mathbb{E}(X) + b\mathbb{E}(Y)$*
*4- **Monotony** : $X < Y(or P(X < Y) = 1) \implies \mathbb{E}(X) < \mathbb{E}(Y)$*
*5- **Function** : $\phi : \mathbb{R} \longrightarrow \mathbb{R}$ (an increasing function):*
Discrete case : $\mathbb{E}(\phi(X)) = \sum_{R_X} \phi(x) p_X(x)$
Continuous case: $\mathbb{E}(\phi(X)) = \int_{R_X} \phi(x) f_X(x) dx$

2.4.2 Variance

The variance measures the dispersion of the values of X with respect to its expectation. A large value of the variance means that X takes values far from the expectation, so the distribution is very dispersed; on the other hand, a small value indicates that the distribution is concentrated around its mean.

Definition 8.
The variance of X denoted $\mathbb{V}(X)$ is the expectation of the squares of the deviations of X from its mean if it is defined.

$$\mathbb{V}(X) = \mathbb{E}((X - \mathbb{E}(X))^2), \text{ if it exists}$$

The square root of $\mathbb{V}$, $\sigma_X = \sigma(X) = \sqrt{\mathbb{V}(X)}$ is called the standard deviation of X.

Note that the variance has a different unit than that of X. If for example X is in meters, the variance is in square meters; for this reason, we define *standard deviation* which is simply the square root of the variance.

Proposition 3. *Variance Properties*
*1- **Constant:** if X is constant (its value is c), then $\mathbb{V}(X) = 0$.*
*2- **Positive:** $\mathbb{V}(X) \geq 0$*
3- $\mathbb{V}(X) = \mathbb{E}(X^2) - \mathbb{E}^2(X)$

Example 6. *If the pmf of a d.r.v X is given by:*

$$p_X = \frac{2}{3}\mathbb{1}_{\{1\}} + \frac{1}{3}\mathbb{1}_{\{2\}}$$

1- Calculate its expectation and its variance.
2- Write the Python code which models this r.v.

Solution

$\mathbb{E}(X) = \sum_{x_i \in R_X} x_i p_X(x_i) = 1 \times 2/3 + 2 \times 1/3 = 4/3.$
$\mathbb{E}(X^2) = \sum_{x_i \in R_X} x_i^2 p_X(x_i) = 1 \times 2/3 + 4 \times 1/3 = 2.$
$\mathbb{V}(X) = \mathbb{E}(X^2) - \mathbb{E}^2(X) = 2 - 16/9 = 2/9.$

py: *sympy.stats.E(RExpr, ..)* returns a symbolic expression which represents the expectation of the given RExpr.
py: *sympy.stats.variance (RExpr, ..)* returns a symbolic expression which represents variance of the given RExpr.

Let's code!

```
#Code204,py

from sympy.stats import FiniteRV, density, cdf, E, variance
import sys;sys.path.append('../lib')
from utils import get_CDF,plot_Pdf_Cdf, get_round_dic,plt

pmf = {1:0.66,2:0.34}

rv_X = FiniteRV('X',pmf)              ;print('RV Range : ',set(pmf.keys()) )
pdf_X = density(rv_X).dict            ;print('pdf_X    : ',pdf_X)
cdf_X = get_round_dic(cdf(rv_X))      ;print('cdf_X    : ',cdf_X)
spdf_X, cdf_X = get_CDF(pdf_X)        ;print('Cdf of X : ',get_round_dic(cdf_X))
E_X, V_X = E(rv_X), variance(rv_X)    ;print('E(X)     : ',round(E_X,2), ', V_X : ', round(V_X,2))

plot_Pdf_Cdf(spdf_X, cdf_X, E_X)

#______________________________   Output  ______________________________________
# RV Range :  {1, 2}
# pdf_X    :  {1: 0.66, 2: 0.34}
# cdf_X    :  {1: 0.66, 2: 1.0}
# Cdf of X :  {1: 0.66, 2: 1.0}
# E(X)     :  1.34 , V_X :  0.22
```

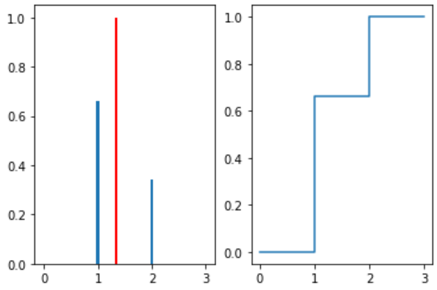


Figure 2.6: Curve of the pmf and the cdf (code204.py) in red: the expectation

2.4.3 Probability inequalities

There are several important inequalities in probability. The Markov and Chebychev inequalities are at the base of several other inequalities, they allow us to have bounds on probabilities by knowing only the expectation and/or the variance of the probability law.

Proposition 4. *Inequalities between $P, \mathbb{E}$ and $\mathbb{V}$:*
- *1- **Cauchy-Schwartz:***

$$\mathbb{E}(XY)^2 \leq \mathbb{E}(X^2)\mathbb{E}(Y^2)$$

- *2- **Markov:** Let X be a non-negative r.v and ϕ an increasing function,*

$$P(\phi(X) \geq t) \leq \frac{\mathbb{E}(\phi(X))}{t}$$

- *3- **Chebychev:***

$$P(|X - \mathbb{E}(X)| \geq c) \leq \frac{\mathbb{V}(X)}{c^2}$$

- *4- **Jensen:** Let ϕ be a convex function:*

$$\mathbb{E}(\phi(X)) \geq \phi(\mathbb{E}(X))$$

Proof

2- **Markov inequality** (For the case $\phi = \mathbb{I}$ is the identity map)

Let a be a given value
Consider the indicator $\mathbb{1}_{X \geq a}$ r.v.
$\mathbb{E}(\mathbb{1}_{X \geq a}) = P(X \geq a)$
$X \geq a\mathbb{1}_{X \geq a}$ since X is non-negative
$\mathbb{E}(X) \geq a\mathbb{E}(\mathbb{1}_{X \geq a}) = aP(X \geq a)$

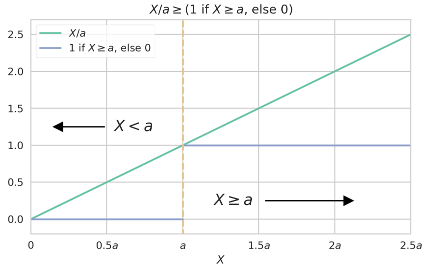


Figure 2.7: Markov inequality

3- **Chebychev inequality**

$(X - \mathbb{E}(X))^2$ is a non-negative r.v

$P((X - \mathbb{E}(X))^2 \geq c^2) \leq \dfrac{E((X - \mathbb{E}(X))^2)}{c^2}$, we apply the inequality of Markov for $a = c^2$

$P(|X - \mathbb{E}(X)| \geq c) \leq \dfrac{E((X - \mathbb{E}(X))^2)}{c^2} = \dfrac{\mathbb{V}(X)}{c^2}$

Example 7. *If the number of requests processed by a server is a r.v X with an average of 500r/min. Find a limit to the probability that it processes at least 1000 requests in one minute. If the variance is 100, find a limit to the probability that it processes between 400 and 600 requests in a minute.*

Solution

1. By the Markov inequality:

$$P(X \geq 1000) \leq \frac{500}{1000} = \frac{1}{2}$$

2. By Chebychev inequality:

$$P(|X - 500| \geq 100) \leq \frac{\sigma^2}{100^2} = \frac{1}{100}, \text{ so } P(|X - 500| < 100) \geq 1 - \frac{1}{100} = \frac{99}{100} = 0.99$$

Example 8. *Three roommates want to decide who should cook dinner. They all flip a fair coin at the same time, they repeat the experiment until one of them gets a different result than the two others. What is the expectation of the number of required flips.*

Solution
Let X be a r.v that gives the number of flips until a different value from the two others is obtained. Its rank $R_X = \mathbb{N}^*$
For each i^{th} flip, we have the universe $\Omega_i = \{P, F\}^3$
Let L_i be the event that the i^{th} flip gives different results (L_i^c similar results).
$L_i^c = \{(P, P, P), (F, F, F)\}$
$P(L_i) = 1 - P(L_i^c) = \frac{3}{4} = p$
The realization of event $X = k$ corresponds to having similar results (all heads or all tails) at the first $k - 1$ flips and a different result at the k^{th} flip.
$(X = k) = (\cap_{i=1}^{k-1} L_i^c) \cap L_k$(intersection of disjoint events) therefore:

$$p_X(k) = P(X = k) = P((\cap_{i=1}^{k-1} L_i^c) \cap L_k) = (1 - p)^{k-1}p, \quad \forall k \in \mathbb{N}^*$$
$$\mathbb{E}(X) = \sum_{k=1}^{\infty} kp_X(k) = \sum_{k=1}^{\infty} k(1 - p)^{k-1}p = \frac{p}{(1 - (1 - p))^2} = \frac{1}{p}$$

we could say that X counts the first success (getting different outcome), it is a geometric r.v. Therefore its mean is $1/p$.

Example 9. *Let X be a d.r.v of rank $R_X = \{-1, 0, 1, 2, 3\}$, its pmf $p_X(k) = 1/5$ for $k \in R_X$ and let $Y = 2|X|$.*
1- Find the rank and the pmf of Y.
2- Write the Python code that models this situation.

Solution
1-$R_Y = \{2|x| \text{ s.t } x \in R_X\} = \{0, 2, 4, 6\}$, its pmf must be defined $\forall k \in R_Y$.

$$\begin{aligned}
p_Y(k) &= P(Y = k) = P(\{2|x| = k \text{ s.t } x \in R_X\}) = P(\{|x| = [k/2] \text{ s.t } x \in R_X\}) \\
&= P(\{x = [k/2] \text{ or } x = -[k/2] \text{ s.t } x \in R_X\}) \\
p_Y(0) &= P(X = 0) = 1/5 \\
p_Y(2) &= P(X \in \{-1, 1\}) = P(X = -1) + P(X = 1) = 2/5 \\
p_Y(4) &= P(X = 2) = 1/5 \\
p_Y(6) &= P(X = 3) = 1/5
\end{aligned}$$

Let’s code!

```
#Code205,py

from sympy.stats import FiniteRV, density, cdf, E, variance
from sympy.functions import Abs
import sys;sys.path.append('../lib')
from utils import get_CDF,plot_Pdf_Cdf, get_round_dic

# RV X
pmf = {-1:0.2,0:0.2,1:0.2,2:0.2,3:0.2}
rv_X = FiniteRV('X',pmf)

# RV Y=2|X|
rv_Y = 2 * Abs(rv_X)
pdf_Y = get_round_dic(density(rv_Y).dict)  ;print('pdf_Y   : ',pdf_Y)
rng_Y = set(pdf_Y.keys())                  ;print('Range_Y : ',rng_Y)
cdf_Y =get_round_dic(cdf(rv_Y))            ;print('cdf_Y   : ',cdf_Y )
spdf_Y, cdf_Y = get_CDF(pdf_Y)             ;print('cdf_Y   : ',get_round_dic(cdf_Y))
E_Y = round(E(rv_Y),2)
V_Y =  round(variance(rv_Y),2)             ;print('E(Y)    : ', E_Y, ', V(Y):', V_Y)

plot_Pdf_Cdf(spdf_Y, cdf_Y, E_Y)

#_______________________________   Output  _______________________________________
# pdf_Y   :  {2: 0.4, 0: 0.2, 4: 0.2, 6: 0.2}
# Range_Y :  {0, 2, 4, 6}
# cdf_Y   :  {0: 0.2, 2: 0.6, 4: 0.8, 6: 1.0}
# cdf_Y   :  {0: 0.2, 2: 0.6, 4: 0.8, 6: 1.0}
# E(Y)    :  2.80 , V(Y): 4.16
```

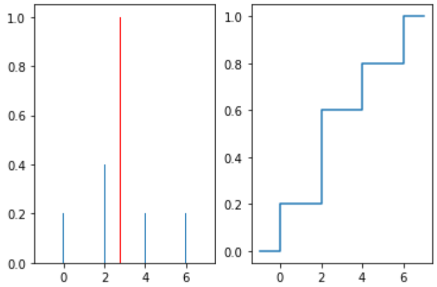


Figure 2.8: Curve of pmf and cdf (code205.py)

Example 10. *Consider the following function* f : $f(x) = \frac{4}{x^5}\mathbb{1}_{\{X\geq 1\}}(x)$
1- Prove that f *is a density function.*
2- Let X *be a c.r.v whose pdf is* $f_X = f$*, find* $\mathbb{E}(X), \mathbb{V}(X)$*,*

Solution
1-f is integrable and its integral:

$$\int_{-\infty}^{+\infty} f_X(t)dt = \int_{-\infty}^{\infty} \frac{4}{t^5}\mathbb{1}_{\{X\geq 1\}}(t)dt = \left[(-t^{-4})\right]_1^{\infty} = 1$$

$f(t)$ is continuous over $\mathbb{R}^*$, positive and the area bounded by its curve is equal to the unit, so it can be a pdf for a r.v.

2-

$$\mathbb{E}(X) = \int_{-\infty}^{+\infty} x f_X(x)dx = \int_{1}^{+\infty} \frac{4}{x^{-4}}dx = \left[-\frac{4}{3}x^3\right]_1^{\infty} = \frac{4}{3}$$

$$\mathbb{E}(X^2) = \int_{-\infty}^{+\infty} x^2 f_X(x)dx = \int_{1}^{+\infty} \frac{4}{x^3}dx = \left[-2x^{-2}\right]_1^{\infty} = 2$$

$$\mathbb{V}(X) = \mathbb{E}(X^2) - \mathbb{E}^2(X) = 2 - \frac{16}{9} = \frac{2}{9}$$

Let's code!
The following code shows how to use symbolic calculation to verify that a given function is a probability density.

```
#Code206.py

# Symbol is a class that allows creating algebric expressions
# integrate(f,D) function calculates the integral of f over domain D
from sympy import Symbol, oo, Piecewise, integrate
from sympy.plotting import plot
from sympy.stats import ContinuousRV, density, E, variance,cdf

# symbols
x,t = Symbol('x'), Symbol('t')

# function f: checks if it is density, its integral has cdf properties
f    = Piecewise((4/x**5, x >= 1), (0, True))    ;print("f(x) = ",f)
val  = integrate(f,(x,-oo,+oo))                  ;print("f is normed ? surface = ", val)
F    = integrate(f,(x,-oo,t))                    ;print("F(t) = ", F)
print("Lim x->-oo F(x)= ", F.subs(t,-oo),"Lim x->+oo F(x)= ", F.subs(t,+oo))

# create RV X with f as PDF
X     = ContinuousRV(symbol=x, density= f)       ;print("PDF_X = ",density(X)(t))
cdf_X = cdf(X)(t)                                ;print("CDF_X = ",cdf_X)

EX2   = integrate(x**2 * f)
print("E(X)=",E(X), ",E(X^2)=",EX2, "=", E(X**2),",Var(X)=", variance(X))

# plot f and F
plot(f, adaptive=False, nb_of_points=400)
plot(F, adaptive=False, nb_of_points=400)

#______________________________ Output ______________________________________
# f(x) =  Piecewise((4/x**5, x >= 1), (0, True))
# F(t) =  Min(1, t)**(-4) - 1/t**4
# Lim x->-oo F(x)=  0 Lim x->+oo F(x)=  1
# f is normed ? surface =  1
# PDF_X =  Piecewise((Piecewise((4/z**5, z >= 1), (0, True)), (z >= -oo) & (z < oo)), (0, True))
# CDF_X =  Min(1, z)**(-4) - 1/z**4
# E(X)= 4/3 ,E(X^2)= Piecewise((0, x <= 1), (2 - 2/x**2, True)) = 2 ,Var(X)= 2/9
```

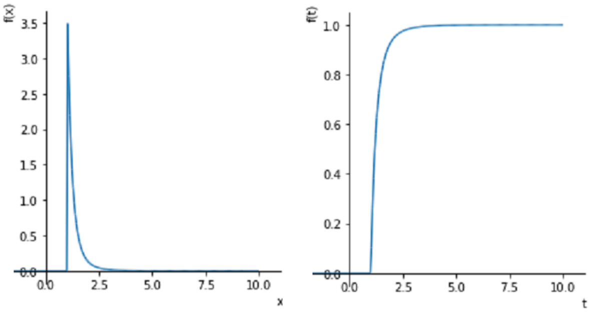
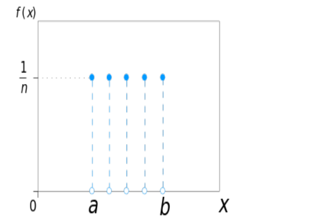
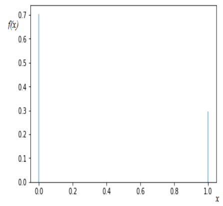
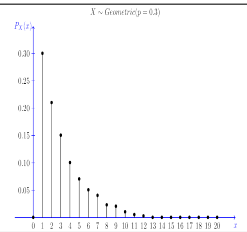
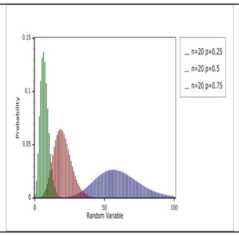
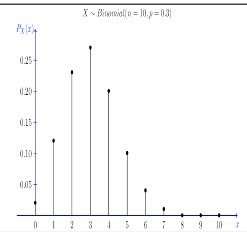
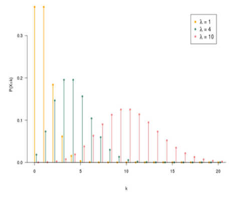


Figure 2.9: Curve of pdf and cdf (code206.py)

2.5 Usual distributions

2.5.1 Discrete distributions

1- **Uniform** $\mathcal{U}(a,b)$: represents a RE with a finite sample space whose elements are all equally likely in the interval$[a,b]$.
2- **Bernoulli** $\mathcal{B}er(p)$: represents a RE with a sample space of only two results, often called *success* and *failure*, parameterized with p which represents the probability of *success* (and therefore $1-p$ is the probability of *failure*).
3- **Geometric** $\mathcal{G}eo(p)$: represents a RE with an infinite sample space whose elements represent the appearance of the first success (have $k-1$ failures followed by a success (k^{th}) for a sequence of Bernoulli trials $\mathcal{B}er(p)$).
4- **Pascal** $\mathcal{P}as(n,p)$: represents the number of necessary repetitions of the Bernoulli test (with probability of success p) until n success.
5- **Binomial** $\mathcal{B}in(n,p)$: represents the total number of successes obtained from n repetitions of Bernoulli trials $\mathcal{B}er(p)$.
6- **Poisson** $\mathcal{P}ois(\lambda)$: represents a RE which counts the number of occurrences of a certain event in a time interval.

Law X	Uniform	Bernouli	Geometric	Pascal	Binomial	Fish
$X \rightsquigarrow$	$\mathcal{U}ni(a,b)$	$\mathcal{B}er(p)$	$\mathcal{G}eo(p)$	$\mathcal{P}as(n,p)$	$\mathcal{B}in(n,p)$	$\mathcal{P}ois(\lambda)$
Param	$a,b \in \mathbb{N}$	$p \in [0,1]$	$p \in [0,1]$	$n \in \mathbb{N}^*$, $p \in [0,1]$	$n \in \mathbb{N}^*$, $p \in [0,1]$	$\lambda \in \mathbb{R}^*$
Rank	$\{a,\cdots,b\}$	$\{0.1\}$	$\mathbb{N}^*$	$\mathbb{N}^*$	$\{0,...,n\}$	$\mathbb{N}$
$p_X(k)$	$\dfrac{1}{b-a+1}$	$p^k(1-p)^{1-k}$	$(1-p)^{k-1}p$	$C_{n-1}^{k-1}p^n(1-p)^{(k-n)}$	$C_n^k p^k(1-p)^{n-k}$	$e^{-\lambda}\lambda^k/k!$
$F_X(x)$	$\frac{[x]-a+1}{b-a+1}\mathbb{1}_{[a,b[}(x)+\mathbb{1}_{[b,\infty[}(x)$	$p\mathbb{1}_{[0,1[}(x)+\mathbb{1}_{[1,\infty[}(x)$	$1-(1-p)^{[x]}$	.	$\sum_{i=1}^{[x]} C_n^i p^i(1-p)^{n-i}$	$e^{-\lambda}\sum_{i=1}^{[x]}\frac{\lambda^i}{i!}$
$\mathbb{E}(X)$	$\dfrac{a+b}{2}$	p	$1/p$	$\frac{n(1-p)}{p}$	np	λ
$\mathbb{V}(X)$	$\dfrac{(b-a+1)^2-1}{12}$	$p(1-p)$	$(1-p)/p^2$	$\frac{n(1-p)}{p^2}$	$np(1-p)$	λ
Curve						

Let’s code!

The following code implements some discrete distributions in Python.

```
#Code207.py

from sympy import S , Symbol, symbols, Rational
from sympy.stats import density, cdf
from sympy.stats import E, variance as V
from sympy.stats import (FiniteRV, Die, Coin, DiscreteUniform,  Bernoulli,
                        Binomial, Geometric, Poisson, Hypergeometric)

# I- Finite Random Var List
myDensity = {0: .1, 1: .2, 2: .3, 3: .4}
p  = S.One / 5
finiteRVs = {
        "Finite R.V       :":FiniteRV(’X’, myDensity),
        "Die6             :":Die(’D6’, 6),
        "Die4             :":Die(’D4’, 4),
        "Coin Half        :":Coin(’C’),
        "Coin3/5          :":Coin(’C2’, Rational(3, 5)),
        "Discrete Uniform3:":DiscreteUniform(’X’, symbols(’a b c’)),
        "Discrete Uniform5:":DiscreteUniform(’Y’, list(range(5))),
        "Bernoulli3/4     :":Bernoulli(’X’, S(3)/4),
        "Bernoulli Half   :":Bernoulli(’X’, S.Half, ’Heads’, ’Tails’),
        "Binomial Half 4  :":Binomial(’X’, 4, S.Half),
        "Hypergeometric   :":Hypergeometric(’X’, 10, 5, 3)
    }

# Print density, Expectation and Variance of each FRV in list above.
for k,X in finiteRVs.items() :
    Pdf, Ex, Vx = density(X).dict,  E(X), V(X)
    print(k,’ \t ’, Pdf, ’ \t ’, Ex, ’ \t ’, Vx)

print("-------------------------------------------------------------")
# II- Discrete (infinite) RV List
k     = Symbol("k")
lamda = Symbol("Lambda", positive=True)               # rate = lamda

discreteRVs = {
        "Geometric:":Geometric("X", p),               # Geometric(p=1/5)
        "Poisson  :":Poisson("X", lamda)              # Poisson(lambda)
    }
print("-----------------------------------------------------------")
for key,X in discreteRVs.items() :
    Pdf, Cdf,  Ex, Vx = density(X)(k), cdf(X)(k),  E(X), V(X)  #simplify(V(X))
    print(key,’ \t ’,Pdf, ’ \t ’, Cdf, ’ \t ’, Ex, ’ \t ’, Vx)

#______________________________   Output  ______________________________________
# R.V              Distribution                         E            Variance
# Finite R.V    :{0:0.1, 1:0.2, 2:0.3, 3:0.4}            2.00        1.00
# Die6          :{1:1/6, 2:1/6, 3:1/6, 4:1/6, 5:1/6, 6:1/6} 7/2          35/12
# Die4          :{1:1/4, 2:1/4, 3:1/4, 4: 1/4}          5/2          5/4
# Coin Half     :{H:1/2, T:1/2}                         H/2+T/2     (-H/2 + T/2)**2/2 + (H/2 -
      T/2)**2/2
# Coin3/5       :{H:3/5, T:2/5}                         3*H/5+2*T/5  2*(-3*H/5 + 3*T/5)**2/5 +
      3*(2*H/5 - 2*T/5)**2/5
# Discrete Unif3:{b:1/3, c:1/3, a:1/3}                  a/3+b/3+c/3  (-a/3 - b/3 + 2*c/3)**2/3
      + (-a/3 + 2*b/3 - c/3)**2/3 + (2*a/3 - b/3 - c/3)**2/3
# Discrete Unif5:{0:1/5, 1:1/5, 2:1/5, 3: 1/5, 4: 1/5}      2             2
# Bernoulli3/4  :{0:1/4, 1:3/4}                         3/4          3/16
# Bernoulli Half:{Tails:1/2, Heads:1/2}                 Heads/2+Tails/2     (-Heads/2 + Tails
     /2)**2/2 + (Heads/2 - Tails/2)**2/2
# BinomialcHalf4:{0:1/16, 1:1/4, 2:3/8, 3:1/4, 4:1/16}      2            1
# Hypergeometric:{0:1/12, 1:5/12, 2:5/12, 3:1/12}          3/2          7/12

# -------------------------------------------------------------------------
# Geometric:(4/5)**(k - 1)/5      Piecewise((1 - 5*(4/5)**(k + 1)/4, k >= 1), (0, True))      5
      20
# Poisson  :Lambda**k*exp(-Lambda)/factorial(k)     Piecewise(((-Lambda**(-k - 1)*Lambda**(k + 1)
     *(k + 1)*exp(Lambda)*lowergamma(k + 1, Lambda)/factorial(k + 1) + exp(Lambda))*exp(-Lambda), k
      >= 0), (0, True))     Lambda      Lambda
```

In what follows we demonstrate some results given in the previous table:
Pmf of the Binomial law:

$$p_X(k) = C_n^k p^k (1-p)^{n-k}$$

The probability that a r.v X takes value k is explained by the fact that there are C_n^k sequences of n results having k successes with probability p and $n-k$ failures with probability of $1-p$ (for the result $(\omega_{i_1}, \omega_{i_2}, \cdots, \omega_{i_n})$ the positions for the k success results are the subset $\{j_1, j_2, \cdots, j_k\}$ whose elements are chosen from $\{1, 2, \cdots, n\}$). If $n = 3$ and $k = 2$, then the possible sequences are: {(success, success, failure), (success, failure, success) and (failure, success, success) } each one corresponds to a subset of two positions ({1,2}, {1,3},{2,3}) the number of these subsets is $C_3^2 = 3$ and each sequence has a probability equal to $p^2 \times (1-p)$ (2 successes with probability p and 1 failure with probability $1-p$).

Note that since the experiment behind the Binomial law is to flip n times a coin, we can think of a Binomial r.v X of parameters (n,p) as the sum of n independent Bernoulli r.vs $X_1, X_2, ..., X_n$ of parameter p. This interpretation is sometimes useful as can be seen in example 11.

Calculation of the expectation of a Geometric d.r.v
Let $X \rightsquigarrow \mathcal{G}eo(p)$, let's say $q = p - 1$

$$\begin{aligned}\mathbb{E}(X) &= \sum_{k=1}^{\infty} kq^{k-1}p = p\sum_{k=1}^{\infty}\frac{d}{dq}(q^k) = p\frac{d}{dq}(\sum_{k=1}^{\infty} q^k) \\ &= p\frac{d}{dq}(\frac{q}{1-q}) = \frac{p}{(1-q)^2} = \frac{1}{p}\end{aligned}$$

Calculation of the expectation of a Poisson d.r.v
Let $X \rightsquigarrow \mathcal{P}oi(\lambda)$

$$\begin{aligned}\mathbb{E}(X) &= \sum_{x=0}^{\infty}\frac{xe^{-\lambda}\lambda^x}{x!} = \sum_{x=1}^{\infty}\frac{e^{-\lambda}\lambda^x}{(x-1)!} \\ &= \lambda e^{-\lambda}\sum_{x=1}^{\infty}\frac{\lambda^{x-1}}{(x-1)!} = \lambda e^{-\lambda}\sum_{k=0}^{\infty}\frac{\lambda^k}{k!} = \lambda e^{-\lambda}e^{\lambda} = \lambda\end{aligned}$$

Calculation of the variance of a Uniform d.r.v
Let$X \rightsquigarrow \mathcal{U}ni(a,b)$
Let $n = b - a + 1$

$$\begin{aligned}\mathbb{V}(X) &= \mathbb{E}(X^2) - \mathbb{E}^2(X) \\ \mathbb{E}(X) &= \sum_{i=b-n+1}^{b}\frac{1}{n}i = \frac{1}{n}\sum_{b-n+1}^{b} i = \frac{n+1}{2} \\ \mathbb{E}(X^2) &= \sum_{i=b-n+1}^{b}\frac{1}{n}i^2 = \frac{1}{n}\sum_{b-n+1}^{b} i^2 = \frac{(n+1)(2n+1)}{6} \\ \mathbb{V}(X) &= \frac{(n+1)(2n+1)}{6} - \frac{(n+1)^2}{4} = \frac{n^2-1}{12}\end{aligned}$$

Example 11. *Let $X \rightsquigarrow \mathcal{B}in(n,p)$ and $Y \rightsquigarrow \mathcal{B}in(m,p)$ be two independent r.vs, and $Z = X+Y$. Find the pmf of Z.*

Solution
A simple method, consists in considering that: $Z = X + Y = \sum_{i=1}^{n} X_i + \sum_{i=1}^{m} Y_i$ such that the $X_i, Y_i \rightsquigarrow \mathcal{B}er(p)$. We can deduce that $Z \rightsquigarrow \mathcal{B}in(n+m,p)$ and therefore its pmf is:

$$P_Z(k) = C_{n+m}^k \times p^k \times (1-p)^{n+m-k}\mathbb{1}_{\{0,1,\cdots,n+m\}}(k)$$

If we look for the same result directly, it becomes more complicated. Indeed, the rank of

$R_Z = \{0, 1, 2, \cdots, n+m\}$ and using the total probability law, we have:

$$\begin{aligned}
P(Z=k) = P(X+Y=k) &= \sum_{i=0}^{n} P(X+Y=k|X=i) \times P(X=i) \\
&= \sum_{i=0}^{n} P(Y=k-i|X=i) \times P(X=i) \\
&= \sum_{i=0}^{n} P(Y=k-i) \times P(X=i) \quad \text{because X and Y are independent} \\
&= \sum_{i=0}^{n} C_m^{k-i} \times p^{k-i} \times (1-p)^{m-k+i} \times C_n^i \times p^i \times (1-p)^{n-i} \\
&= p^k \times (1-p)^{m+n-k} \sum_{i=0}^{n} C_m^{k-i} \times C_n^i \\
&= C_{n+m}^k \times p^k \times (1-p)^{n+m-k} \mathbb{1}_{\{0,1,\cdots,n+m\}}(k)
\end{aligned}$$

Let’s code!

```
#Code208.py

from sympy.stats import Binomial
from sympy import S
from sympy.stats import density

# sum2Bin: creates two binomial r.v X,Y with parameters (n,p) and (m,p) as well
# as their sum Z. W is binomial r.v with parameters (n+m, p)
def sum2Bin(n, m, p):
    X, Y, W = Binomial(’X’, n, p), Binomial(’Y’, m, p), Binomial(’X’, n+m, p)
    Z = X + Y
    return [{ ’dic’: density(X).dict, ’legend’: ’X ~ Bin(10,1/2)’},
            { ’dic’: density(Y).dict, ’legend’: ’Y ~ Bin(20,1/2)’},
            { ’dic’: density(Z).dict, ’legend’: ’Z = X+Y’},
            { ’dic’: density(W).dict, ’legend’: ’W  ~ Bin(30,1/2)’}]

# plotter: plots X,Y,Z and W
def plotter(rv_infos):
    import matplotlib.pyplot as plt
    plt.figure(figsize=(12, 8))
    for i in range(len(rv_infos)):
        plt.subplot(221 + i).set_title(rv_infos[i][’legend’])
        plt.bar(list(rv_infos[i][’dic’].keys()), rv_infos[i][’dic’].values())
    plt.show()
#
s2b=sum2Bin(4, 8, S.Half)

#  prints associated distribution of X,Y,Z and W.
for i in range(len(s2b)):
    print("Pmf de " + s2b[i][’legend’][0] + ’ : ’, s2b[i][’dic’])
plotter(s2b)
#______________________________   Output   _____________________________________
# Pmf de X :  {0: 1/16, 1: 1/4, 2: 3/8, 3: 1/4, 4: 1/16}
# Pmf de Y :  {0: 1/256, 1: 1/32, 2: 7/64, ..., 8: 1/256}
# Pmf de Z :  {0: 1/4096, 1: 3/1024, 2: 33/2048, ..., 12: 1/4096}
# Pmf de W :  {0: 1/4096, 1: 3/1024, 2: 33/2048, ..., 2: 1/4096}
```

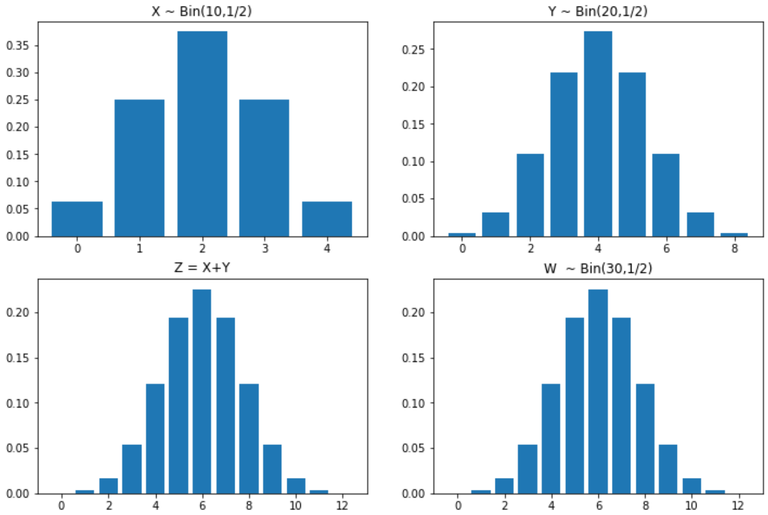
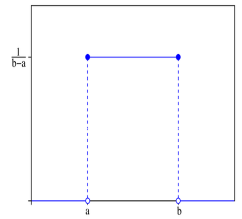
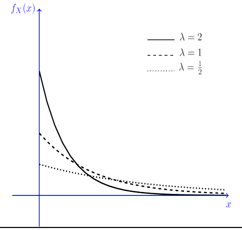
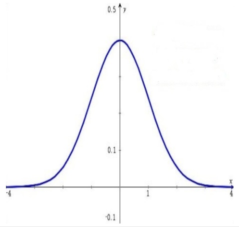
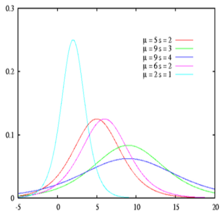
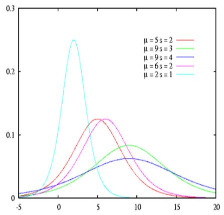
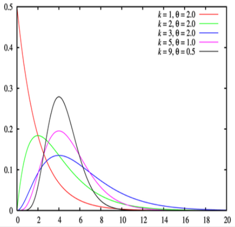


Figure 2.10: Curve of pmf (code208.py)

2.5.2 Continuous distributions

1- **Uniform** $\mathcal{U}(a,b)$: is characterized by a uniform representation of the values taken by the r.v over the interval$[a,b]$.
2- **Exponential** $\mathcal{E}xpo(\lambda)$): models the lifetime of a memoryless phenomenon with an average $1/\lambda$.
3- **Normal** $\mathcal{N}(\mu,\sigma^2)$: the density of a normal r.v is the Gauss function.
4- **Logistic** $\mathcal{L}ogistic$: its distribution function is a logistic function.
5- **Triangular** $\mathcal{T}ri(a,b,c)$: its density function is increasing affine for values between a and c (c is its mode) and decreasing affine between c and b.
6- **Gamma** $\mathcal{G}am(n,\lambda)$: represents the sum of n $expo(\lambda)$ rvs. It is based on the Gamma function.

Law X	Uniform	Exponential	Normal	Logistics	Triangular	Gamma
$X \rightsquigarrow$	$\mathcal{U}(a,b)$	$\mathcal{E}xpo(\lambda)$	$\mathcal{N}(\mu,\sigma^2)$	$\mathcal{L}ogistic$	$\mathcal{T}ri(a,b,c)$	$\mathcal{G}am(n,\lambda)$
Params	$a,b \in \mathbb{R}$	$\lambda \in \mathbb{R}$	μ,σ	-	a,b,c	n,λ
Rank	$[a,b]$	$\mathbb{R}$	$\mathbb{R}$	$\mathbb{R}$	$\mathbb{R}$	$[0,\infty[$
$f_X(x)$	$\frac{1}{b-a}\mathbb{1}_{[a,b]}(x)$	$\lambda e^{-\lambda x}\mathbb{1}_{[0,\infty[}(x)$	$\frac{1}{\sqrt{2\pi\sigma}}e^{\frac{(x-\mu)^2}{2\sigma^2}}$	$\frac{e^x}{(1+e^x)^2}$	$\begin{cases}\frac{2(x-a)}{(b-a)(b-c)} & \text{on }]a,c] \\ \frac{2(b-x)}{(b-a)(b-c)} & \text{on }]c,b]\end{cases}$	$\frac{\lambda^\alpha x^{\alpha-1}e^{-\lambda x}}{\Gamma(\alpha)}\mathbb{1}_{[0,\infty[}(x)$
$F_X(x)$	$\frac{x-a}{b-a}\mathbb{1}_{[a,b[}(x)+\mathbb{1}_{[b,\infty[}(x)$	$1-e^{-\lambda\, x}$	$\int_{-\infty}^{x}\frac{1}{\sqrt{2\pi}}e^{-t^2/2}\,dt$	$\frac{e^x}{1+e^x}$	$\begin{cases}\frac{(x-a)^2}{(b-a)(c-a)} & \text{on }]a,c] \\ -\frac{b-x^2}{(b-a)(b-c)} & \text{on }]c,b]\end{cases}$	$\int_0^t \frac{\lambda^\alpha t^{\alpha-1}e^{-\lambda t}}{\Gamma(\alpha)}\mathbb{1}_{[0,\infty[}(t)dt$
$\mathbb{E}(X)$	$\frac{a+b}{2}$	$\frac{1}{\lambda}$	μ	0	$\frac{a+b+c}{3}$	$\frac{\alpha}{\lambda}$
$\mathbb{V}(X)$	$\frac{(b-a+1)^2-1}{12}$	$\frac{1}{\lambda^2}$	σ	$\pi^2/3$	$\frac{a^2+b^2+c^2-ab-ac-bc}{18}$	$\frac{\alpha}{\lambda^2}$
Plot						

Memoryless property of the Exponential distribution
The most important property of the exponential law is the *memoryless* property.

> **Proposition 5.**
> *A c.r.v $X \rightsquigarrow \mathcal{E}xp(\lambda)$ if and only if X satisfies the following property:*
>
> $$P(X > x + a | X > a) = P(X > x) \quad \textit{for } x, a \geq 0$$
>
> *This property is called the memoryless property.*

Proof
1- Consider a c.r.v $X \rightsquigarrow \mathcal{E}xpo(\lambda)$ and x, a two positive real numbers,

$$\begin{aligned} P(X > x + a | X > a) &= \frac{P(X > x + a, X > a)}{P(X > a)} = \frac{P(X > x + a)}{P(X > a)} \\ &= \frac{1 - F_X(x + a)}{1 - F_X(a)} = \frac{e^{-\lambda(x+a)}}{e^{-\lambda a}} = e^{-\lambda x} = P(X > x) \end{aligned}$$

Then X satisfy the memoryless property.

2- Consider a c.r.v X that satisfies the memoryless property and x, a two positive real numbers,

$$\mathsf{P}(X > x + a) = \mathsf{P}(X > a)\mathsf{P}(X > x + a \mid X > a) = \mathsf{P}(X > x)\mathsf{P}(X > a)$$

$$1 - F_X(x + a) = (1 - F_X(x))(1 - F_X(a))$$

$$G_X(x + a) = G_X(x)G_X(a) \qquad \text{such that } G_X(z) := 1 - F_X(z)$$

$$G_X(n) = G_X(1)^n = e^{nlnG_X(1)} \qquad \forall n \in \mathbb{N}$$

$$G_X(1/m) = G_X(1)^{1/m} \qquad \forall m \in \mathbb{N}$$

$$G_X(r) = G_X(1)^r \qquad \forall r \in \mathbb{Q}$$

$$G_X(x) = G_X(1)^x = e^{xlnG_X(1)} \qquad \forall x \in \mathbb{R}$$

Then $X \rightsquigarrow \mathcal{E}xpo(\lambda)$ with $\lambda = -lnG(1)$

The last result, generalization for all reals, is based on the fact that all real is a limit of a rational sequence (here is the proof).
This means that if no event has occurred until time a $(X > a)$, then the waiting time from time a until the next event arrives $(Y = X - a)$ follows the same distribution as the waiting time from the initial moment (instant 0).

Example 12. *Suppose the duration of a phone booth call (in minutes) is $X \rightsquigarrow \mathcal{E}xp(1/10)$.*
1- If someone arrives just before you (in booth call), find the probability that you will wait: a) less than 5 minutes, b) more than 10 minutes, c) between 5 and 10 minutes.
2- Calculate the mean waiting time as well as its variance.
3- Write the Python code that models this situation.

Solution
Let X be a r.v that represents the duration of the call $X \rightsquigarrow \mathcal{E}xpo(\lambda)$.
Let Y be a r.v that represents your waiting time.
It is assumed that you arrived after A minutes of calling from whoever occupies the booth. You already have the information (and he too) that his call duration X is greater than A. With this condition, your waiting time will be the rest of his call i.e $Y = X - A$. "Y does not exceed 5 min" is equivalent to $X < 5 + A$ given that $X > A$ (figure 2.11 shows the relationship between

X and Y and the waiting bounds Max and Min). Using the *memoryless* property of $\mathcal{E}xpo$ r.v, we can calculate the probability of waiting no more than d minutes.

$$\begin{aligned} P(Y < d) &= P(X < A + d | X > A) \\ &= 1 - P(X > A + d | X > A) \\ &= 1 - P(X > d) = P(X \leq d) \end{aligned}$$

This expression shows that the waiting time is not function of the arrival time (it does not remember when we arrived - no memory-) and it follows the same law as the call duration $Y \rightsquigarrow \mathcal{E}xpo(\lambda)$.

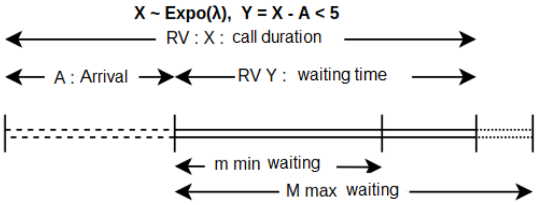


Figure 2.11: Relationship between the two r.v X and Y

$$\begin{aligned} P(Y < 5) &= P(X < 5) = 1 - e^{-\lambda 5} = 1 - e^{-\frac{1}{2}} = 0.3934 \\ P(Y > 10) &= P(X > 10) = 1 - P(X \leq 10) = 1 - 1 + e^{-1} = 0.3678 \\ P(5 \leq Y \leq 10) &= P(5 \leq X \leq 10) = 1 - [P(X \leq 5) + P(X > 10)] = 1 - (0.3934 + 0.3678) = 0.2388 \\ \mathbb{E}(Y) &= \mathbb{E}(X) = \frac{1}{\lambda} = 10 \\ \mathbb{V}(Y) &= \mathbb{V}(X) = \frac{1}{\lambda^2} = 100 \end{aligned}$$

Let’s code!

```
#Code209.py

# subs: substitutes parameters with values in the algebric expression
# simplify: simplifies an algebric expression.
from sympy.stats import Exponential, density, cdf, E, variance, given, P
from sympy import Symbol, simplify, And

lamda , z = Symbol("lambda", positive=True), Symbol("z")

X = Exponential("x", lamda)
pdf_X, cdf_X, E_X, V_X = density(X)(z),  cdf(X)(z), E(X), variance(X)
print(’RV X. \n pdf:’,pdf_X,’\n cdf:’,cdf_X,’\n E:’,E_X,’\n V:’,V_X)

Z = X.subs(lamda,1/10)
pdf_Z = density(Z)(z)  ;cdf_Z = cdf(Z)(z) ;E_Z = E(Z) ;V_Z = variance(Z)
print(’RV Z. \n pdf:’, pdf_Z,’\n cdf:’, cdf_Z,’\n E:’,E_Z,’\n V:’,V_Z)

Y = given(X - 3, X > 3)
pdf_Y = simplify(density(Y)(z))
P_Y5  = P(Y < 5).subs(lamda, 1/10)
P_Y10 = P(Y > 10).subs(lamda, 1/10)
P_Y510 = P(And(Y > 5,Y < 10)).subs(lamda, 1/10)
print(’RV Y. \n pdf:’, pdf_Y,’\n P(Y<5):’,P_Y5,’\n P(Y<10):’,P_Y10,’\n P(Y>5,Y<10):’,P_Y510)

#______________________________   Output   ______________________________________
# RV X.
#  pdf: lambda*exp(-lambda*z)
#  cdf: Piecewise((1 - exp(-lambda*z), z >= 0), (0, True))
#  E: 1/lambda
#  V: lambda**(-2)
# RV Z.
#  pdf: 0.1*exp(-0.1*z)
#  cdf: Piecewise((1 - exp(-0.1*z), z >= 0), (0, True))
#  E: 10.0000000000000
#  V: 100.000000000000
# RV Y.
#  pdf: -lambda*(Heaviside(-z) - 1)*exp(-lambda*z)
#  P(Y<5): 0.393469340287367
#  P(Y<10): 0.367879441171442
#  P(Y>5,Y<10): 0.238651218541191
```

Example 13. *A manufacturer wants to assure that his cars have a probability of less than* $p = 0.1$ *of breaking down over the first year. Assuming that the time until the first breakdown follows* $\mathcal{E}xpo$ *law:*
1- What is the minimum average of the first breakdown time of the car?
2- Write the Python code of this example.

Solution
Let T_p be the r.v giving the time of the first breakdown of the car which follows an exponential law of parameter λ_p ($T_p \rightsquigarrow \mathcal{E}xpo(\lambda_p)$, parameterized by p). The manufacturer wants to ensure that:

$$\begin{aligned} P(T_p \leq 1) = 1 - e^{-\lambda_p \times 1} \leq p &\implies \lambda_p \leq -ln(1-p) \\ &\implies \mathbb{E}(T_p) = \frac{1}{\lambda_p} \geq -\frac{1}{ln(1-p)} \end{aligned}$$

For $p = 0.1$, $\mathbb{E}(T) = \frac{1}{\lambda} \geq 9.5$.
So the minimum average is 9.5 years before getting the first breakdown (for a first breakdown probability less than 0.1).

The standard Normal law (normalized Normal law).
A r.v X that follows the standard Normal distribution is denoted by$X \rightsquigarrow \mathcal{N}(0,1)$, its expectation is equal to 0 and its variance is equal to 1. This law is generally defined by its distribution function $\phi : \mathbb{R} \to \mathbb{R}^+$:

$$\phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{-\frac{1}{2}t^2} dt \quad \forall x \in R$$

There is no analytical expression for this function. To get the value of $\phi(x) = P(X \leq x)$, we often use the distribution's tables.
The area under the bell curve of the standard normal distribution is equal to 1. It is symmetrical with respect to $z = 0$ (as it is an even function $\forall x \in \mathbb{R}, \phi(x) = \phi(-x)$, 50% is to the left of 0 and 50% to its right). The full plot of this distribution is shown in figure 2.12. To have the value of $\phi(z)$, we use the tables of the standard normal distribution left or right. The first one gives us the area to the right of the value z and the second one gives us the area to its left.

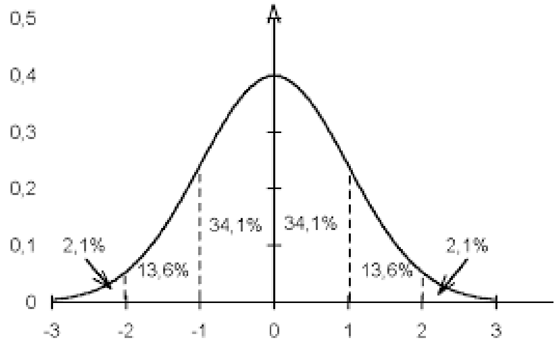


Figure 2.12: The distribution of the density of the reduced centered Normal law

Example 14. *The number of sunny days per year in a given city follows normal distribution with an average of* $\mu = 230$ *days and a standard deviation equal to* $\sigma = 52$ *days.*
1- What is the probability that the next year in this city there will be more than 300 sunny days?

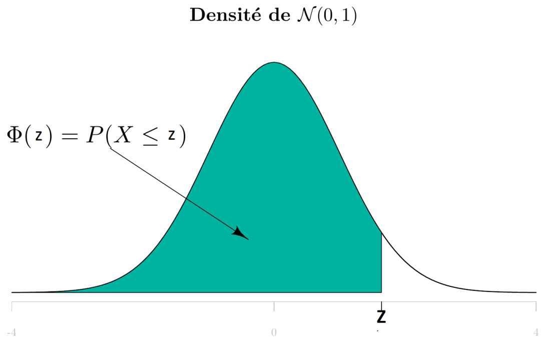


Figure 2.13: The value of the cdf at point z

Solution

If X is the r.v representing the number of sunny days per year in this city, $X \rightsquigarrow \mathcal{N}(\mu, \sigma)$ then:

$$P(X > 300) = P(\frac{X - 230}{52} > \frac{300 - 230}{52}) = 1 - \phi(1.346)$$

= 0.09012 (using the right Normal distribution table)
= 1-0.9099 = 0.0912 (using the left Normal distribution table)
= 0.5-0.4099 = 0.09012 (using the 0 to z table)
(*See standard normal distribution table (Z-table) in appendix C for the value of* $\phi(1.346)$)

Let’s code!

The following code implements a reduced centered r.v X and $Y = 3 * X + 5$.

```
#Code210,py

from sympy.stats import E,variance, Normal,density
from sympy import Symbol
from sympy.plotting import plot

def processNormalRV(rv_Name, rv_X):
    print(’E(’,rv_Name,’)= ’,E(rv_X),’   V(’,rv_Name,’)= ’,variance(rv_X))
    z = Symbol(’z’)
    plot(density(rv_X)(z), (z, -10, 20))

processNormalRV(’X’, Normal(’X’,0,1))
processNormalRV(’Y’, 3*Normal(’X’,0,1)+5)

#______________________________   Output   ______________________________________
# E( X )=  0    V( X )=  1
# E( Y )=  5    V( Y )=  9
```

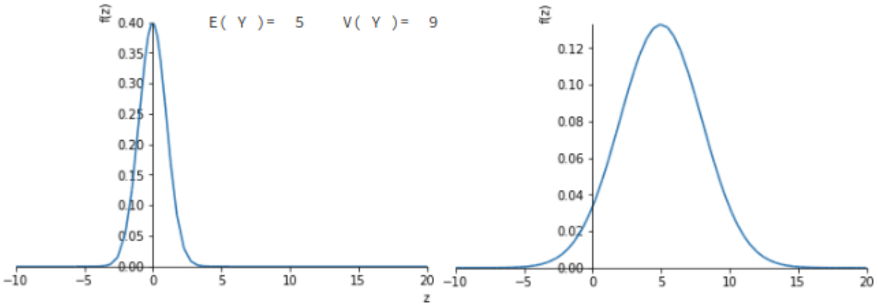

E(Y)= 5   V(Y)= 9

Figure 2.14: Standard vs non-standard normal distribution (code210.py)

In this code, a standard normal X r.v is created and used to create another one equal to $3 * X + 5$ (scaled with 3 and shifted with 5), the expectation and variance of the two variables are calculated and the two r.v are plotted using the *plotting package* of *sympy*. On the two graphs, we can see that X is centered on the axis $x = 0$ with $\mathbb{E}(X) = 0$ while Y is centered on the axis $x = 5$ with $\mathbb{E}(Y) = 5$; the variance of Y ($\mathbb{V}(Y) = 9$) is also much larger compared to that of X ($\mathbb{V}(X) = 1$) which has given a larger graph than that of X.

2.6 Multiple random variables

In many experiments, one may be interested in more than one random variable in relation to each other. For example, in the study of the effectiveness of a given diet, we can look at the age of the participants, their gender as well as the nature of their work (physical, intellectual,...). Here, we present the case of two r.vs (*bivariate* r.v), which can easily be extended to several r.vs (*multivariate*).

2.6.1 Joint, marginal and conditional distribution

Definition 9.
Let X and Y be two r.vs. The joint probability law of X and Y is defined by:

$$P_{XY}(B_1 \times B_2) = P((X,Y) \in B_1 \times B_2) = P(X \in B_1, Y \in B_2), \quad \forall (B_1, B_2) \in \mathcal{B}^2$$

The rank of this joined law is: $R_{XY} = R_X \times R_Y = \{(x,y) | x \in R_X \text{ and } y \in R_Y\}$

The joint distribution function of X and Y is defined by:

$$F_{XY}(x,y) = P_{XY}(]-\infty, x] \times]-\infty, y]) = P((X,Y) \in]-\infty, x] \times]-\infty, y]) = P(X \leq x, Y \leq y)$$

The marginal distribution function with respect to X (respec Y) is defined by:

$$F_X(x) = F_{XY}(x, \infty) = P_{XY}(] - \infty, x] \times \mathbb{R}) = P(X \leq x)$$

$$F_Y(y) = F_{XY}(\infty, y) = P_{XY}(\mathbb{R} \times] - \infty, y]) = P(Y \leq y)$$

The conditional distribution of X given Y is defined by:

$$P_{X|Y}(B_1, B_2) = P(X \in B_1 | Y \in B_2) = \frac{P_{XY}(B_1, B_2)}{P_Y(B_2)}, \quad \forall (B_1, B_2) \in \mathcal{B}^2$$

$$F_{X|Y}(x,y) = P_{X|Y}(] - \infty, x],] - \infty, y]) = \frac{F_{XY}(x,y)}{F_Y(y)}$$

Definition 10. ***discrete case***
Let X and Y be two d.r.v:

1- The function of joined mass of X and Y is defined by:

$$p_{XY}(x,y) = P_{XY}(\{(x,y)\}) = P((X,Y) = (x,y)) = P(X = x, Y = y)$$

2- The function of marginal mass with respect to X (respectively Y) is:

$$p_X(x) = \sum_{y \in R_Y} p_{XY}(x,y) \quad (P_Y(y) = \sum_{x \in R_X} p_{XY}(x,y))$$

3- The conditional mass function of X and Y is defined by:

$$p_{X|Y}(x,y) = P(X = x | Y = y) = \frac{p_{XY}(x,y)}{p_Y(y)}$$

Example 15. *Let X and Y be two d.r.v with the following joint mass function:*
$R_{XY} = R_X \times R_Y = \{0,1\} \times \{0,1,2\}$ *Find:*

1. $P(X = 0, Y \leq 1)$

(X,Y)	$(0,0)$	$(0,1)$	$(0,2)$	$(1,0)$	$(1,1)$	$(1,2)$
$p_{XY}(x,y)$	$\frac{1}{6}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{6}$	$\frac{1}{6}$

2. The marginals of X and Y

3. $P(Y=1|X=0)$

Solution

1-$P(X=0, Y\leq 1) = P_{XY}(\{0\}\times\{0,1\}) = p_{XY}(0,0) + p_{XY}(0.1) = 1/6 + 1/4 = 5/12$

2- The marginals $\forall x \in R_X$:

$$p_X(x) = \sum_{y\in\{0,1,2\}} p_{XY}(x,y) = p_{XY}(x,0) + p_{XY}(x,1) + p_{XY}(x,2) = \frac{13}{24}\mathbb{1}_{\{0\}}(x) + \frac{11}{24}\mathbb{1}_{\{1\}}(x)$$

Similarly, for Y,

3- $P(Y=1|X=0) = \dfrac{P(X=0,Y=1)}{P(X=0)} = 1/4 \times 24/13 = 8/13$

(X, Y)	0	1	2	P_X
0	$\frac{1}{6}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{13}{24}$
1	$\frac{1}{8}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{11}{24}$
P_Y	$\frac{7}{24}$	$\frac{5}{12}$	$\frac{7}{24}$	1

Let’s code!

```python
#Code211.py

import numpy as np
from functools import reduce
from scipy.stats import  rv_discrete
import matplotlib.pyplot as plt
import seaborn as sns
from random import random

generateDRV=lambda rv: [rv.ppf(random()) for _ in range(1000)]

def getbar3Data(Rg_x, Rg_y, p_xy):
    xpos, ypos = reduce(lambda ls, e: ls+[e]*len(Rg_y), Rg_x, []) , [0,1]*len(Rg_x)
    zpos = np.zeros(len(xpos))
    dx, dy = np.ones(len(xpos))*0.02, np.ones(len(xpos))*0.02
    dz = list(p_xy.flatten())
    return xpos, ypos, zpos, dx, dy, dz

def plotjointDistribution(xpos, ypos, zpos, dx, dy, dz):
    fig = plt.figure()
    ax1 = fig.add_subplot(111, projection=’3d’)
    ax1.bar3d(xpos, ypos, zpos, dx, dy, dz, color=’#00ceaa’)
    plt.show()

# random variables
Rg_x, Rg_y = np.array([0,1,2]), np.array([0,1])
p_xy = np.array([[1/6,1/4,1/8],[1/8,1/6,1/6]])

# data for joint distribution plotting
xpos, ypos, zpos, dx, dy, dz = getbar3Data(Rg_x, Rg_y, p_xy)
plotjointDistribution(xpos, ypos, zpos, dx, dy, dz)

# data for marginal plotting
p_x, p_y = np.sum(p_xy, axis=0), np.sum(p_xy, axis=1)
rv_X = rv_discrete(name=’X’, values=(Rg_x, p_x))
rv_Y = rv_discrete(name=’Y’, values=(Rg_y, p_y))

sns.jointplot(generateDRV(rv_X),generateDRV(rv_Y)).set_axis_labels("X", "Y")
```

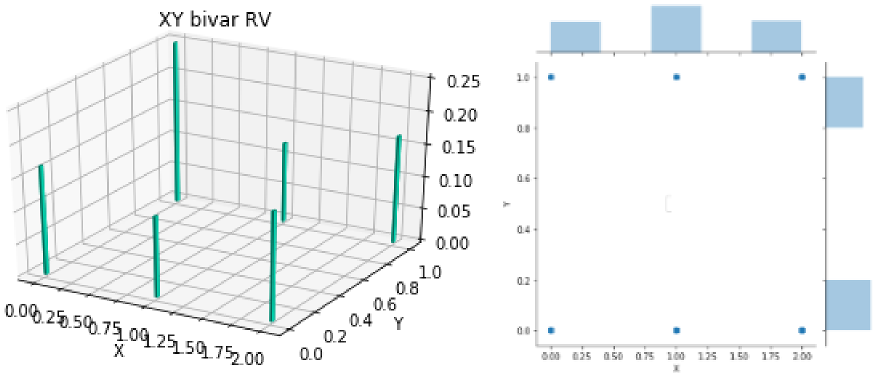


Figure 2.15: Joint and marginal pmf (code211.py)

Example 16. *Let X and Y be two d.r.v representing the lifetime of two connected components of a given machine. The joint mass function is given by: $p_{XY}(x,y) = \frac{e^{-2}}{x!(yx)!}$ for $x \in \mathbb{N} \wedge y \geq x$. Calculate:*

1- The joint mass function of X and $Y - X$

2- The marginal of X ,$Y - X$ and Y

***Note:** The sum of two Poisson r.v X and Y of respective rates λ_1 and λ_2 is a Poisson r.v of rate $\lambda_1 + \lambda_2$ (this result will be demonstrated in the next chapter).*

Solution

Let $Z = Y - X$

$$p_{XZ}(x,z) = P(X = x, Z = z) = P(X = x, Y - X = z) = P(X = x, Y = z + x)$$

$$= p_{XY}(x, z + x) = \frac{e^{-2}}{x!z!}, \quad \forall x, z \in \mathbb{N}$$

$$\text{So } p_X(x) = \sum_{z=0}^{\infty} = p_{XZ}(x,z) = \sum_{z=0}^{\infty} \frac{e^{-2}}{x!z!} = \frac{e^{-1}}{x!}, \qquad \left(\text{the same for } p_Z(z) = \frac{e^{-1}}{z!}\right)$$

So, we have $X, Z \rightsquigarrow \mathcal{P}ois(1)$ (they follow the same law). We have,

$$P(X = x, Y - X = z) = \frac{e^{-2}}{x!z!} = e^{-1}x!\frac{e^{-1}}{z!} = P(X = x) \times P(Y - X = z)$$

We deduce that X and $Y - X$ are independent, and $Y \rightsquigarrow \mathcal{P}ois(2)$.

Definition 11. ***continuous case***

Let X and Y be c.r.v:

1- The joint density function of X and Y is the function verifying:

$$P_{XY}(B_1 \times B_2) = \int_{B_1}\int_{B_2} f_{XY}(x,y)dxdy, \quad \forall (B_1, B_2) \in \mathcal{B}^2$$

2- The relation between this function and the joint cdf:

$$f_{XY}(x,y) = \frac{\partial^2}{\partial x \partial y}F_{XY}(x,y) \quad F_{XY}(x,y) = \int_{-\infty}^{x}\int_{\infty}^{y} f_{XY}(x,y)dxdy$$

3- The marginal density function with respect to X is:

$$f_X(x) = f_{XY}(x, \infty) = \int_{-\infty}^{\infty} f_{XY}(x,y)dy = \frac{\partial}{\partial x}F_X(x)dx$$

4- The conditional density function (c-pdf) of X and Y is the function satisfying:

$$f_{X|Y}(x,y) = \frac{f_{XY}(x,y)}{f_Y(y)}$$

Example 17. *A point is chosen randomly inside a circular disk D of radius r. Let X be the r.v that represents the distance of the segment going from the center of this disc to the chosen point and Y the r.v measuring the angle (in radians) between this segment and the horizontal axis. What is the joint distribution of X and Y?*

Solution
Both r.vs X, Y are defined on the same probability space $(\Omega, \mathcal{F}, P)$, Ω is the set of points on the disk, $\mathcal{F}$ a $\sigma-algebradefinedonthissetand$P$theuniformprobabilityofchoosinganareaonthisdisk$(P(B) = |B|/|D|, $where$|Z|$istheareaofthezone$Z).F$_{XY}(x,y) = P(X \leq x, Y \leq y)$ corresponds to the probability that the chosen point is in the segment with radius x and angle y whose area is $\frac{yx^2}{2}$. So the cdf and its corresponding pdf:

$$F_{XY}(x,y) = \frac{yx^2}{2\pi r^2}\mathbb{1}_{[0,r]\times[0,2\pi]}(x,y)$$
$$f_{XY}(x,y) = \frac{\partial^2}{\partial x \partial y}F_{XY}(x,y) = \frac{x}{\pi r^2}\mathbb{1}_{[0,r]\times[0,2\pi]}(x,y)$$

2.6.2 Covariance and correlation coefficient

The covariance of two r.v X and Y is a measure that gives information about the linear dependence of X and Y. That is, how the values of X change according to those of Y.

Definition 12.
The covariance between X and Y is defined as:

$$Cov(X,Y) = \mathbb{E}[(X - \mathbb{E}(X))(Y - \mathbb{E}(Y))] = \mathbb{E}(XY) - \mathbb{E}(X)\mathbb{E}(Y)$$

When $Cov(X,Y) = 0$, X and Y are said to be *uncorrelated*.
If $Cov(X,Y)$ is positive, they are said to be *positively correlated* i.e. either $X - \mathbb{E}(X) > 0$ and $Y - \mathbb{E}(Y) > 0$ or $X - \mathbb{E}(X) < 0$ and $Y - \mathbb{E}(Y) < 0$; otherwise X and Y are said to be *negatively correlated*.
The covariance can also be used in variance calculation:

$$\mathbb{V}(X+Y) = \mathbb{V}(X) + \mathbb{V}(Y) + 2Cov(X,Y)$$

So if X and Y are *uncorrelated* ($Cov(X,Y) = 0$) then: $\mathbb{V}(X+Y) = \mathbb{V}(X) + \mathbb{V}(Y)$
This result can be generalized to several uncorrelated pairs of r.vs. Independent r.vs are always uncorrelated, but the opposite is not necessarily true.
Correlation is often measured by the *correlation coefficient*.

Definition 13.
The correlation coefficient is the ratio defined by:

$$\rho_{XY} = \frac{Cov(X,Y)}{\sqrt{\mathbb{V}(X)}\sqrt{\mathbb{V}(Y)}}, \quad \textit{satisfaing } -1 \leq \rho_{XY} \leq 1$$

It is zero when the two r.vs are uncorrelated, positive if they are positively correlated and negative otherwise.
In fact, ρ is the normalized version of the covariance. It is obtained by calculating the covariance of the two r.v $X' = \frac{X - \mathbb{E}(X)}{\sigma_X}$ and $Y' = \frac{Y - \mathbb{E}(Y)}{\sigma_Y}$

Proposition 6. *Covariance properties:*
1- $Cov(X, X) = \mathbb{V}(X)$
2- $Cov(X, Y) = Cov(Y, X)$
3- $Cov(aX, bY) = abCov(X, Y)$
4- $Cov(X + Y, Z) = Cov(X, Z) + Cov(Y, Z)$
5-

$$Cov(\sum_{i=1}^{n} a_i X_i, \sum_{j=1}^{m} b_j Y_j) = \sum_{i=1}^{n}\sum_{j=1}^{m} a_i b_j Cov(X_i, Y_j)$$

Example 18.
1. Calculate the covariance and the correlation coefficient of X *and* Y *from Example 15.*
2. Write the corresponding code

Solution

1.

$$Cov(X, Y) = \mathbb{E}(XY) - \mathbb{E}(X)\mathbb{E}(Y) = 1/2 - 11/24 \times 1 = 1/24$$

$$\rho_{XY} = \frac{Cov(X, Y)}{\sqrt{\mathbb{V}(X)}\sqrt{\mathbb{V}(Y)}} = 0.109$$

2. **Let's code!**

```
#Code212,py

import numpy as np
from sympy.stats import FiniteRV
from sympy.stats import E,variance
import math

XDensity,YDensity = {0:13/24,1:11/24},{0:7/24,1:5/12,2:7/24}
X, Y = FiniteRV(’X’,XDensity ), FiniteRV(’Y’,YDensity )

JDensity = {(0,0):1/6,(0,1):1/4,(0,2):1/8,(1,0):1/8,(1,1):1/6,(1,2):1/6}

ZDensity = {(k[0]-E(X))*(k[1]-E(Y)):v for k,v in JDensity.items()}
cov = np.dot(list(ZDensity.keys()),list(ZDensity.values()))

print("E(X)=",E(X),"   E(Y)=",E(Y))
print("V(X)=",variance(X),"   V(Y)=",variance(Y))
print("Cov(X,Y)=",cov)
print("Correlation Coefficient=",cov/(math.sqrt(variance(X))*math.sqrt(variance(Y))))

#______________________________    Output  ______________________________________
# E(X)= 0.458333333333333    E(Y)= 1.00000000000000
# V(X)= 0.248263888888889    V(Y)= 0.583333333333333
# Cov(X,Y)= 0.0416666666666667
# Correlation Coefficient= 0.109489780290272
```

Example 19.
1. Calculate the covariance and the correlation coefficient of the two r.vs X *and* Y *having the joint function:*

$$f_{XY}(xy) = 3x \qquad 0 \leq y \leq x \leq 1$$

Solution

$$f_X(x) = \int_{-\infty}^{+\infty} f_{XY}(x,y)dy = \int_0^x 3xdy = 3x^2$$

$$f_y(y) = \int_{-\infty}^{+\infty} f_{XY}(x,y)dx = \int_y^1 3xdx = \frac{3(1-y^2)}{2}$$

$$\mathbb{E}(XY) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} xyf_{xy}(x,y)dxdy = \frac{3}{10}[x^5]_0^1 = \frac{3}{10}$$

$$\mathbb{E}(X) = \int_0^1 xf_X(x)dx = \frac{3}{4}$$

$$\mathbb{E}(Y) = \int_0^1 yf_Y(y)dy = \frac{3}{8}$$

$$Cov(X,Y) = \mathbb{E}(XY) - \mathbb{E}(X)\mathbb{E}(Y) = \frac{3}{160}$$

$$\mathbb{E}(X^2) = \int_0^1 x^2 f_X(x)dx = \frac{3}{5}$$

$$\mathbb{E}(Y^2) = \int_0^1 y^2 f_X(y)dy = \frac{1}{5}$$

$$\mathbb{V}(X) = \mathbb{E}(X^2) - \mathbb{E}^2(X) = \frac{3}{80}$$

$$\mathbb{V}(Y) = \mathbb{E}(Y^2) - \mathbb{E}^2(Y) = \frac{19}{320}$$

$$\rho_{XY} = \frac{Cov(X,Y)}{\sqrt{\mathbb{V}(X)}\sqrt{\mathbb{V}(Y)}} = 0.397$$

Let’s code!

```
#Code213.py

from sympy import Symbol, integrate, Interval
from sympy.stats import density, E, variance as V
from math import sqrt

x = Symbol(’x’); y = Symbol(’y’)

fxy = 3*x
fx = integrate(fxy,(y,0,x)) ; print("marginal of X:",fx)
fy = integrate(fxy,(x,y,1)) ; print("marginal of Y:",fy)

# Direct method
Ex , Ey = integrate(x*fx,(x,0,1)) , integrate(y*fy,(y,0,1))
print("E(X)=",Ex,"   E(Y)=",Ey)

Exy = integrate(integrate(x*y*fxy,(y,0,x)),(x,0,1)) ; print("E(X*Y)=",Exy)
cov = Exy-Ex*Ey ; print("Cov(X,y)=",cov)

Ex2 , Ey2 = integrate((x**2)*fx,(x,0,1)) , integrate((y**2)*fy,(y,0,1))
vx , vy = Ex2-Ex**2 , Ey2-Ey**2
print("V(X)=",vx) ; print("V(Y)=",vy)

cor = cov/(sqrt(vx)*sqrt(vy)) ; print("Correlation coefficient:",cor)

# ContinuousRV
from sympy.stats import ContinuousRV

X     = ContinuousRV(symbol=x, density= fx, set=Interval(0, 1))
Y     = ContinuousRV(symbol=y, density= fy, set=Interval(0, 1))
print("E(X)=",E(X), ",E(X^2)=", E(X**2),",Var(X)=", V(X))
print("E(Y)=",E(Y), ",E(Y^2)=", E(Y**2),",Var(Y)=", V(Y))

cov = Exy-E(X)*E(Y) ; print("Cov(X,y)=",cov)

cor = cov/(sqrt(V(X))*sqrt(V(Y))) ; print("Correlation coefficient:",cor)

#______________________________   Output  ______________________________________
# marginal of X: 3*x**2
# marginal of Y: 3/2 - 3*y**2/2
# E(X)= 3/4    E(Y)= 3/8
# E(X*Y)= 3/10
# Cov(X,y)= 3/160
# V(X)= 3/80
# V(Y)= 19/320
# Correlation coefficient: 0.397359707119513
```

2.7 Moment Generating Functions

Definition 14.
The n^{th} moment of a r.v X is defined to be $m_X^{(n)} = \mathbb{E}(X^n)$.
The n^{th} central moment of X is defined to be $\bar{m}_X^{(n)} = \mathbb{E}((X - \mathbb{E}(X))^n)$.
The moment generating function (MGF) of a r.v X is a function $M_X(t)$ that associates to t the expectation of the r.v e^{tX}.

$$M_X(t) = \mathbb{E}(e^{tX})$$

M_X exists, if there exists a positive constant c such that $\mathbb{E}(e^{tX})$ is finite for all $|t| \leq c$ (neighborhood of 0, $t \in \delta(0, c)$).

$$\exists c > 0, \forall |t| < c, \mathbb{E}(e^{tX}) < \infty \implies \exists M_X$$

- The first moment is the expectation of X $m_X^{(1)} = m_X = \mathbb{E}(X)$.
- The second central moment is the variance of X, $\bar{m}_X^{(1)} = \bar{m}_X = \mathbb{V}(X)$.
- The MGF of X gives us all moments of X.

Example 20. *For each of the following r.vs, find the MGF.*
a- $X \sim \mathcal{B}er(p)$
b- $X \sim \mathcal{U}ni(0, 1)$
c- $X \sim \mathcal{E}xpo(\lambda)$
d- $X \sim \mathcal{P}ois(\lambda)$

Solution
we have $M_X(t) = \mathbb{E}[e^{tX}]$

$$a{-}X \sim \mathcal{B}er(p) \qquad : M_X(t) = e^t p + e^{0t}(1-p) = pe^t - p + 1$$

$$b{-}X \sim \mathcal{U}ni(0,1) \qquad : M_X(t) = \int_0^1 e^{tx} dx = \frac{e^{tx}}{s}\bigg|_0^1 = \frac{e^t - 1}{t}$$

$$c{-}X \sim \mathcal{E}xpo(\lambda) \qquad : M_X(t) = \int_0^\infty e^{tx} \lambda e^{-\lambda x} dx = \lambda \int_0^\infty e^{-(\lambda - t)x} dx = \frac{\lambda}{\lambda - t}$$

$$d{-}X \sim \mathcal{P}ois(\lambda) \qquad : M_X(t) = \sum_{k=0}^\infty e^{tk} e^{-\lambda} \frac{\lambda^k}{k!} = e^{-\lambda} \sum_{k=0}^\infty \frac{(e^t \lambda)^k}{k!} = e^{\lambda(e^t - 1)}$$

2.7.1 MGF as a tool to find the moments

Substituting e^{tX} by its Taylor expansion in M_X shows that $m_X^{(k)}$ represents the k^{th} coefficient of this serie which can be determined by taking the k^{th} derivative of $M_X(t)$ and evaluating it at $t = 0$.

$$M_X(t) = \mathbb{E}[e^{tX}] = \mathbb{E}[\sum_{k=0}^\infty \frac{X^k t^k}{k!}] = \sum_{k=0}^\infty \mathbb{E}[X^k] \frac{t^k}{k!} = \sum_{k=0}^\infty \frac{\mathbb{E}[X^k]}{k!} t^k$$

$$\frac{d^k}{ds^k} M_X(t) = M_X^{(k)}(t) = \sum_{n=k}^\infty \frac{\mathbb{E}[X^n]}{(n-k)!} t^{n-k} = \mathbb{E}[X^k] + t(\sum_{n=0}^\infty \frac{\mathbb{E}[X^{n+k+1}]}{(n+1)!} t^n)$$

Then, we have :

$$\frac{d^k}{dt^k} M_X(t)\bigg|_{t=0} = M_X^{(k)}(t)\bigg|_{t=0} = \mathbb{E}[X^k] = m_X^{(k)}$$

Example 21. *For the questions of example 20, find $\mathbb{E}[X^k]$ using $M_X(t)$.*

Solution

we have $\mathbb{E}(X) = \frac{d}{dt}M_X(t)\Big|_{t=0}$

$$a{-}X \sim \mathcal{B}er(p) : \mathbb{E}(X) = \frac{d}{dt}(pe^t - p + 1)\Big|_{t=0} = pe^s\Big|_{t=0} = p$$

$$b{-}X \sim \mathcal{U}ni(0,1) : \mathbb{E}(X) = \frac{d}{dt}\frac{e^t - 1}{t}\Big|_{t=0} = \frac{(t-1)e^t + 1}{t^2}\Big|_{t=0} = \overbrace{\lim_{x \to 0}\left(\frac{e^t t - e^t + 1}{t^2}\right)}^{\text{Hopital rule}} = \frac{1}{2}$$

$$c{-}X \sim \mathcal{E}xpo(\lambda) : \mathbb{E}(X) = \frac{d}{dt}\frac{\lambda}{\lambda - t}\Big|_{t=0} = \frac{\lambda}{(\lambda - t)^2}\Big|_{t=0} = \frac{1}{\lambda}$$

$$d{-}X \sim \mathcal{P}ois(\lambda) : \mathbb{E}(X) = \frac{d}{dt}e^{\lambda(e^t - 1)}\Big|_{t=0} = \lambda e^t e^{\lambda(e^t - 1)}\Big|_{t=0} = \lambda$$

if M_X exists, it uniquely determines the distribution of X (F_X).

> **Definition 15.** *Let X and Y be two r.vs*
> *Suppose that there exists a positive constant c such that the MGFs of X and Y are finite and identical $\forall |t| \leq c$. Then, $F_X(x) = F_Y(x), \forall x \in \mathbb{R}$*
>
> $$\exists c > 0, M_X, M_Y \ \& \ \forall |t| \leq c, M_X(t) = M_Y(t) \implies F_X(x) = F_Y(x), \forall x \in \mathbb{R}$$

> **Definition 16.** *Sum of Independent Random Variables with MGF:*
> *Suppose $X_1, X_2, ..., X_n$ are n independent r.vs*
>
> $$\forall 1 \leq i,j \leq n, i \neq j, \quad X_i \perp X_j \implies M_{\sum_{i=1}^n X_i} = \prod_{i=1}^n M_{X_i}$$

Proof

$$M_{\sum_{i=1}^n X_i}(t) = \mathbb{E}[e^{t(\sum_{i=1}^n X_i)}] = \mathbb{E}[\prod_{i=1}^n e^{tX_i}] = \prod_{i=1}^n M_{X_i}(t)$$

Example 22.
a- For $X \sim \mathcal{B}in(n,p)$, find its MGF.
b- Prove that :

$$X \sim \mathcal{B}in(m,p) \ \& \ Y \sim \mathcal{B}in(n,p) \ \& \ X \perp Y \implies X + Y \sim \mathcal{B}in(m+n,p)$$

Solution
Let $X \sim \mathcal{B}in(n,p)$
Then $X = \sum_{i=1}^n X_i$ such that $X_i \sim \mathcal{B}er(p)$ & $\forall 1 \leq i,j \leq n, i \neq j$, we have $X_i \perp X_j$

$$M_X(t) = \mathbb{E}[e^{t(\sum_{i=1}^n X_i)}] = \prod_{i=1}^n M_{X_i}(t) = (M_{X_0}(t))^n$$

$$M_{X+Y}(t) = M_X(t)M_Y(t) = (M_{X_0}(t))^n(M_{X_0}(t))^m = (M_{X_0}(t))^{n+m}$$

X+Y $\sim \mathcal{B}in(n+m,p)$

2.8 Transformation of random variables

Definition 17. ***Variable change***
$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \in \mathbb{R}^2$
Changing variable $\mathbf{x}$ *and* $\mathbf{y}$ *is a bijective map (transformation)* $\phi : \mathbb{R}^2 \to \mathbb{R}^2$ *such that:*

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \phi(\mathbf{x}) = \begin{pmatrix} \phi_1(x_1, x_2) \\ \phi_2(x_1, x_2) \end{pmatrix}$$

The inverse transformation of ϕ *is* $\phi^{-1} : \mathbb{R}^2 \to \mathbb{R}^2$ *defined by:*

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \phi^{-1}(\mathbf{y}) = \begin{pmatrix} \phi_1^{-1}(y_1, y_2) \\ \phi_2^{-1}(y_1, y_2) \end{pmatrix}$$

Definition 18. ***The law of a transform***
Consider the vector of r.v in $\mathbb{R}^2$, $X = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}$, *the vector* $Y = \begin{pmatrix} Y_1 \\ Y_2 \end{pmatrix}$ *is the transform of* X *with a transformation* ϕ: $Y = \phi(X)$ *if the law of* Y *is given by*

$$\forall A \in \mathcal{B}, \quad P_Y(A) = P(Y \in A) = P(X \in \phi^{-1}(A)) = P_X(\phi^{-1}(A))$$

Example 23. *Given the following transformation:*

$$\mathbf{y} = \phi(\mathbf{x}) = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} \phi_1(\mathbf{x}) \\ \phi_2(\mathbf{x}) \end{pmatrix} = \begin{pmatrix} x_1 - x_2 \\ x_1 + x_2 \end{pmatrix}$$

Find its inverse transformation and find $P(Y_2 < 1)$.
Solution
The inverse transformation :

$$\mathbf{x} = \phi^{-1}(\mathbf{y}) = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \phi_1^{-1}(\mathbf{y}) \\ \phi_2^{-1}(\mathbf{y}) \end{pmatrix} = \begin{pmatrix} \frac{y_1+y_2}{2} \\ \frac{y_2-y_1}{2} \end{pmatrix}$$

$$Y = \phi(X)$$

$$P(Y_2 < 1) = P(Y \in \mathbb{R}\times] - \infty, 1[) = P(\phi_1^{-1}(\mathbb{R}\times] - \infty, 1[)) = P(X_1 + X_2 < 1)$$

Definition 19. ***Jacobian of a transformation***
Consider the transformation ϕ *over* $\mathbb{R}^m \times \mathbb{R}^n$, *the Jacobian of* $\phi = (\phi_1, \cdots, \phi_m)^T$ *is given by:*

$$\mathbb{J}_\phi(\mathbf{x}) = \frac{\partial \phi(\mathbf{x})}{\partial \mathbf{x}} = \begin{vmatrix} \frac{\partial}{\partial x_1}\phi(\mathbf{x}) & \cdots & \frac{\partial}{\partial x_n}\phi(\mathbf{x}) \end{vmatrix} = \begin{vmatrix} \frac{\partial}{\partial x_1}\phi_1(\mathbf{x}) & \cdots & \frac{\partial}{\partial x_n}\phi_1(\mathbf{x}) \\ \vdots & \ddots & \vdots \\ \frac{\partial}{\partial x_1}\phi_m(\mathbf{x}) & \cdots & \frac{\partial}{\partial x_n}\phi_m(\mathbf{x}) \end{vmatrix}$$

For the case of ϕ in the first example :

$$\mathbb{J}_\phi(\mathbf{x}) = \begin{vmatrix} \frac{\partial}{\partial x_1}\phi_1(\mathbf{x}) & \frac{\partial}{\partial x_2}\phi_1(\mathbf{x}) \\ \frac{\partial}{\partial x_1}\phi_2(\mathbf{x}) & \frac{\partial}{\partial x_2}\phi_2(\mathbf{x}) \end{vmatrix} = \frac{\partial y_1}{\partial x_1}\frac{\partial y_2}{\partial x_2} - \frac{\partial y_1}{\partial x_2}\frac{\partial y_2}{\partial x_1} = 2$$

For the case of ϕ^{-1} in the first example :

$$\mathbb{J}_{\phi^{-1}}(\mathbf{y}) = \begin{vmatrix} \frac{\partial}{\partial y_1}\phi_1^{-1}(\mathbf{y}) & \frac{\partial}{\partial y_2}\phi_1^{-1}(\mathbf{y}) \\ \frac{\partial}{\partial y_1}\phi_2^{-1}(\mathbf{y}) & \frac{\partial}{\partial y_2}\phi_2^{-1}(\mathbf{y}) \end{vmatrix} = \frac{\partial x_1}{\partial y_1}\frac{\partial x_2}{\partial y_2} - \frac{\partial x_1}{\partial y_2}\frac{\partial x_2}{\partial y_1} = 1/2$$

Theorem 1. *Density of a transform*
The density of a random vector Y transformed from X by ϕ is:

$$f_Y(\mathbf{y}) = f_X(\phi^{-1}(\mathbf{y}))|\mathbf{J}_{\phi^{-1}}(\mathbf{y})|$$

Example 24. *Consider X and its transform Y from the previous example. The density of X over the domain $A = \{(x_1, x_2) \in \mathbf{R}^2 | 0 < x_1 < x_2 < \infty\}$ is given by:*

$$f_X(\mathbf{x}) = e^{-x_2}\mathbb{1}_A$$

1- Find the density of Y
2- Calculate $P(Y_2 < 1)$
Solution
The domain of Y is the transform of A by ϕ :

$$\begin{aligned}\phi^{-1}(A) &= \{(y_1, y_2) \in \mathbf{R}^2 | 0 < y_2 - y_1 < y_1 + y_2 < \infty\} \\ &= \{(y_1, y_2) \in \mathbf{R}^2 | 0 < y_1 < y_2 < \infty\}\end{aligned}$$

$$f_Y(\mathbf{y}) = f_X(\phi^{-1}(\mathbf{y}))|\mathbf{J}_{\phi^{-1}}(\mathbf{y})| = e^{-(y_1+y_2)/2}\mathbb{1}_{\phi^{-1}(A)}\frac{1}{2} = \frac{1}{2}e^{-(y_1+y_2)/2}\mathbb{1}_{\phi^{-1}(A)}$$

$$\begin{aligned}P(Y_2 < 1) = P(X_1 + X_2 < 1) &= \int_0^{1/2}\int_0^{x_2} f_X(x_1, x_2)dx_1dx_2 \\ &= \int_0^{1/2}\int_0^{x_2} e^{-x_2}\mathbb{1}_A dx_1dx_2 = \underbrace{\int_0^{1/2} x_2e^{-x_2}dx_2 = 0.09020}_{\text{Integration by part}}\end{aligned}$$

Example 25. *Let ϕ be an invertible transformation defined over $\mathbb{R}^2$*

$$\mathbf{y} = \phi(\mathbf{x}) = \begin{pmatrix} x_1^2 + x_2^2 \\ x_1/x_2 \end{pmatrix}$$

Let X be a random vector with the following density function:

$$f_X(\mathbf{x}) = f_X(x_1, x_12) = \frac{e^{-(x_1^2+x_2^2)/2}}{2\pi}$$

and its transform Y by ϕ

$$Y = \phi(X)$$

Determine the density of Y.
Solution
The inverse transform and its jacobian :

$$\mathbf{x} = \phi^{-1}(\mathbf{y}) = \begin{pmatrix} y_2\frac{\sqrt{y_1}}{\sqrt{1+y_2^2}} \\ \frac{\sqrt{y_1}}{\sqrt{1+y_2^2}} \end{pmatrix}$$

$$\mathbf{J}_{\phi^{-1}}(\mathbf{y}) = \begin{vmatrix} \frac{y_2}{2\sqrt{y_1}\sqrt{1+y_2^2}} & -\frac{\sqrt{y_1}}{(y_2^2+1)^{\frac{3}{2}}} \\ \frac{1}{2\sqrt{y_1}\sqrt{1+y_2^2}} & -\frac{y_2\sqrt{y_1}}{(y_2^2+1)^{\frac{3}{2}}} \end{vmatrix} = \frac{-1}{2(1+y_2^2)}$$

Using the previous theorem, the density of Y *is :*

$$f_Y(\mathbf{y}) = f_X(\phi^{-1}(\mathbf{y}))|\mathbf{J}(\mathbf{y})| = \underbrace{\frac{e^{-y_1/2}}{2}}_{Y_1 \sim Expo(\frac{1}{2})} \underbrace{\frac{1}{\pi(1+y_2^2)}}_{Y_2 \sim Cauchy}$$

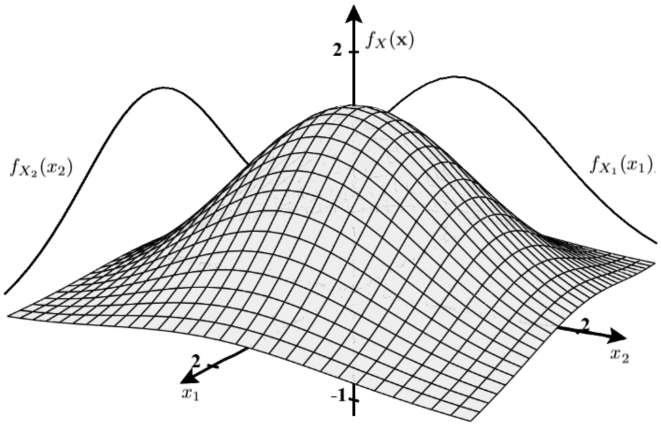


Figure 2.16: f_X Density of X

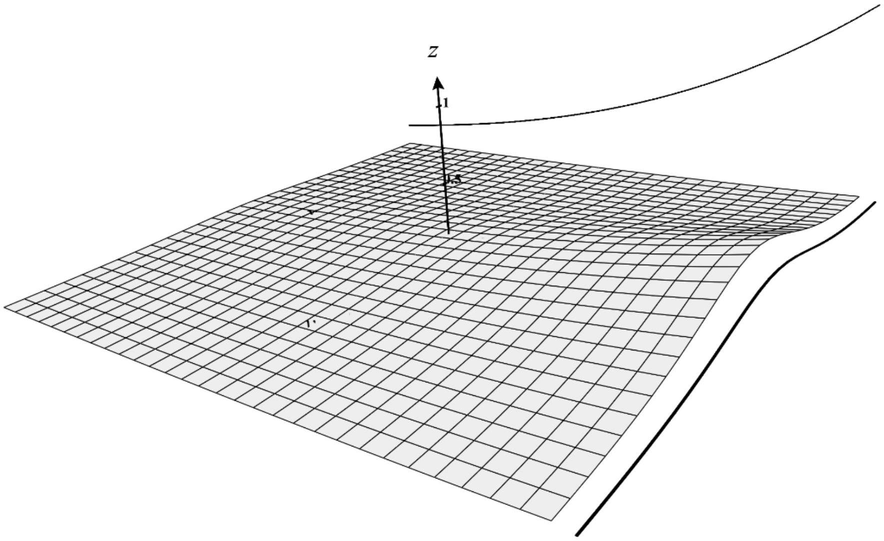


Figure 2.17: f_Y Density of Y

2.9 Independent random variables

The concept of independent random variables is similar to that of independent events. Indeed, if X, Y are two r.vs, then the two events $E_A = X \in A$ and $E_B = Y \in B \ \forall A, B \in \mathcal{B}(\mathbb{R})$ can be independent and in this case we speak of the independence of r.v.

Definition 20.

1- Two r.v X *and* Y *are independent* ($\perp$) *if their joint probability is the product of the two respective probabilities:*

$$\forall A, B \in \mathcal{B}(\mathbb{R}), \quad P(X \in A, Y \in B) = P(X \in A) \times P(Y \in B) \implies X \perp Y.$$

> *i.e.*
> $$P_{XY}(A,B) = P_X(A) \times P_Y(B), \quad \forall A, B \in \mathcal{B}(\mathbb{R})$$
> $$F_{XY}(x,y) = F_X(x) \times F_Y(y), \forall (x,y) \in R_X \times R_Y$$
>
> *2- Discrete case:* $p_{XY}(x,y) = p_X(x) \times p_Y(y)$
> *3- Continuous case:* $f_{XY}(x,y) = f_X(x) \times f_Y(y)$
>
> *In general:*
> $p_{X_1,X_2,..X_n}(x_1, x_2, \cdots, x_n) = \prod_{i=1}^{n} P(X_i = x_i)$
> $f_{X_1,X_2,..X_n}(x_1, x_2, \cdots, x_n) = \prod_{i=1}^{n} f_{X_i}(x_i)$

Intuitively, two random variables are independent if knowing the value of one of them does not change the probability of the other. Just like with events, it is sometimes easy to see that two random variables are independent just because they have no physical interactions between them. If for example we toss a coin $2N$ times, and we define X as the number of heads obtained in the N first tosses and Y in the N last tosses, as X and Y are the result of independent tosses so they are independent.

> **Proposition 7.**
> *If X and Y are two independent r.vs then*
>
> 1. *$f(X)$ and $g(Y)$ are also independent for all functions f and g.*
> 2. *$\mathbb{E}(XY) = \mathbb{E}(X)\mathbb{E}(Y)$*
> 3. *$\mathbb{V}(X+Y) = \mathbb{V}(X) + \mathbb{V}(Y)$*

Example 26. *Let X_1 and X_2 be two independent r.vs with mean 0 and variance σ. What is the covariance and the correlation coefficient of $X_1 + 2X_2$ and $3X_1 - X_2$?*

Solution
Let $U = X_1 + 2X_2$ and $V = 3X_1 - X_2$, by applying the properties of the covariance:

$$\begin{aligned}
Cov(U,V) &= Cov(X_1 + 2X_2, V) \\
&= Cov(X_1, V) + 2Cov(X_2, V) \\
&= [3Cov(X_1, X_1) - Cov(X_1, X_2)] + 2[3Cov(X_2, X_1) - Cov(X_1, X_1)] \\
Cov(X_1, X_2) &= 0 \text{ and } Cov(X_1, X_1) = \mathbb{V}(X_1) = \sigma^2 (\text{by independence}) \\
Cov(U,V) &= 3\sigma^2 - 2\sigma^2 = \sigma^2 \\
Corr(U,V) &= \frac{Cov(U,V)}{\sqrt{\mathbb{V}(U)\mathbb{V}(V)}} = \frac{\sigma^2}{\sqrt{5\sigma^2 10\sigma^2}} = 0.14
\end{aligned}$$

Example 27. *Calculate the variance of a d.r.v $X \rightsquigarrow \mathcal{B}in(n,p)$.*

Solution
Since the Binomial d.r.v represents the number of successes in n independent trials each one with a success parameter p, it can be written in the form: $X = \sum_{i=1}^{n} X_i$ such that the r.vs $X_i \rightsquigarrow \mathcal{B}er(p)$ are *independent*. Therefore:

$$\mathbb{V}(X) = \mathbb{V}(\sum_{i=1}^{n} X_i) = \sum_{i=1}^{n} \mathbb{V}(X_i) = n \times p \times (1-p)$$

2.10 Expectation and conditional variance

> **Definition 21.**
> *Consider the probability space $(\Omega, \mathcal{F}, P)$.*
> *1- Conditioned by an event:*

The expectation of X conditioned by a non-zero event H $(P(H) > 0)$

$$\mathbb{E}(X|H) = \frac{\mathbb{E}(\mathbb{1}_H X)}{P(H)} = \sum_x \frac{xP(\{X = x\} \cap H)}{P(H)}$$

2- Conditioned by a d.r.v Y:
The expectation of X conditioned by a d.r.v Y is a r.v noted $\mathbb{E}(X|Y)$ from Ω to $\mathbb{R}$ (or a map $f(Y)$ from R_Y to $\mathbb{R}$).
a- Y is a d.r.v:

$$\mathbb{E}(X|Y)(y) = \mathbb{E}(X|Y = y) = \sum_{R_X} xP(X = x|Y = y) \qquad (P(Y = y) > 0)$$

b- Y is a c.r.v:

$$\mathbb{E}(X|Y)(y) = \int_{R_X} x f_{X|Y}(x, y) dx$$

3- The variance of X conditioned by a r.v Y is defined:

$$\mathbb{V}(X|Y) = \mathbb{E}((X - \mathbb{E}(X|Y))^2|Y)$$

Example 28. *A box contains 30 balls: 5 out of them are white, 10 are black and 15 are red. Three balls are randomly drawn. Let the r.vs X and Yrepresent, respectively, the number of white and black drawn balls.*
1. Find the conditional pmfs of X given Y and of Y given X.
2. Calculate $\mathbb{E}(Y|X = 0)$

Solution
1. pmf.

$$P(X = x, Y = y) = \frac{C_5^x \times C_{10}^y \times C_{15}^{3-xy}}{C_{30}^3} \text{for } 0 \leq x \leq 3 \text{ and } 0 \leq y \leq 3 - x$$

$$P(X = x) = \sum_{y=0}^{3-x} P(X = x, Y = y) = \frac{C_5^x \times C_{25}^{3-x}}{C_{30}^3}$$

$$P(Y = y) = \sum_{x=0}^{3-y} P(X = x, Y = y) = \frac{C_{10}^y \times C_{20}^{3-y}}{C_{30}^3}$$

$$P(X = x|Y = y) = \frac{P(X = x, Y = y)}{P(Y = y)} = \frac{C_5^x \times C_{15}^{3-xy}}{C_{20}^{3-y}} \text{ for } 0 \leq x \leq 3 - y$$

$$P(Y = y|X = x) = \frac{P(X = x, Y = y)}{P(X = x)} = \frac{C_{10}^y \times C_{15}^{3-xy}}{C_{25}^{3-x}} \text{ for } 0 \leq y \leq 3 - x$$

2. Expectation

$$\mathbb{E}(Y|X = 0) = \sum_{y=0}^{3} yP(Y = y|X = 0) = 1.10$$

Example 29. *Consider two r.v X and Y having the joint function:*

$$f_{XY}(x, y) = \frac{12}{5} xy(3 - x - y) \textit{ for } 0 \leq x \textit{ and } y \leq 1$$

Calculate the conditional expectation of X given $Y = y$.

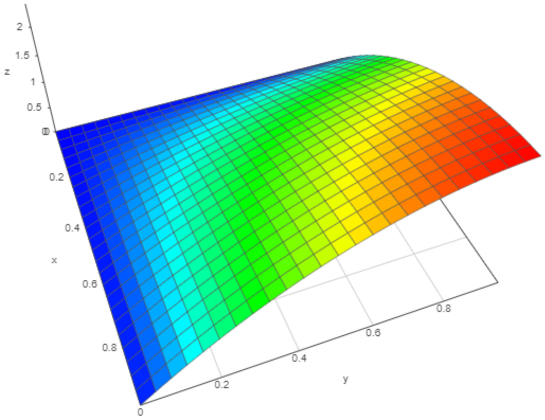


Figure 2.18: joint pdf of example 23

Solution

$$\begin{aligned}
\mathbb{E}(X|Y=y) &= \int_{R_X} x f_{X|Y}(x,y)dx \\
f_{X|Y}(x,y) &= \frac{f_{XY}(x,y)}{f_Y(y)} \\
f_Y(y) &= \int_{R_X} f_{XY}(x,y)dx = \int_0^1 \frac{12}{5}xy(3-xy)dx = \frac{2y\,(-3y+7)}{5} \\
f_{X|Y}(x,y) &= \frac{6x(3-x-y)}{7-3y} \\
\mathbb{E}(X|Y=y) &= \int_0^1 \frac{6x^2(3-xy)}{7-3y}dx = \frac{9-4y}{2(7-3y)}
\end{aligned}$$

2.10.1 Law of total expectation

This law allows us to calculate the unconditional expectation of a r.v X by conditioning X with respect to another r.v Y.

Theorem 2.
Let X, Y be two r.v defined on the same probability space, we have:

$$\mathbb{E}(X) = \mathbb{E}(\mathbb{E}(X|Y))$$

Discrete case:

$$\mathbb{E}(X) = \sum_y \mathbb{E}(X|Y=y)P(Y=y)$$

Continuous case:

$$\mathbb{E}(X) = \int_y \mathbb{E}(X|Y=y)f_Y(y)dy$$

Proof

$$\begin{aligned}
\sum_y \mathbb{E}(X|Y=y)P(Y=y) &= \sum_y \sum_x xP(X=x|Y=y)P(Y=y)\\
&= \sum_y \sum_x x\frac{P(X=x,Y=y)}{P(Y=y)}P(Y=y)\\
&= \sum_y \sum_x xP(X=x,Y=y)\\
&= \sum_x x \sum_y P(X=x,Y=y)\\
&= \sum_x xP(X=x)\\
&= \mathbb{E}(X)
\end{aligned}$$

In other words, the expectation of X is considered as the expectation of $\mathbb{E}(X|Y=y)$ with respect to Y for $y \in R_Y$ (expectation of the conditioned expectation which is a r.v) which is a function of R_Y in $\mathbb{R}$.

Example 30. *A student has to present his work in a conference, he chooses English with probability 2/3 and Spanish with probability 1/3. If the number of linguistic mistakes he may make in English follows a U(3,12) law and the number of errors in Spanish follows a U(4,20) law. What is the expectation of the number of errors he will make?*

Solution
Let X be the value of the number of errors he makes, and Y the chosen language.
$R_X = \mathbb{N}$ and $R_Y = \{0,1\}$ (0: English and 1: Spanish)

$$\begin{aligned}
\mathbb{E}(X) = &\sum_{y=0}^{1} \mathbb{E}(X|Y=y)P(Y=y)\\
&= \mathbb{E}(X|Y=0)P(Y=0) + \mathbb{E}(X|Y=1)P(Y=1)\\
&= 15/2 \times 2/3 + 24/2 \times 1/3 = 9
\end{aligned}$$

Example 31. *This example illustrates the use of the law of total expectation with recurrence. A person is in a room with three exits. The first exit (chosen with probability 1/2) leads to the outside.The second and the third (both chosen with probability 1/4) with 7m and 5m path long respectively. He returns back to the room by choosing them. What is the expectation of the total distance to leave the room.*

Solution
Let X be the r.v representing the crossed distance to exit to the outside, and Y the chosen exit.
$R_Y = \{1,2,3\}$.
$\mathbb{E}(X) = \sum_{R_Y} \mathbb{E}(X|Y=y)P(Y=y)$
$\mathbb{E}(X|Y=1) = 10$
$\mathbb{E}(X|Y=2) = 7 + \mathbb{E}(X)$
This is explained by the fact that when this person returns to the room, the same experiment (independent of the previous ones) is repeated.
$\mathbb{E}(X|Y=3) = 5 + \mathbb{E}(X)$
$\mathbb{E}(X) = 10 \times \frac{1}{2} + (7 + \mathbb{E}(X)) \times \frac{1}{4} + (5 + \mathbb{E}(X)) \times \frac{1}{4}$
$\mathbb{E}(X)$ =16.

Let's code!

```
# Code214.py

import numpy as np
from random import choices
```

```
def go_out():
    distance=0
    while True:
        s=choices([1,2,3],prob)[0]
        distance+=10 if s==1 else(7 if s==2 else 5)
        if(s==1):break
    return distance

n=100000
prob=[0.5,0.25,0.25]
d=[go_out() for _ in range(n)]

print("Expexted crossed distance: ",np.mean(d))

#______________________________   Output  ______________________________________
# Expexted crossed distance:  15.99408
```

The expectation of the geometric law

Another way to calculate the expectation of a geometric r.v with parameter p is to use recurrence. Let X be the r.v representing the number of coin tosses necessary to obtain the first head. p is the probability of success.
Let Y be the r.v representing the outcome of the first toss: 1 if head and 0 if tail.
Each time we get a failure ($Y = 0$), the same experiment is repeated (recurrence on X) with one more toss. When we get a success (Y =1), we stop.

$$\mathbb{E}(X) = \mathbb{E}(X|Y = 0)P(Y = 0) + \mathbb{E}(X|Y = 1)P(Y = 1)$$
$$\mathbb{E}(X|Y = 0) = 1 + \mathbb{E}(X)$$
$$\mathbb{E}(X|Y = 1) = 1$$
$$\text{So } \mathbb{E}(X) = (1 + \mathbb{E}(X))(1 - p) + p \text{ which gives } \mathbb{E}(X) = 1/p.$$

Example 32. *Each year, a park opens its doors for two months (60 days) given that it does not rain. During this period, every day, there is a probability p of raining. Consider the two r.vs: Y the number of days the park opens its doors and X the total number of visitors during the Y days. If $X|Y \rightsquigarrow Poisson(100Y)$ what is the distribution and the expectation of Y. Calculate the expectation of X.*

Solution

$Y \rightsquigarrow \mathcal{B}in(60, 1 - p)$ and $\mathbb{E}(Y) = n(1 - p)$
Since $X|Y \rightsquigarrow Poisson(100Y)$ then $\mathbb{E}(X|Y) = 100Y$
$\mathbb{E}(X) = \mathbb{E}(\mathbb{E}(X|Y)) = \mathbb{E}(100Y) = 100\mathbb{E}(Y) = 100 \times 60 \times (1 - p) = 6000(1 - p)$

Let’s code!

```
#Code215.py

import numpy as np
from sympy import *
from sympy.stats import Poisson, Binomial, E
from sympy import Symbol


p, n, mu = Symbol("p"), Symbol("n"), Symbol("mu")
Y = Binomial("Y",n,1-p);  XgivenY = Poisson("X",mu*Y)
print('E(Y)    = ', E(Y))
print('E(X|Y)  = ', E(XgivenY))
print('E(X)    = ', E(E(XgivenY)))

p0, mu0, n0 = 0.3, 100, 60
print('Resultat Symbolique :')
print("Esperance de Y        : ", E(Y).subs([(n,n0),(p,p0)]).doit())
print("Esperance de X|Y      : ", E(E(XgivenY)).subs([(n,n0),(p,p0),(mu,mu0)]).doit())

N        = 1000
Ye       = np.random.binomial(n0 ,1-p0, N)
XgivenYe = np.random.poisson(mu0*Ye, (N, len(Ye)))

print('Resultat Empirique :')
print("Esperance de Ye       : ", Ye.mean())
print("Esperance de X|Ye     : ", XgivenYe.mean())

#______________________________   Output  ______________________________________
# E(Y)    = Sum(Piecewise((_k*p**(-_k + n)*(1 - p)**_k*binomial(n, _k),
#                            (_k >= 0) & (_k <= n)), (0, True)), (_k, 0, n))
# E(X|Y) = mu*Y
# E(X)   = Sum(Piecewise((_k*mu*p**(-_k + n)*(1 - p)**_k*binomial(n, _k),
#                            (_k >= 0) & (_k <= n)), (0, True)), (_k, 0, n))
# Resultat Symbolique :
# Esperance de Y           :  41.99
# Esperance de X|Y         :  4199.99
# Resultat Empirique :
# Esperance de Ye          :  42.13
# Esperance de X|Ye        :  4213.06
```

The expectation and conditional variance preserve the properties of the classical expectation and variance. Some properties are specific to this type (the conditional):

> **Proposition 8.** *Properties of conditional expectation and variance*
> *1-* $X \perp Y \implies \mathbb{E}(X|Y) = \mathbb{E}(X)$
> *2-* $X \perp Y \implies \mathbb{E}(XY|Z) = \mathbb{E}(X|Z)\mathbb{E}(Y|Z)$
> *3-* $\mathbb{E}(\mathbb{E}(X|Y)|Y) = \mathbb{E}(X)$
> *4-* $\mathbb{E}(\mathbb{E}(X|Y,Z)|Y) = \mathbb{E}(X|Y)$
> *5-* $\mathbb{V}(X|Y) = \mathbb{E}(X^2|Y) - (\mathbb{E}(X|Y))^2$
> *6-* $\mathbb{V}(X) = \mathbb{E}(\mathbb{V}(X|Y)) - (\mathbb{V}(\mathbb{E}(X|Y))$

2.11 Sequence of r.vs and Convergence

2.11.1 Sequence of r.vs (s.r.v)

> **Definition 22.** *A sequence of r.vs is an indexed family $(X_i)_{i>0}$ such that $\forall i$, X_i is a r.v. The sequence is denoted X_n^* (or $X^* = (X_i)_{i>0}$).*

Consider the sequence of r.v's $X_n^* = (X_i)_{1 \le i \le n}$ such that $X_i \sim \mathcal{B}er(1/2)$, then X_n^* is a random sequence of bits (binary digits). For example X_{16}^* =0111010101001001.

Example 33. *Consider the sequence of r.v $X^* = (X_i)_{i \ge 0}$ such that $Y \sim \mathcal{B}er(2/3)$ and*

$$X_i = Y\frac{1}{i+1} + (1-Y)\frac{1}{i+2} = \frac{1}{i+2-Y}$$

a- Give X_{10}^*
b- Are X_n independent?
c- Find the pmf (p_{X_n}) and the cdf (F_{X_n}) of X_n, for $n > 0$.

Solution
a- First we generate a sequence of 10 r.v of Bernouli Y_i

i	0	1	2	3	4	5	6	7	8	9
Y_i	0	0	1	0	1	1	1	0	0	1
X_i	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{6}$	$\frac{1}{7}$	$\frac{1}{9}$	$\frac{1}{10}$	$\frac{1}{10}$

a- Since $\forall i \neq j, X_i \not\perp Y \vee X_j \not\perp Y$, then $X_i \not\perp X_j$. For example for X_1 and X_2:

$$P(X_1 = 1/3, X_2 = 1/4) = P(Y = 0) \qquad = \frac{1}{3}$$

$$P(X_1 = 1/3) \cdot P(X_2 = 1/4) = P(Y = 0) \cdot P(Y = 0) \qquad = \frac{1}{9}$$

So $X_1 \not\perp X_2$

R_{X_n}	p_{X_n}	F_{X_n}
$\{\frac{1}{n+2}, \frac{1}{n+1}\}$	$\frac{1}{3}\mathbb{1}_{\{\frac{1}{n+2}\}} + \frac{2}{3}\mathbb{1}_{\{\frac{1}{n+1}\}}$	$\frac{1}{3}\mathbb{1}_{[\frac{1}{n+2},\frac{1}{n+1}[} + \mathbb{1}_{[\frac{1}{n+1},\infty[}$

2.11.2 Sum of independent sequences of r.v

Consider the sequence of independent r.v $X^* = (X_n)_{n>0}$. Let S_n be the r.v which is equal to the sum of the first n r.vs X_i of X^*.

S_n	$f_{S_n}(x)$	$\mathbb{E}(S_n)$	$\mathbb{V}(S_n)$
$\sum_{i=1}^{n} X_i$	$f_{X_1}(x)*f_{X_2}(x)*\cdots*f_{X_n}(x)$	$\sum_{i=1}^{n} \mathbb{E}(X_i)$	$\sum_{i=1}^{n} \mathbb{V}(X_i)$

The probability distribution of the sum of a sequence of independent r.vs is the *convolution* of their individual distributions which is denoted by $*$.

$$\forall X, Y \quad X \perp Y \implies f_{X+Y} = f_X * f_Y.$$

Discrete case:

$$p_X(z) * p_Y(z) = \sum_{w \in R_X} p_X(w) p_Y(z-w) = \sum_{w \in R_Y} p_Y(w) p_X(z-w)$$

Continuous case:

$$f_X(z) * f_Y(z) = \int_{-\infty}^{\infty} f_X(w) f_Y(z-w) dw = \int_{-\infty}^{\infty} f_Y(w) f_X(z-w) dw$$

Example 34. *Let X and Y be two r.vs. X represents a die toss and Y represents a coin flip. Give the pmf of $S_1 = X + Y$ and $S_2 = X + 2Y$*

Solution

X	1	2	3	4	5	6
p_X	1/6	1/6	1/6	1/6	1/6	1/6

Y	0	1
p_Y	2/3	1/3

The rank of S_1 is $R_{S_1} = \{1, 2, 3, 4, 5, 6, 7\}$

$p_{S_1}(k) = \sum_h p_x(k-h) \times p_y(h)$, so that $h \in R_Y$ and $k - h \in R_X$

$p_{S_1}(1) = p_x(1)p_y(0) = 1/6 \times 2/3 = 1/9$

$p_{S_1}(2) = p_x(1)p_y(1) + p_x(2)p_y(0) = 1/6 \times 1/3 + 1/6 \times 2/3 = 1/6$

$p_{S_1}(7) = p_x(6)p_y(1) = 1/6 \times 1/3 = 1/18$

S_1	1	2	3	4	5	6	7
p_{S_1}	1/9	1/6	1/6	1/6	1/6	1/6	1/18

$S_2 = X + 2Y = S_1 + Y$

The rank of S_2 is $R_{S_2} = \{1, 2, 3, 4, 5, 6, 7, 8\}$

$p_{S_2}(k) = \sum_h p_{S_1}(k-h) \times p_y(h)$, such that $h \in R_Y$ and $k - h \in R_{S_1}$

$p_{S_2}(1) = p_{S_1}(1)p_y(0) = 1/9 \times 2/3 = 2/27$

$p_{S_2}(2) = p_{S_1}(2)p_y(0) + p_{S_1}(1)p_y(1) = 1/6 \times 2/3 + 1/9 \times 1/3 = 4/27$

$p_{S_2}(3) = p_{S_1}(3)p_y(0) + p_{S_1}(2)p_y(1) = 1/6$

$p_{S_2}(7) = p_{S_1}(7)p_y(0) + p_{S_1}(6)p_y(1) = 1/18 \times 2/3 + 1/6 \times 1/3 = 5/54$

$p_{S_2}(8) = p_{s_1}(7)p_y(1) = 1/18 \times 1/3 = 1/54$

S_2	1	2	3	4	5	6	7	8
p_{S_2}	2/27	4/27	1/6	1/6	1/6	1/6	5/54	1/54

Example 35. *Let X and $Y \rightsquigarrow \mathcal{U}(0,1)$. Find the distribution of $Z = X + Y$*

Solution

$$\begin{aligned} f_Z(z) &= \int_{-\infty}^{+\infty} f_X(z-y)f_Y(y)dy = \int_0^1 f_X(z-y)dy \\ &= \int_0^z dy = z \text{ if } 0 \le z \le 1 \\ &= \int_{z-1}^1 dy = 2-z \text{ if } 1 < z \le 2 \\ &= z\mathbb{1}_{[0,1]}(z) + (2-z)\mathbb{1}_{[1,2]} \end{aligned}$$

So $z \rightsquigarrow \mathcal{T}ri(0,2,1)$

2.11.3 Convergence

Consider a sequence of r.v's $X^* = (X_n)_{n>0}$ and a r.v X.

> **Definition 23.**
> *X^* converges in distribution to X (denoted $X_n \xrightarrow{d} X$) if:*
>
> $$\lim_{n\to\infty} F_{X_n}(x) = F_X(x), \quad \forall x, \quad F_X(x) \textit{ is continuous}$$

Example 36. *Let X^* be a s.r.v such that $F_{X_n}(x) = \mathbb{1}_{\{x>0\}}(1-(1-q/n)^{nx/p}) \quad p,q>0$. Show that $X_n \xrightarrow{d} X \sim \mathcal{E}xp(p/q)$*

Solution
Consider the r.v $X \sim \mathcal{E}xp(p/q)$,
i- In case $x \le 0$, we have

$$F_{X_n}(x) = F_X(x) = 0, \qquad \forall n \ge 2$$

ii- In case $x \ge 0$, we have

$$\lim_{n\to\infty} F_{X_n}(x) = \lim_{n\to\infty}\left(1-\left(1-\frac{q}{n}\right)^{nx/p}\right) = 1 - \underbrace{\lim_{k\to\infty}\left(1-\frac{1}{k}\right)^{k\frac{p}{q}x}}_{k=\lfloor n/q\rfloor} = 1-e^{-\frac{p}{q}x} = F_X(x), \quad \forall x$$

> **Theorem 3.**
> *Let X^* be a s.r.v and X a d.r.v. Suppose that X and X_n (for all n) are non-negative integer values, ($R_X \subset \mathbb{N}$, $R_{X_n} \subset \mathbb{N}, \forall n \in \mathbb{N}^*$). So,*
>
> $$X_n \xrightarrow{d} X \iff \lim_{n\to\infty} p_{X_n}(k) = p_X(k), \quad \forall k \in \mathbb{N}$$

Proof.
We prove both directions of the implication.
i- Since X is an integer valued r.v, its cdf,$F_X(x)$, is continuous at each $x \in \mathbb{R}/\mathbb{N}$.
If $X_n \xrightarrow{d} X$, then

$$\lim_{n\to\infty} F_{X_n}(x) = F_X(x), \qquad \forall x \in \mathbb{R}/\mathbb{N}.$$

For $k \in \mathbb{N}$, we have

$$\begin{aligned}
\lim_{n\to\infty} P_{X_n}(k) &= \lim_{n\to\infty}\left[\underbrace{F_{X_n}\left(k+\frac{1}{2}\right) - F_{X_n}\left(k-\frac{1}{2}\right)}_{X_n \quad \text{an integer valued}}\right] \\
&= \lim_{n\to\infty} F_{X_n}\left(k+\frac{1}{2}\right) - \lim_{n\to\infty} F_{X_n}\left(k-\frac{1}{2}\right) \\
&= \underbrace{F_X\left(k+\frac{1}{2}\right) - F_X\left(k-\frac{1}{2}\right)}_{(\text{since } X_n \xrightarrow{d} X)} \\
&= \underbrace{P_X(k)}_{X i.v.r.v}
\end{aligned}$$

ii- Suppose that: $\lim_{n\to\infty} p_{X_n}(k) = p_X(k), \quad \forall k \in \mathbb{N}$. So

$$\begin{aligned}
\forall x \in \mathbb{R}, \quad \lim_{n\to\infty} F_{X_n}(x) &= \lim_{n\to\infty} P(X_n \le x) \\
&= \overbrace{\lim_{n\to\infty}\sum_{k=0}^{\lfloor x\rfloor} P_{X_n}(k) = \sum_{k=0}^{\lfloor x\rfloor}\lim_{n\to\infty}}^{x fixed, \quad \{0,1,\cdots,\lfloor x\rfloor\}\text{finished}} \underbrace{P_{X_n}(k) = \sum_{k=0}^{\lfloor x\rfloor} P_X(k)}_{(\text{hypothesis})} \\
&= P(X \le x) = F_X(x).
\end{aligned}$$

□

Example 37. *Let X^* be a s.r.v such that $X_n \sim \mathcal{B}in(n, \lambda/n), \forall n \in \mathbb{N}, n > \lambda$, and $\lambda > 0$ is constant.*
Show that $X_n \xrightarrow{d} X \sim \mathcal{P}ois(\lambda)$.

Solution

$$\begin{aligned}
\lim_{n\to\infty} P_{X_n}(k) &= \lim_{n\to\infty} C_n^k \left(\frac{\lambda}{n}\right)^k \left(1-\frac{\lambda}{n}\right)^{nk} \\
&= \frac{\lambda^k}{k!}.\lim_{n\to\infty}\left(\underbrace{\left[\frac{n(n-1)(n-2)...(n-k+1)}{n^k}\right]}_{1(n\to\infty)} \underbrace{\left(1-\frac{\lambda}{n}\right)^n}_{e^{-\lambda}(n\to\infty)} \underbrace{\left(1-\frac{\lambda}{n}\right)^{-k}}_{1(n\to\infty)}\right). \\
&= \frac{e^{-\lambda}\lambda^k}{k!} = P_X(k) \quad \text{and} \quad X \sim \mathcal{P}ois(\lambda)
\end{aligned}$$

Definition 24.
X^* converges in probability to X (denoted $X_n \xrightarrow{p} X$) if :

$$\lim_{n\to\infty} P\big(|X_n - X| \ge \epsilon\big) = 0, \forall \epsilon > 0$$

Example 38. *Let X^* be a s.r.v such that $X_n \sim \mathcal{E}xp(n)$, show that $X_n \xrightarrow{p} 0$.*

Solution
Let $\epsilon > 0$, then $\lim_{n\to\infty} P\big(|X_n - 0| \ge \epsilon\big) = \underbrace{\lim_{n\to\infty} P\big(X_n \ge \epsilon\big)}_{X_n \ge 0} = \underbrace{\lim_{n\to\infty} e^{-n\epsilon}}_{X_n \sim \mathcal{E}xp(n)} = 0$

Definition 25.
X_n^* converges almost sure to X (noted $X_n \xrightarrow{a.s} X$) if :

$$P\left(\left\{\omega \in \Omega : \lim_{n\to\infty} X_n(\omega) = X(\omega)\right\}\right) = 1$$

Example 39. *Consider the sequence X^* and the r.v X defined on the probability space ($\Omega = [0,1], P([a,b]) = b-a$) for all $0 \leq a \leq b \leq 1$ by:*

$$X_n(\omega) = \mathbb{1}_{\{0\leq\omega<\frac{n+2}{3n}\}} \quad \text{and} \quad X(\omega) = \mathbb{1}_{\{0\leq\omega<\frac{1}{3}\}}$$

Consider the following set:

$$A = \left\{\omega \in \Omega : \lim_{n\to\infty} X_n(\omega) = X(\omega)\right\}$$

- Determine the event A and its probability. What conclusion can you make?

Solution
$i-$ If $\omega \in [0, \frac{1}{3}[, \omega \in [0, \frac{n+2}{3n}]$ therefore $X_n(\omega) = X(\omega) = 1$, then $[0, \frac{1}{3}[\subset A$
$ii-$ If $\omega \in]\frac{1}{3}, 1[$

$$\underbrace{\exists n_\omega \in \mathbb{N}, \text{such that } \frac{1}{n_\omega} < (3\omega-1)/2,}_{propertyof Archimedes}$$

we have $\dfrac{n_\omega + 2}{3n_\omega} < \omega$, so $X_{n_\omega}(\omega) = X(\omega) = 0$ then $]\frac{1}{3}, 1[\subset A$
From i and ii, $\Omega = A \cup \{\frac{1}{3}\}$ and $P(\frac{1}{3}) = 0$, then $P(A) = P(\Omega) = 1$
So $X_n \xrightarrow{a.s} X$
A sequence can converge to one type but not to another. Some of these types of convergence are stronger than others.

Definition 26.
Convergence type A is stronger than convergence type B (B is weaker than A), if type A implies type B.

$$X_n^* \xrightarrow{A} X \implies X_n^* \xrightarrow{B} X$$

Theorem 4.
The almost sure convergence is stronger than the probability convergence which is stronger than the distribution convergence. Let X^* be a s.r.v, we have :

$$X_n \xrightarrow{p.s} X \implies X_n \xrightarrow{p} X \implies X_n \xrightarrow{d} X$$

2.11.4 Large numbers law

Theorem 5. *Weak law of large numbers*
Let X^* be a s.r.v such that X_n are i.i.d, of finite mean $\mathbb{E}(X_n) = \mu < \infty$. Then the empirical mean $\overline{X}_n = \frac{X_1+X_2+\cdots+X_n}{n}$ tends in probability towards μ ($\overline{X}_n \xrightarrow{p} \mu$).

$$\lim_{n\to\infty} P\big(|\overline{X}_n - \mu| \geq \epsilon\big) = 0, \forall\epsilon > 0$$

Let's code!

```
#Code216.py

import numpy as np
from numpy.random import geometric, binomial

# Large number law for s.r.v
def generate(law, args, k):
    return np.sum([i*(law(*args)==i).sum() for i in range(100)])/k

def plotter(E):
    import matplotlib.pyplot as plt

    plt.plot(range(N), E, linewidth=0.5,label=str(N))
    plt.xlabel('N')
    plt.ylabel('Empirical expectation')
    plt.show()

p, N, n = 1/12, 1000, 100

E1 = [generate(geometric, [p,k*10], k*10)       for k in range(N)] ;plotter(E1)
E2 = [generate(binomial , [n,p, k*10 ], k*10) for k in range(N)] ;plotter(E2)

print('Theoretical Expectation - Binomial(np)= ', n*p , ', - Geometric  (1/p)= ', 1/p)

#________________________________   Output  _______________________________________
#Theoretical Expectation - Binomial(np)=  8.333333333333332 , - Geometric  (1/p)=  12.0
```

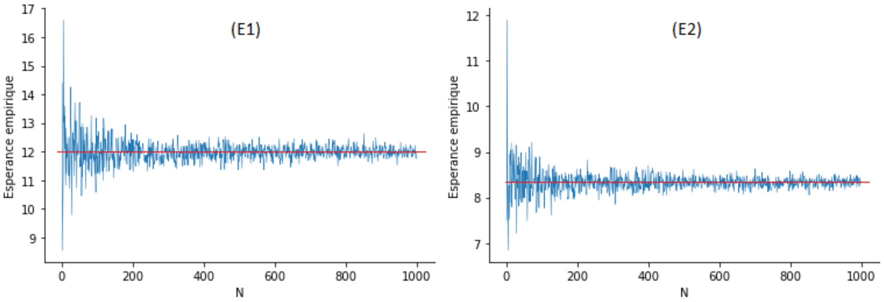


Figure 2.19: WLLN for the Geometric law(E1) and Binomial(E2) In red is the theoretical mean

Let's code!

```
#Code217.py

import numpy as np
from numpy.random import geometric
import matplotlib.pyplot as plt
from fractions import Fraction
from scipy.stats import geom

N, M = 100000, 3000 # N: number of generated r.v,  M: the limit
parameters=[1/70,1/30,1/12,1/2]

# generate: generates N geometric r.v samples for different parameters
# and returns their frequencies
def generate():
    f=[]
    for i in range(len(parameters)):
        z= geometric(parameters[i], size=N)
        somme=np.array([(z==k).sum() for k in range(M)])/N
        f.append(somme)
    return f

freqs = generate()
esps =[f*range(M) for f in freqs ]

for i in range(len(parameters)):
    plt.plot(range(1,199), freqs[i][1:199], color='green', linewidth=1,label="P="
             +str(Fraction(parameters[i]).limit_denominator())+" E="+str(round(np.sum(esps[i]),2))
     )
    plt.legend(); plt.xlabel('i'); plt.ylabel('Frequence')
    plt.plot(range(1,199), geom.pmf(range(1,199), parameters[i]),linewidth=0.7, color='r')
    plt.show()
```

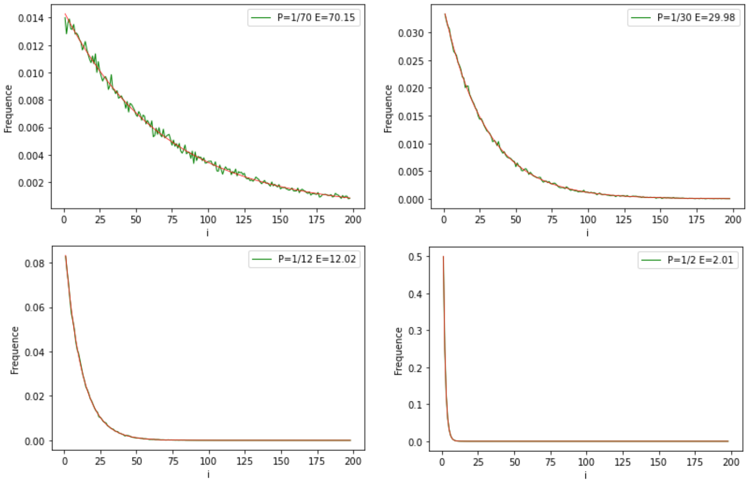


Figure 2.20: Empirical frequency for different values of p

2.11.5 Central limit theorem

This theorem is related to the *Normal* distribution. It indicates that the sum of n independent r.vs having the same probability distribution of mean μ and standard deviation σ, has approximately a *Normal* distribution of mean $n\mu$ and standard deviation $\sigma\sqrt{n}$ for a large enough value of n :

Theorem 6. *TCL*
Consider X_n^* such that X_i are i.i.d with finite mean $\mathbb{E}(X_i) = \mu$ and variance $\mathbb{V}(X_i) = \sigma^2$, so $\hat{S}_n$ (S_n standard : $\hat{S}_n = \frac{S_n - \mathbb{E}(S_n)}{\sqrt{\mathbb{V}(S_n)}} = \frac{S_n - n\mu}{\sqrt{n}\sigma}$) converges in distribution to a r.v which follows the standard normal distribution.

$$X_n^*/\ X_i iid, \quad \mathbb{E}(X_i) < \infty, \quad \mathbb{V}(X_i) < \infty \implies \hat{S}_n \xrightarrow{d} Z \rightsquigarrow \mathcal{N}(0,1)$$

$$\lim_{n\to\infty} F_{\hat{S}_n}(x) = \lim_{n\to\infty} P(\frac{X_1 + X_2 + \cdots + X_n - n\mu}{\sigma\sqrt{n}} \leq x) = F_Z(x) = \phi(x) \quad \forall x$$

Let's code!

```
#Code218.py

import numpy as np
from numpy.random import geometric
import matplotlib.pyplot as plt
import math
from scipy.stats import norm

def generate(p,k,mu,sigma):
    z = geometric(p, size=k)
    s = (np.sum(z)-(k*mu))/(sigma*math.sqrt(k))
    return s

def plotter(result):
    # Central limit theorem using sampling as histogram
    M = max(np.abs(result))
    ls = np.linspace(start=-M, stop=+M,num=50 + 1, endpoint=True)
    plt.hist(result,bins=ls,density=1,histtype='step',label="Histogramme")

    # Central limit theorem using theoretic approximation (Normal distribution)
    mu, variance = 0, 1
    sigma = math.sqrt(variance)
    x = np.linspace(mu - 5*sigma, mu + 5*sigma, 100)
    plt.plot(x, norm.pdf(x, mu, sigma),linewidth=0.7, color='r')
    plt.show()

def TCL(N):
    k, p = 1000, 1/12
    mu, sigma = 1/p, math.sqrt((1-p)/(p**2))
    result = [generate(p,k,mu,sigma) for _ in range(N)]
    plotter(result)

TCL(100000)
```

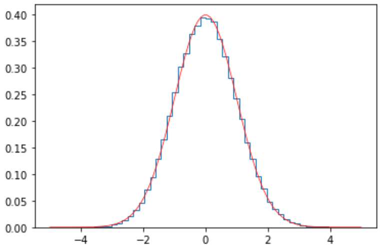


Figure 2.21: TCL applied to the geometric law

Example 40. *Let Y be the number of heads obtained by tossing a coin 40 times.*
Find $P_Y(20)$ the probability that $Y = 20$.
Find the probability of getting 20 heads
Find the probability of getting 30 heads

Solution

1- $Y = S_n$ is the sum of n i.i.d r.vs $(\forall i, X_i \rightsquigarrow \mathbb{B}er(1/2))$.
Since $\hat{S}_n \xrightarrow{d} Z \rightsquigarrow \mathcal{N}(0,1)$, then $P_{\hat{S}_n}(k)$ can be approximated by $P(|Xk| < c)$ such that $2c$ the length of the interval centered by k must not contain any other values of $R_{\hat{S}_n}$ except k. We choose $c = \frac{0.5}{\sqrt{n}\sigma}$.

$$\begin{aligned}
P_{S_{40}}(20) &= P_{\hat{S}_{40}}(\frac{20-(40\times 0.5)}{\sqrt{40}\times 0.5}) = P_{\hat{S}_{40}}(0)\\
&= P(-0.158 \leq \hat{S}_{40} \leq +0.158)\\
&\approx P(-0.158 \leq X \leq +0.158) = P(|X| < 0.158)\\
&= \phi(+0.158) - \phi(-0.158) = 0.1272
\end{aligned}$$

Direct calculation gives: $P(Y = 20) = C_{40}^{20} \times (\frac{1}{2})^{40} = 0.1268$

$$\begin{aligned}
P_{S_{40}}(30) &= P_{\hat{S}_{40}}(\frac{30-(40\times 0.5)}{\sqrt{40}\times 0.5}) = P_{\hat{S}_{40}}(3.16)\\
&= P(3.16 - 0.158 \leq \hat{S}_{40} \leq 3.16 + 0.158) = P(3.002 \leq \hat{S}_{40} \leq 3.318)\\
&\approx P(3.002 \leq X \leq 3.318) = P(|X - 3.16| < 0.158)\\
&= \phi(3.318) - \phi(3.002) = 0.0008
\end{aligned}$$

2.12 Application: linear regression

Linear regression is widely used in machine learning and data analysis. It consists in establishing a linear relationship (model) between a variable Y (called dependent variable) and one or more variables X (called independent variables). In its most basic form, a single variable X is used to predict the value of variable Y.
In case X and Y are Normal bivariate r.vs, the linear regression model denotes a model in which the conditional expectation of Y given X is an affine function:

$$\mathbb{E}(Y|X) = a + bX$$

this is equivalent to:

$$Y = a + bX + \epsilon$$

such that ϵ is a Normal r.v representing the error and it is independent of X with $\mathbb{E}(\epsilon|X) = 0$.

Proof

1- Assume that we have $Y = a + bX + \epsilon$ such that ϵ is a Normal r.v representing the error and it is independent of X with $\mathbb{E}(\epsilon|X) = 0$, then :

$$\mathbb{E}(Y|X) = \mathbb{E}(Y|a) + \mathbb{E}(bX|X) + \mathbb{E}(\epsilon|X) = a + bX$$

2- We have $\mathbb{E}(Y|X) = a + bX$, assume that $\epsilon = Y - a + bX$

As Y, X are bivariate Normal then ϵ is Normal r.v. verifying

$$\mathbb{E}(\epsilon|X) = \mathbb{E}(Y|X) - \mathbb{E}(a + bX) = 0$$

We have

$$\mathbb{E}(\epsilon) = \mathbb{E}(\mathbb{E}(\epsilon|X)) = 0$$
$$\mathbb{E}(\epsilon X) = \mathbb{E}(\mathbb{E}(\epsilon X|X)) = \mathbb{E}(X\mathbb{E}(\epsilon|X)) = 0 = \mathbb{E}(\epsilon)\mathbb{E}(X)$$

Then $\epsilon \perp X$ (independent).

This result allows us to define the value of a and b in terms of $\mathbb{E}(X)$, $\mathbb{E}(Y)$, $\mathbb{V}(X)$ and $Cov(X, Y)$.

As ϵ and X are independent, they are uncorrelated, then

$$Cov(X, Y) = Cov(X, a + bX + \epsilon) = Cov(X, a) + bCov(X, X) + Cov(X, \epsilon) = b\mathbb{V}(X)$$

So:

$$b = \frac{Cov(X, Y)}{\mathbb{V}(X)}$$

we have

$$\mathbb{E}(Y) = \mathbb{E}(\mathbb{E}(Y|X)) = \mathbb{E}(a + bX) = a + b\mathbb{E}(X)$$

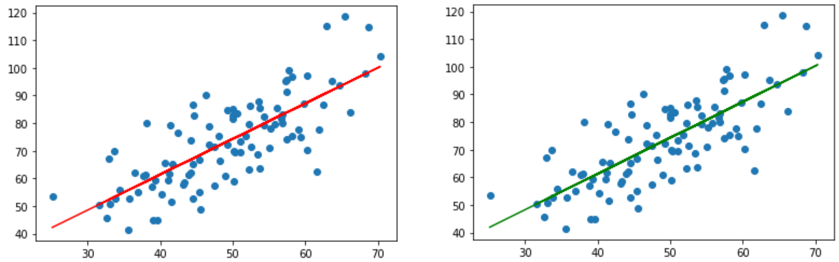
Then

$$a = \mathbb{E}(Y) - b\mathbb{E}(X) = \mathbb{E}(Y) - \frac{Cov(X, Y)}{\mathbb{V}(X)}\mathbb{E}(X)$$

In the following code we apply the result obtained in this section and compare it to the result obtained by the *linear_model* module of the *sklearn* library which uses the least squares method.

Let's code!

```
#Code219.py

import numpy as np
import matplotlib.pyplot as plt

# sklinreg: performs linear regression using sklearn package
def sklinreg():
    import pandas as pd                        # To read data
    from sklearn.linear_model import LinearRegression
    data = pd.read_csv(’data.csv’)             # load data set
    X = data.iloc[:, 0].values.reshape(-1, 1)  # values converts it into a numpy array
    Y = data.iloc[:, 1].values.reshape(-1, 1)  # -1 means that calculate the dimension of rows,
     but have 1 column
    linear_regressor = LinearRegression()      # create object for the class
    m=linear_regressor.fit(X, Y)               # perform linear regression
    Y_pred = linear_regressor.predict(X)       # make predictions
    return X,Y,m.intercept_[0],m.coef_[0][0],Y_pred

# analytic: performs linear regression using the results of the application
def analytic(X0,Y0):
    X=[X0[i][0] for i in range(len(X0))]
    Y=[Y0[i][0] for i in range(len(Y0))]
    Ex,Ey,Vx=np.mean(X),np.mean(Y),np.var(X)
    Cov=np.cov(X,Y)[0][1]
    b=Cov/Vx; a=Ey-Ex*b
    predictedY=a+b*np.array(X)
    return a,b,predictedY

def plotter(X,Y,predictedY,col):
    plt.scatter(X, Y)
    plt.plot(X, predictedY, color=col)
    plt.show()

X,Y,ska,skb,skY_pred=sklinreg()
print("SKlearn results: a=",ska,"  b=",skb)

anala,analb,analY_pred=analytic(X,Y)
print("Analytic: ""a=",anala,"  b=",analb)

plotter(X,Y,skY_pred,'red')
plotter(X,Y,analY_pred,'green')

#______________________________   Output  ______________________________________
# SKlearn results: a= 9.908606190326537   b= 1.2873573700109313
# Analytic: a= 9.263291199945208   b= 1.3004936697049203
```



SKLearn

Analytique

Figure 2.22: Linear regression

2.13 Exercises

Exercise 1. *Two counterfeit banknotes were concealed in a packet of three real banknotes. To withdraw them, the banknotes are checked one by one in a random order. Let X be the random variable representing the minimum number of banknotes that must be checked to find the two fake banknotes.*
1. Give the mass function of X.
2. Calculate its expectation and its variance.
3. Code the exercise.

Exercise 2. *Consider the following function:*

$$f(x) = cx^2(1-x)\mathbb{1}_{0<x<1}$$

1. Determine the value of c so that f is a density function.
2. Suppose that f is the density function of the c.r.v X which represents the volume of yearly precipitation over an area on which a dam will be built. What capacity should be provided for this dam so that the probability of overflow in a given year does not exceed 1%?

Exercise 3. *Knowing that the number of major earthquakes each year in the world follows Poisson law with an average of 4.2 earthquakes a year. What is the probability that there will be more than 7 major earthquakes in the world next year?*

Exercise 4. *Let X and $Y \rightsquigarrow \mathcal{U}(a,b)$*
1. Find the distribution of $Z = X + Y$
2. Write the corresponding code.

Exercise 5. *Let X ,Y be two r.vs following $\mathcal{B}er(1/2)$ law.*
1. Find the distribution of the r.v $|X-Y|$.
2. Find the distribution of the r.v $|X-Y|$ when X and Y follow $\mathcal{B}er(p)$.
3. Write the corresponding code.

Exercise 6. *Let $X_1, X_2, \cdots, X_n$ be n exponential r.vs with respective rates $\lambda_1, \lambda_2, \cdots, \lambda_n$. Show that $min(X_1, X_2, \cdots, X_n)$is an exponential r.v of rate $\sum_{i=1}^{n} \lambda_i$.*

Exercise 7. *Let X_1 and X_2 be two exponential r.v of respective rates λ_1 and λ_2. What is the probability that X_1 is less than X_2?*

Exercise 8. *An urn contains 10 balls numbered 1 to 10. One ball at a time is repeatedly drawn (with replacement). Let the r.v X representing the number of draws until the appearance of number 3, and the r.v Y representing the number of draws until the appearance of number 7.*
1. What is the joint mass function of $P(X = x, Y = y)$.
2. Calculate $P(X = 2|Y = 4)$

Exercise 9. *Buffon's experiment consists in throwing needles of length h on a plane on which parallel lines separated by the same distance l are drawn. The number of needles crossing these lines are observed.*
1. Let X be the r.v which represents the distance from the center of the needle to the nearest line; and let θ be the r.v representing the angle of the needle with respect to the lines of the plane. Find the joint distribution of X and θ.
2. What is the probability that a needle crosses a line?

Exercise 10. *An electronic device contains two important components. The device fails if one of the two components fails. The joint density function of the lifetimes X and Y (in years) of the two components is:*

$$f(x,y) = \frac{1}{12}(1+x+y)\mathbb{1}_{0<x,y<2}$$

What is the probability that the electronic device fails in the first year of operation.

Exercise 11. *A coin is repeatedly tossed. What is the average number of tosses until we get tail followed by head (TH) for the first time? same question for getting two successive tails for the first time.*

Exercise 12.
1. Consider the c.r.v $X \sim \mathcal{U}([a,b])$, check that :

$$\mathbb{V}(X) \leq \frac{(b-a)^2}{4}$$

2. Let X be a r.v, prove the following implication:

$$P(a \leq X \leq b) = 1 \implies \mathbb{V}(X) \leq \frac{(b-a)^2}{4}$$

Exercise 13. *A person types a text of 2000 characters. He may enter an incorrect character for every 200 characters typed. Suppose that the errors of the entry are independent.*
1. What is the probability that the number of errors exceeds 50?
2. What is the size of the text (number of characters) so that the probability that the number of errors is less than 25 equals 0.9.

Exercise 14. *Consider $X_n = X + Y_n$, such that $\mathbb{E}(Y_n) = 1/n$, $\mathbb{V}(Y_n) = \sigma^2 n$, $\sigma > 0$ is a constant, show that $X_n \xrightarrow{p} X$.*

Exercise 15. *Consider X^* a s.r.v:*
1- Let

$$p_{X_n}(x) = \frac{1}{n}\mathbb{1}_{\{n^2\}} + (1 - \frac{1}{n})\mathbb{1}_{\{0\}}$$

Show that $X_n \xrightarrow{p} 0$
2- Let $Y \rightsquigarrow \mathcal{B}er(1/3)$ and

$$X_n = \frac{n}{n+1}Y + \frac{n}{n^2+1}(1 - Y)$$

a- Study the convergence of $X_n(\omega), \forall \omega \in \{P, F\}$.
b- Find $P\left(\{\omega_i \in \Omega : \lim_{n\to\infty} X_n(\omega_i) = 0\}\right)$.
3- Let

$$p_{X_n} = \frac{1}{2}\mathbb{1}_{\{\frac{-1}{n}\}} + \frac{1}{2}\mathbb{1}_{\{\frac{1}{n}\}}$$

Show that $X_n \xrightarrow{p.s.} 0$.

Exercise 16. *Consider the probability space $(\Omega = [0,1], P([a,b]) = b - a, \forall\ 0 < a < b < 1)$, We define X^* a s.r.v by $X_n = \mathbb{1}_{[0, \frac{n+1}{2n}[}$ $\forall n > 0$ and a r.v X by $X = \mathbb{1}_{[0,1/2[}$. Show that $X_n \xrightarrow{d} X$.*

2.14 Solutions

Solution 1.

1. The rank of X, $R_X = \{2, 3, 4\}$.
$\Omega = \{00111, 01011, 01101, 01110, 10011, 10101, 10110, 11001, 11010, 11100\}$.
0 represents a fake banknote and 1 represents a good banknote
It is a permutation with repetition of length 5 of the elements of B and M (B is repeated 3 times and M 2 times), so $|\Omega| = \dfrac{5!}{2!3!} = 10$

ω_i	00111	01011	01101	01110	10011	10101	10110	11001	11010	11100
x_i	2	3	4	4	3	4	4	4	4	3

X_i is the minimum number of required tests to find the fake banknotes.
$P(X = 2) = 1/10$, $P(X = 3) = 3/10$, $P(X = 4) = 6/10$
We can directly use the chain rule as follows:

- $P(X = 2) = P(M, M) = 2/5 \times 1/4 = 1/10$
- $P(X = 3) = P(B, M, M) + P(M, B, M) + P(B, B, B) = 3/5 \times 2/4 \times 1/3 + 2/5 \times 3/4 \times 1/3 + 3/5 \times 2/4 \times 1/3 = 3/10$
- $P(X = 4) = 1 - (P(X = 2) + P(X = 3)) = 6/10$

2. Calculate the expectation and variance of X :
$\mathbb{E}(X) = 2P(X = 2) + 3P(X = 3) + 4P(X = 4) = 3.5$
$\mathbb{V}(X) = (2 - 3.5)^2 P(X = 2) + (3 - 3.5)^2 P(X = 3) + (4 - 3.5)^2 P(X = 4) = 0.45$

Solution 2.

1. Find c

$$\int_0^1 c(1-x)dx = \left[c(x - x^2/2)\right]_0^1 = c/2 = 1$$

So $c = 2$

2. Let x_0 be the capacity of the dam.
The probability that the volume of precipitation exceeds the storage capacity:

$$P(X > x_0) = 1 - P(X) = 1 - F(x_0) = 1 - \int_0^x {}_0(2t - t^2)dt = x_0^2 - 2x_0 + 1$$

$$P(X > x_0) \leq 0.01 \implies x_0^2 - 2x_0 + 1 \leq 0.01 \implies x_0^2 - 2x_0 + 0.99 \leq 0$$

The solution of this inequality is $x = 0.9$, so the capacity of the dam in thousands of litres is 0.9

Solution 3.

Using a random variable X following the Poisson distribution of parameter $\lambda = 4.2$, we can calculate

$$P(X > 7) = 1 - P(X \leq 7) = 1 - \sum_{k=0}^{7} e^{-4.2} 4.2^k / k! = 0.064$$

Solution 4.

$$\begin{aligned}
f_Z(z) &= \int_{-\infty}^{+\infty} f_X(z-y) f_Y(y)dy \\
&= \int_a^b f_X(z-y)dy \\
&= \int_a^z dy = a + z \text{ } if \text{ } a \leq z \leq \frac{a+b}{2} \\
&= \int_{z-\frac{a+b}{2}}^b dy = a + bz \text{ } if \text{ } \frac{a+b}{2} < z \leq a + b \\
&= z\mathbb{1}_{[a, \frac{a+b}{2}]}(z) + (a + bz)\mathbb{1}_{[\frac{a+b}{2}, a+b]}
\end{aligned}$$

So $Z \rightsquigarrow \mathcal{T}riang(a, a+b, \frac{a+b}{2})$

Solution 5.

(X,Y)	*(1,1)*	*(1,0)*	*(0,1)*	*(0,0)*
$\lvert X-Y\rvert$	*0*	*1*	*1*	*0*
$p_{\lvert X-Y\rvert}$	*1/4*	*1/4*	*1/4*	*1/4*

So the rank of $|X-Y|$ *is:* $\{0,1\}$ *and* $|X-Y| \rightsquigarrow \mathcal{B}er(1/2)$.
2.

(X,Y)	*(1,1)*	*(1,0)*	*(0,1)*	*(0,0)*
$\lvert X-Y\rvert$	*0*	*1*	*1*	*0*
$p_{\lvert X-Y\rvert}$	p^2	$p(1-p)$	$p(1-p)$	$(1-p)^2$

So the rank of $|X-Y|$ *is:* $\{0,1\}$ *and* $|X-Y| \rightsquigarrow \mathcal{B}er(2p(1-p))$.

Solution 6.
We have $\{min(X_1, X_2, ..., X_n) > x\} = \{\cap_{i=1}^n (X_i > x)\}$

$$\overbrace{P(\cap_{i=1}^n (X_i > x)) = \prod_{i=1}^n P(X_i > x)}^{X_i \text{ are independent}} \underbrace{= \prod_{i=1}^n e^{-\lambda_i x}}_{X_i \rightsquigarrow \mathcal{E}xp(\lambda_i)} = e^{-(\sum_{i=1}^n \lambda_i)x}$$

So $min(X_1, X_2, ..., X_n) \rightsquigarrow \mathcal{E}xp(\sum_{i=1}^n \lambda_i))$

Solution 7.
This probability is easily calculated by conditioning on X_1*:*

$$\begin{aligned}
P(X_1 < X_2) &= \overbrace{\int_0^\infty P(X_1 < X_2 | X_1 = x)\lambda_1 e^{-\lambda_1 x} dx}^{Total\ law\ and\ X_1 \rightsquigarrow \mathcal{E}xp(\lambda_1)} \\
&= \overbrace{\int_0^\infty P(x < X_2)\lambda_1 e^{-\lambda_1 x} dx = \int_0^\infty e^{-\lambda_2 x}\lambda_1 e^{-\lambda_1 x} dx}^{X_2 \rightsquigarrow \mathcal{E}xp(\lambda_2)} \\
&= \int_0^\infty \lambda_1 e^{-(\lambda_1 + \lambda 2)x} dx \\
&= \frac{\lambda_1}{\lambda_1 + \lambda_2}
\end{aligned}$$

Solution 8.
1. Let $k = min(x, y)$

$$P(X = x, Y = y) = (\frac{8}{10})^{k-1} \times 1/10 \times (\frac{9}{10})^{|yx|} \times 1/10$$

2. Find
$P(X = 2|Y = 4) = \dfrac{P(X = 2, Y = 4)}{P(Y = 4)}$
$P(Y = y) = (\dfrac{9}{10})^{y-1} \times \dfrac{1}{10}$
$P(X = 2|Y = 4) = \dfrac{\frac{8}{10} \times \frac{1}{10} \times \frac{9}{10} \times \frac{1}{10}}{(\frac{9}{10})^3 \times \frac{1}{10}} = \dfrac{8}{81}.$

Solution 9.

1. $R_X = [0, l/2]$ *and* $R_\theta = [0, \pi/2]$.
$X \sim \mathcal{U}([0, l/2])$ *and* $\theta \sim \mathcal{U}([0, \pi/2])$
X *and* θ *are independent because the orientation of the needle does not depend on the position of its center.*

$$P_{X,\theta}(x, \alpha) = P(X = x, \theta = \alpha) = P(X = x)P(\theta = \alpha) = \frac{4}{l\pi}$$

2. The needle inclined by an angle θ *crosses the line if* X *is less than the length of the opposite side of the angle* θ *in the right triangle whose hypotenuse is* $h/2$ *(see figure 2.23).*

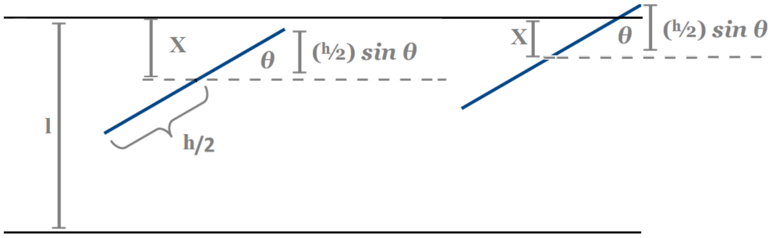


Figure 2.23: Buffon's experiment

$$\begin{aligned} P(X < h/2\sin\theta) &= \int_0^{\pi/2}\int_0^{h/2\sin\alpha} P_{X,\theta}(x,\alpha)dxd\alpha \\ &= \int_0^{\pi/2}\int_0^{h/2\sin\alpha} \frac{4}{l\pi}dxd\alpha = \frac{4}{l\pi}\int_0^{\pi/2}\int_0^{h/2\sin\alpha} dxd\alpha = \frac{4}{l\pi} \end{aligned}$$

Solution 10.

The device fails when the first of the two components fails. So the requested probability is:

$$\begin{aligned} P(min(X,Y) \leq 1) &= 1 - P(min(X,Y) > 1) \\ P(min(X,Y) > 1) &= P(X > 1, Y > 1) \\ &= \int_1^2\int_1^2 1/12(1+x+y)dxdy \\ &= 1/12\int_1^2 dx\int_1^2 (1+x+y)dxdy \\ &= 1/12\int_1^2 (1+x+1.5)dx = 1/3. \end{aligned}$$

So $P(min(x,y) \leq 1) = 2/3$.

Solution 11.

1. In the total expectation section, we have shown that the expectation of obtaining the first tail in successive flips of a coin is equal to $1/p$. *In this question, the average number of needed flips for the first TH is the sum of the average number of the first T and the average number of the first H, so it is equal to 1/p+1/q. For p=1/2, it will be equal to 4.*
Another solution: let X *be a r.v of the number of necessary flips to obtain the first TH,* Y *r.v of the result of the first flip and* Z *r.v of the number of necessary flips to obtain the first H.*

$\mathbb{E}(X) = \mathbb{E}(X|Y = H)\frac{1}{2} + \mathbb{E}(X|Y = T)\frac{1}{2}$
$\mathbb{E}(X) = (1 + \mathbb{E}(X))\frac{1}{2} + (2 \times \frac{1}{2} + (2 + \mathbb{E}(Z))\frac{1}{2})\frac{1}{2}$. $\mathbb{E}(Z) = 2$
$\mathbb{E}(X) = 4$

2. Let X *be the r.v of the average number of necessary tosses for the first TT, and* Y *the r.v of the result of the first toss. If the first toss gives H, we will have lost a toss and we will resume the experiment, otherwise, we will condition on the result of the second toss. If the latter is T, we will have obtained TT otherwise 2 tosses are lost and we resume the experiment from the beginning.*

$\mathbb{E}(X) = \mathbb{E}(X|Y = H)\frac{1}{2} + \mathbb{E}(X|Y = T)\frac{1}{2}$
$\mathbb{E}(X) = (1 + \mathbb{E}(X))\frac{1}{2} + (2 \times \frac{1}{2} + (2 + \mathbb{E}(X))\frac{1}{2})\frac{1}{2}$
$\mathbb{E}(X) = 6$

Solution 12.
1. Let $X \sim \mathcal{U}([a, b])$, then

$$\mathbb{V}(X) = \frac{(b-a)^2}{12} \leq \frac{(b-a)^2}{4}$$

2. We have $P(a \leq X \leq b) = 1$, so $P(0 \leq X - a \leq b - a) = 1$

$$\begin{aligned}\mathbb{V}(X) = \mathbb{V}(X-a) &= \mathbb{E}((X-a)^2) - \mathbb{E}^2(X-a)\\ &\leq (b-a)\mathbb{E}(X-a) - \mathbb{E}^2(X-a)\\ &\leq \frac{(b-a)^2}{4} - (\frac{(b-a)}{2} - (b-a)\mathbb{E}(X-a) + \mathbb{E}^2(X-a))\\ &\leq \frac{(b-a)^2}{4} - (\frac{(b-a)}{2} - \mathbb{E}(X-a))^2\\ &\leq \frac{(b-a)^2}{4}\end{aligned}$$

Solution 13.
1. Let $X_i \rightsquigarrow \mathcal{B}er(p = 1/50)$ for $i = 1, \cdots n$ ($n = 2000$), equal to 1 if the character i is typed wrong and 0 otherwise. $S_n = X_1 + X_2 + \cdots + X_n$ is the number of wrong characters in the message.
$\mathbb{E}(X_i) = p = 0.02$ and $\mathbb{V}(X_i) = p(1-p) = 0.0196$.
By applying the central limit theorem:

$$\begin{aligned}P(S_n > 50) &= P(\frac{S_n - np}{\sqrt{n}\sigma} > \frac{50 - np}{\sqrt{n}\sigma})\\ &= P(\frac{S_n - np}{\sqrt{n}\sigma} > \frac{50 - 40}{\sqrt{39.2}})\\ &= 1 - \phi(1.59) = 0.0559.\end{aligned}$$

2. Find :

$$\begin{aligned}P(S_n < 65) &= 0.95\\ P(S_n < 65) &= P(\frac{S_n - np}{\sqrt{n}\sigma} < \frac{65 - np}{\sqrt{n}\sigma})\\ &= \phi(\frac{65 - 0.02n}{\sqrt{n}0.14}) = 0.9 = \phi(1.65)\\ \frac{65 - 0.02n}{\sqrt{n}0.14} &= 1.65\\ n &= 2655.\end{aligned}$$

Solution 14.

$$P\big(|X_n - X| \geq \epsilon\big) = P\big(|Y_n| \geq \epsilon\big) \leq \underbrace{P\left(|Y_n - \mathbb{E}(Y_n)| + |\mathbb{E}(Y_n)| \geq \epsilon\right)}_{\textit{inequality triangular}}$$

$$= \underbrace{P\left(|Y_n - \mathbb{E}(Y_n)| \geq \epsilon - \frac{1}{n}\right) \leq \frac{\mathbb{V}(Y_n)}{\left(\epsilon - \frac{1}{n}\right)^2}}_{\textit{Chebyshev inequality}} = \frac{\sigma^2}{n\left(\epsilon - \frac{1}{n}\right)^2}$$

So $\lim_{n\to\infty} P\big(|X_n - X| \geq \epsilon\big) = 0$, we can conclude that $X_n \xrightarrow{p} X$

Solution 15.

1- Let $\epsilon > 0$,

$$P\big(|X_n - X| \geq \epsilon\big) = P\big(|X_n| \geq \epsilon\big) = 1 - P\big(|X_n| < \epsilon\big) < 1 - P\big(|X_n| \leq 0\big) = \frac{1}{n}$$

So $\lim\limits_{n\to\infty} P\big(|X_n - X| \geq \epsilon\big) = 0$, *we can conclude that* $X_n \xrightarrow{p} X$

2- We separate the cases according to ω*:*

a1- For $\omega = H$, *we have* $Y = 1$ *and* $X_n(\omega) = \frac{n}{n+1}$ *therefore* $\lim\limits_{n\to\infty} X_n = 1$

a2- For $\omega = T$, *we have* $Y = 0$ *and* $X_n(\omega) = \frac{n}{n^2+1}$ *therefore* $\lim\limits_{n\to\infty} X_n = 0$

b- $P\left(\{\omega_i \in \Omega : \lim_{n\to\infty} X_n(\omega_i) = 0\}\right) = P(T) = 2/3$

3- Let Ω *be the sample space associated with* X, $\forall \omega \in \omega$, *we have the following three cases:*

i1- $X_n(\omega) = \frac{1}{n}$, *with probability* $1/2$, *so* $\lim\limits_{n\to\infty} X_n(\omega) = 0$

i2- $X_n(\omega) = \frac{-1}{n}$, *with probability* $1/2$, *so* $\lim\limits_{n\to\infty} X_n(\omega) = 0$

i3- $X_n(\omega) \in \mathbb{R}/\{1/n, -1/n\}$, *with probability* 0

Then from i1 and i2 we have: $P\left(\left\{\omega \in \Omega : \lim\limits_{n\to\infty} X_n(\omega) = X(\omega)\right\}\right) = 1$

so $X_n \xrightarrow{ps} 0$

Solution 16.

It suffices to show that $X_n \xrightarrow{p.s} X$ *and then use* $\xrightarrow{p.s} >> \xrightarrow{d}$ *to conclude that* $X_n \xrightarrow{d} X$

We have two cases :

i- $\omega \in [0, \frac{1}{2}[$

$\omega \in [0, \frac{n+1}{2n}]$ *therefore* $X_n(\omega) = X(\omega) = 1$

then $[0, \frac{1}{2}[\subset A$

$ii - \omega \in]\frac{1}{2}, 1[$

$\exists n_\omega \in \mathbb{N}$, *such that* $\frac{1}{n_\omega} < 2\omega - 1$

we have $\frac{n_\omega + 1}{2n_\omega} < \omega$

therefore $X_{n_\omega}(\omega) = X_(\omega) = 0$ *then* $]\frac{1}{2}, 1[\subset A$

From i and ii, $\Omega = A \cup \{\frac{1}{2}\}$ *and* $P(\frac{1}{2}) = 0$ *then* $P(A) = P(\Omega) = 1$

So $X_n \xrightarrow{p.s.} X$*and resulting in* $X_n \xrightarrow{d} X$

Part II

Stochastic Processes

Chapter 3

Stochastic Processes

3.1 Introduction

For many complex phenomena, the systems that govern their evolution are dynamic and random (given their complexity). This random aspect is not limited only to a given moment but it follows this dynamic by becoming a function of time. In order to study this class of phenomena, we must mathematically model its different components and in particular the dynamic random part. Stochastic processes are one of the mathematical tools to carry out this modeling. They describe the evolution of the state of a random entity of a dynamic system over time by determining its probability laws and its statistical characteristics. Examples of dynamic phenomena that can be modeled using random processes are the evolution of the price of a listed company stock or the daily temperature during a year.

> **Definition 1.**
> *A Stochastic process is a family of random variables X_t indexed by t a parameter with values in a set T and defined on a given probability space $(\Omega, \mathcal{F}, P)$, and its rank is $R_{X_t} = S$.*
>
> $$X_{(T,S)} = \{X_t \text{ is r.v on } (\Omega, \mathcal{F}, P) \text{ in } S : \forall t \in T\}$$
>
> *T is called the set of parameters/indices, S is called the state space whose elements s are the states of this process.*

The parameter t often represents time, but it can represent other quantities such as a position in the space.

Notation.
- In the rest of the document, we abbreviate the stochastic process by ***SP***.
- If S is well known, then $X_{(T,S)}$ will simply be noted $X_T = (X_t)_{t \in T}$.

$X(t, \omega), t \in T, \omega \in \Omega$ is a two-variable function ($T \times \Omega \mapsto S$), if we set :

- t, we get X_t a r.v $X_t : \Omega \mapsto S$
- ω : $X_t(\omega)$ a function of t called a *realization* of the SP (a *trajectory*) associated with ω.

Depending on the nature of the elements of the set T and of S (the *countability property*), we can classify the set of SPs into several types. Table 3.1 gives the families of SPs according to their nature:

Figure 3.1 in its part E shows the realizations corresponding to the eventualities ω_1, ω_2 and ω_3 and the r.v associated with instant t_1. The distribution law of X_t (P_{X_t}) is a function of t. The sub-figures A, B, C and D give the graphic representation of the classes of SPs.

T | X_t	Discrete	Continuous
Discrete	*discrete values PS and discrete time: B*	*continuous values PS and discrete time: C*
Continuous	*discrete values PS and continuous time: A*	*continuous values PS and continuous time: D*

Table 3.1: Types of stochastic processes

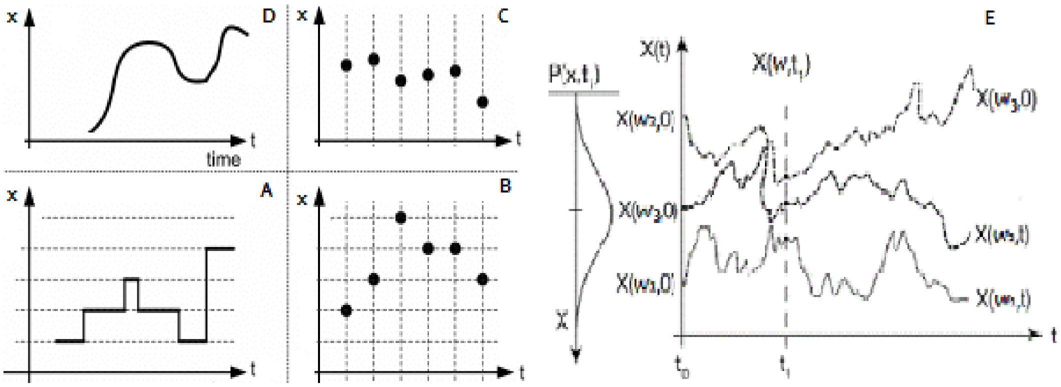


Figure 3.1: (A,B,C,D): types of SP. E: realisations of SP

Example 1. *Several phenomena are characterized by their dynamic and stochastic nature and may be subject of SP modeling. As examples :*

1. *The fortune of a player after playing n heads or tails (winning 1 dinar for heads and losing 1 dinar for tails). SP $X_{(\mathbb{N},\mathbb{Z})}$.*

2. *The temperature on the n^{th} day of the year. SP $X_{(\mathbb{N},\mathbb{R})}$.*

3. *The number of requests arriving to a server at a given time t. SP $X_{(\mathbb{R},\mathbb{N})}$.*

4. *The wind speed at a given time t. SP $X_{(\mathbb{R},\mathbb{R})}$.*

Mathematically, we can integrate the random aspect into any equation that describes a dynamic of a system. To have a stochastic model we introduce a random disturbing factor (one or more r.vs). As an example of this category :

1. *if $Y \sim \mathcal{B}er(p)$ and $R_Y = \{+1, -1\}$ and we define $X_n = X_{n-1} + Y$ then $X_{(\mathbb{N},\mathbb{Z})}$ is a SP (called random walk).*

2. *if $Y \sim \mathcal{N}(0,1)$ and we define $X_n = aX_{n-1} + Y$ then$X_{(\mathbb{N},\mathbb{R})}$ is a SP.*

Let's code!

The codes in this section will use some helpers that implement recurring features.

```
# utils.py (Continuation)

# generate_SP: generates a stochastic process X* one sample Path for T[0..999]
def generate_SP(rvY, N=1000):
    spX, X_n = [0], 0   #stochastic process, random variable X_n
    for i in range(N):
        X_n = X_n + rvY()     #current r.v X_n+1
        spX.append(X_n)     #newt step in SP X*
    return spX

# plotSP: plots the SP's sample paths
def plotSP(generate_SP1, nbSP=5, N=100) :
    for i in range(nbSP):
        spX = generate_SP1(); #ith sample path
        plt.plot(spX[:N])
```

Let's code!

The following code shows how to generate a simple random process modeling the sum of a Bernoulli test sequence (modeling the fortune of a player tossing a coin by winning +1 to the head and losing 1 to the tail).

```
#Code301.py

import numpy as np
import random
import sys;sys.path.append('../lib')
from utils import generate_SP,plotSP

fY=lambda : 1 if random.randint(0, 1) else -1
def generate_BerSum1():
    return generate_SP(fY)

# generate_BerSum2: equavalent version of generate_BerSum1 using umpy.cumsum
# (cummulative sum of the r.v values)
def generate_BerSum2():
    return np.cumsum([fY() for i in range(1000)])

plotSP(generate_BerSum2)
```

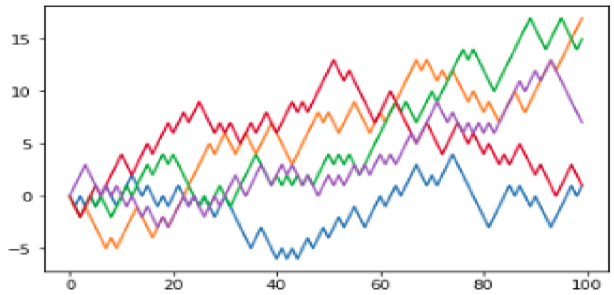


Figure 3.2: Multiple trajectories of the SP modeling the sum of a sequence of Bernoulli tests (-1/1)

3.1.1 CDF functions and statistics of SP

It is not enough to define a SP by the distribution of values X_t or their CDFs (P_{X_t} or F_{X_t}) but it is necessary to integrate the interaction factor between its components by defining the joint functions of any finite subset of its r.vs.

A SP is specified by the joint distribution of a finite set of its r.vs. To describe the relationship between the values of the SP at different times, we define the finite-dimensional distribution function.

Definition 2.
A SP is defined by the following finite-dimensional joint CDF :

$$\forall k \in \mathbb{N}, \{t_1, \cdots, t_k\} \subset T, F_{t_1,t_2,\cdots,t_k}(x_1, x_2, \cdots, x_k) = P(X_{t_1} \leq x_1, \cdots, X_{t_k} \leq x_k)$$

In the context of SPs, the statistical quantities of r.vs ($\mathbb{E}, \mathbb{V}, \cdots$) are functions parameterized by t. Subsequently, they will be the necessary tool to characterize each studied SP (it is not always easy to determine the finite-dimensional joint CDF).

Definition 3.
Let t and s be two different moments in T and $k \in \mathbb{N}^*$:
- ***Function of moments of order k** : the function which gives the moment of order k of each r.v at each moment t.*
- ***Average function** (moment of order 1 mathematical expectation): $m(t) = \mathbb{E}(X_t)$*
- ***Variance function** (moment of order 2): $v(t) = \mathbb{V}(X_t)$*

> - ***Autocovariance function*** : $K(t,s) = Cov(X_t, X_s)$
> - ***Autocorrelation function*** : $\phi(t,s) = K(t,s)/\sqrt{v(t)v(s)}$.

Example 2. *Consider a SP $X^*_{(N,N)}$ represented by the sum of the sequence of the r.vs that follow Bernoulli's distribution, we know that this sum is a Binomial r.v (Binomial SP) so we know its statistic characteristics:*

- *Consider $Y \sim \mathcal{B}er(p)$ and $R_Y = \{1, 0\}$, we define $X_n = X_{n-1} + Y$ so $X_n \sim \mathcal{B}in(n,p)$.*
- *$m(n) = \mathbb{E}(X_n) = np$ and $v(n) = \mathbb{V}(X_n) = np(1-p)$ are function of the parameter n*

Let's code!

The following code shows the mean function $m(t)$ and variance $v(t)$ associated to Binomial SP.

```
#Code302.py

import numpy as np
import random
import matplotlib.pyplot as plt
import sys;sys.path.append('../lib')
from utils import generate_SP,plotSP

# plotter: plots SP realisations and E and V functions
def plotter(rgN, gen_SP, fE, fV):
    n = rgN
    plt.step(n, fE(n)          , color='red')
    plt.step(n, fE(n) + fV(n)/2, color='black')
    plt.step(n, fE(n) - fV(n)/2, color='black')
    plotSP(gen_SP, 5, N)

N, p = 30, 0.5;  rgN = np.arange(0,N)
fYBer = lambda : random.randint(0, 1)
fnEBer = lambda n: n*p
fnVBer = lambda n: n*p*(1-p)
def generate_SP_Ber(): return generate_SP(fYBer)

plotter(rgN, generate_SP_Ber, fnEBer, fnVBer)
```

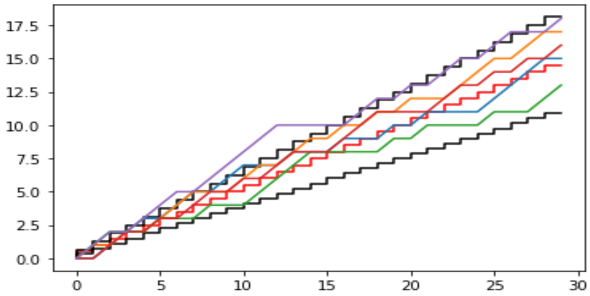


Figure 3.3: SP that models the sum of a sequence of Bernoulli (0/1) trials

3.1.2 Classes of stochastic processes

The set of r.vs forming the SP presents certain interesting aspects which allow us to characterize the SP (its tendency, its components, the relationship between its elements, ...). This characterization will be useful to simplify the analysis of the behavior of the SP.

> **Definition 4.**
> *Consider a SP $(X_t)_{t\in T}$, the SP is said :*
>
> 1. ***Independent and identically distributed iid*** *if the generated r.vs have the same law and are independent.*
>
> $$(X_t)_{t\in T} \text{ is iid } \iff \forall t,s \in T, t \neq s \implies X_t \perp X_s \text{ et } P_{X_t} = P_{X_s}$$
>
> 2. ***Stationary in the strict sense s.s.s*** *: when the joint distribution of a set of its*

r.v, (for example. from the first to the fourth) is the same as the joint distribution of another disjoint set of the same SP, (example from the sixth to the ninth). Its cdf depends only on the temporal difference (the difference between times).

$$(X_t)_{t\in T} \text{ is Stationary } \iff \forall s>0, i\neq j, X_{t_i+s}, X_{t_j+s} \text{ has the same joint CDF as } X_{t_i}, X_{t_j}$$

or

$$\forall s>0, \forall k\in\mathbb{N}, \{t_1,\cdots,t_k\}\subset T, F_{t_1+s,\cdots,t_k+s}(x_1,\cdots,x_k) = F_{t_1,\cdots,t_k}(x_1,\cdots,x_k)$$

3. ***Stationary in the weak sense s.w.s** : The expectation and the variance are constant and finite, and the function of the covariance does not depend on the instants t, but the relative difference between these instants (SP whose statistical characteristics do not change as a function of time).*

$$\exists c>0, c'>0, \quad \mathbb{E}(X_t)=c\leq\infty, \mathbb{V}(X_t)=c'\leq\infty \text{ et } \forall t,s>0, K(t,s)=f(t-s)$$

4. ***With independent increments**: we call increment a r.v which gives the difference between the two values taken by the SP X_t at instants s and t, $\Delta X_{(t,s)} = X_t - X_s$ such that $s<t$. (if $s=0$ then the increment will be noted ΔX_t).*
We say that X_t with independent increments if the increments r.vs for non-overlapping parameters are independent.

$$\forall 0<t_1<t_2<s_1<s_2, \quad \Delta X_{(t_1,t_2)} \perp \Delta X_{(s_1,s_2)}$$

5. ***Of stationary increments** if $\forall t>s>0, c>0, \Delta X_{(t+c,s+c)}$ follows the same distribution as $\Delta X_{(t,s)}$ (the distribution is invariant by translation over time).*

6. ***Memoryless (Markovian)** if its future evolution does not depend on the past, but only on the present. In case, of discret space SP, we have:*

$$\forall t_1<\cdots<t_{n+1}, \forall n, \quad P(X_{t_{n+1}}=x_{n+1}|X_{t_1}=x_1,\cdots,X_{t_n}=x_n) = P(X_{t_{n+1}}=x_{n+1}|X_{t_n}=x_n)$$

Example 3.
*1- The **white noise** $(X_t)_{t\geq 0}$ is an i.i.d SP (a sequence of i.i.d r.vs) which has a constant mean and a finite variance. This SP is stationary.*
$\mathbb{E}(X_t)=\mu \qquad \mathbb{V}(X_t)=\sigma^2$
For $t=s$: $K(s,t)=Cov(X_t,X_s)=cov(X_t,X_t)=\mathbb{V}(X_t)=\sigma^2$
For $t\neq s$: $K(t,s)=Cov(X_t,X_s)=0$

$\phi(s,t)=K(s,t)/\sqrt{v(s)v(t)}$
For $t=s$: $\phi(s,t)=1$
For $t\neq s$: $\phi(s,t)=0$

*2- The **random walk** $(X_t)_{t\geq 0}$ is a SP, for $t>1$, $X_t=X_{t-1}+\epsilon_t$ Such that ϵ_t is white noise with zero mean and variance σ^2.*
$\mathbb{E}(X_t)=\mathbb{E}(X_{t-1}+\epsilon_t)=\mathbb{E}(X_{t-1})+\mathbb{E}(\epsilon_t)=0$
$\mathbb{V}(X_t)=cov(X_t,X_t)=\mathbb{V}(X_{t-1}+\epsilon_t)=\mathbb{V}(X_{t-1})+\mathbb{V}(\epsilon_t)=t\sigma^2$
For $t=s$:

$$K(s,t)=Cov(X_t,X_s)=cov(X_t,X_t)=\mathbb{V}(X_t)=t\sigma^2$$

For $s < t$:

$$\begin{aligned}
K(t,s) = Cov(X_t, X_s) &= Cov(X_s + \sum_{h=s+1}^{t} \epsilon_h, X_s) \\
&= Cov(X_s, X_s) + Cov(\sum_{h=s+1}^{t} \epsilon_h, X_s) \\
&= s\sigma^2 + \sum_{h=s+1}^{t} Cov(\epsilon_h, X_s) = s\sigma^2
\end{aligned}$$

For $t = s$: $\phi(s,t) = 1$
For $t \neq s$:

$$\phi(t,s) = K(t,s)/\sqrt{v(s)v(t)} = \frac{s\sigma^2}{\sqrt{s\sigma^2 t\sigma^2}} = \sqrt{\frac{s}{t}}$$

3- Consider a SP with $X_n = X_{n-1} + Y$ such that $Y \sim \mathcal{B}er(0.5)$ (1,-1) then $X_n \sim \mathcal{B}in(n,p)$. $m(n) = \mathbb{E}(X_n) = 0$ and $v(n) = 1$ are constants and as the covariance function depends on the deviation not on the relative positions, then this SP is stationary.

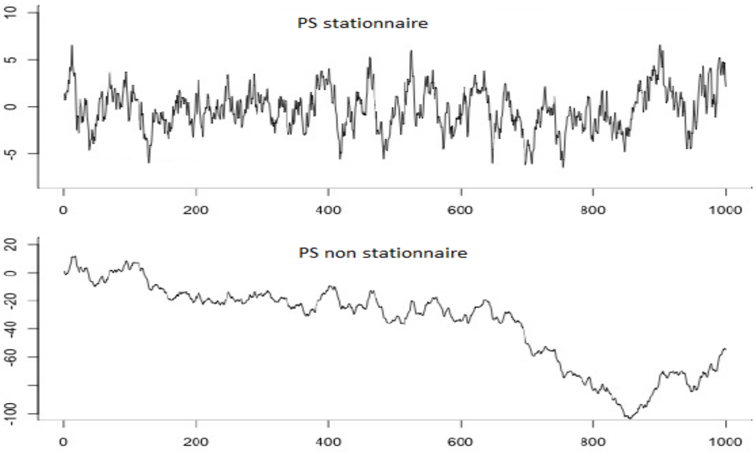


Figure 3.4: stationary SP vs non-stationary SP

3.2 Counting process

To model the change of state of a system whose dynamics are governed by the appearance of a particular type of random event over time, we are sometimes interested in the number of events during the life of this system which evolves in amplitude jumps of one or more units.

Definition 5.
The stochastic process $(N_t)_{t \in T}$ is a counting process (enumeration) if it has non-negative integer values, increasing whose values represent the total number of events that have occurred up to time t. N_t must meet the following conditions:
*1- **Non-negative integer valued** : $R_{N_t} \subset \mathbb{N}$*
*2- **Increasing** : $s < t$ then $N_s \leq N_t$*
*3- **Jump** (increment) : For $s < t$, $\Delta N_{(t,s)} = N_t - N_s$ is equal to the number of events that have occurred in the time interval $[s,t]$*

The last event associated with instant t will be noted $E_{N_t} = E_n$, such that $n = N_t$ called n^{th} arrival.

Example 4.
1- Calls to a telephone center, E_t = *'receiving a call at time* t*'*
2- Radioactive particle emissions, E_t =*'a radioactive particle emission'*
3- Customers who have arrived at a counter until a given time. E_t =*'the arrival of a customer at a ticket counter'.*
4- Consider a series of Bernoulli tests. E_t =*'Bernoulli's experiment that gives success'.* $Y \sim \mathcal{B}er(p)$ *and* $R_Y = \{1, 0\}$ *and the SP defined by* $X_t = X_{t-1} + Y$*.* X_t *represents the number of Bernoulli's experiments that gave success after a certain number of repetitions.*

a- $R_{X_t} \subset \mathbb{N}$

b- $\forall s, t > 0, s \geq t, X_s \geq X_t$ *increasing.*

c- $\Delta X_{(t,s)}$ *equals the number of events* E_t *corresponding to an experiment giving success in* $[s, t]$*. So* X_t *is a counting SP.*

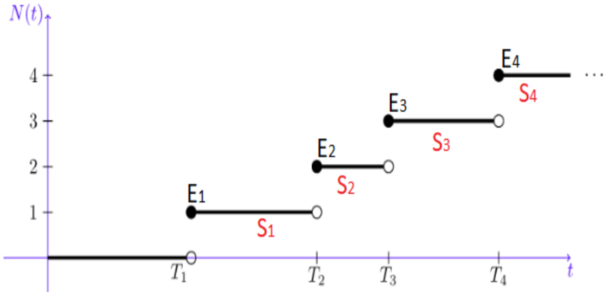


Figure 3.5: Counting process

The set of r.vs that appear naturally with the counting process are the *sojourn time (inter-arrival)* S_n between event E_n and E_{n+1} for $n \geq 0$. $(S_n)_{n>0}$ is the SP of the inter-arrival time of the events. The quantity N_t can be expressed as a function of another r.v T_n the *arrival time of the* n^{th} *event* E_n. $(T_n)_{n>0}$ is the SP associated with the variation of this quantity.
Thereafter we agree on the following terminology :
- N_t : number of arrivals (events) until time t.
- T_n : arrival time of the n^{th} event E_n.
- S_n : sojourn time of the process in state n. (the inter-arrival time between E_n and E_{n+1}).

The relationships between these quantities are :

$$N_t = \sum_{k=0}^{\infty} \mathbb{1}_{\{T_k < t\}}$$

$$\{T_n < t\} = \{N_t \geq n\} \text{ and } \{T_n \geq t\} = \{N_t \leq n\}$$

$$T_n = S_0 + S_1 + \cdots S_n = \sum_{k=0}^{n} S_k$$

3.3 Poisson process

One of the most important counting processes is the Poisson process.

Definition 6.
A counting process $(N_t)_{t \geq 0}$ *is a Poisson process of rate* $\lambda > 0$ *if :*

1. $N_0 = 0$

2. N_t *has independent and stationary increments.*

> *3. $P(N_h = 1) = \lambda h + o(h)$ and $P(N_h > 1) = o(h)$ when $h \to 0^+$*

A function $f(h)$ is said $o(h)$ if $\lim\limits_{h\to 0} \frac{f(h)}{h} = 0$.

The third condition is equivalent to the fact that the probability of observing more than one event in an interval δt of length h tends toward 0 as h tends toward 0.
Since N_t has stationary increments, then over the interval $[s, s+h]$ of length h, the increment is equal to its value at instant h.

$$\Delta N_{(s,s+h)} = \Delta N_{(0,h)} = N_h$$

λ represents the average number of events occurring in a unit of time.

> **Proposition 1.**
> *If $(N_t)_{t\geq 0}$ is a Poisson process of rate $\lambda > 0$ then :*
> *1- $N_t \sim Pois(\lambda t)$ i.e $p_{N_t}(n) = e^{-\lambda t}\frac{(\lambda t)^n}{n!}$*
> *2- $T_n \sim Gamma(\lambda, n)$ i.e $P(T_n \leq t) = \lambda e^{-\lambda t}\frac{(\lambda t)^{n-1}}{(n-1)!}$*
> *3- $S_n \sim Exp(\lambda)$ i.e $P(S_n \leq t) = 1 - e^{-\lambda t}$.*

3.3.1 Law of N_t

The Poisson distribution is used in the modeling of rare events. It is the limit of the Binomial distribution of parameters n and p when n (the number of repetitions of a Bernoulli experiment) is very large and p (the probability of success) is very small , the total number of successes follows Poisson's law.
We have: $\lambda = np$ and $X \rightsquigarrow \mathcal{B}in(n,p)$.

$$\lim_{n\to\infty, p\to 0} P(X = k) = e^{-\lambda}\frac{\lambda^k}{k!} \qquad \text{for } k = 0, 1, 2, ...$$

Let $p = \lambda/n$

$$\begin{aligned} P(X = k) &= C_n^k(\frac{\lambda}{n})^k(1 - \frac{\lambda}{n})^{(n-k)} \\ &= \frac{\lambda^k}{k!}(1 - \frac{\lambda}{n})^n(\frac{n!}{n^k(n-k)!})(1 - \frac{\lambda}{n})^{-k} \\ \frac{n!}{n^k(n-k)!} &= \frac{n(n-1)...(n-k+1)}{n^k} = (1 - \frac{1}{n})...(1 - \frac{k-1}{n}) \end{aligned}$$

This expression tends towards 1 as n tends to ∞ and the term $(1 - \frac{\lambda}{n})^{-k}$ too. We have also:

$$\lim_{n\to\infty}(1 + \frac{b}{n})^n = e^b \ \forall\ b \in R, \text{ so } \lim_{n\to\infty}(1 - \frac{\lambda}{n})^n = e^{-\lambda}$$

3.3.2 Law of S_n

We consider instant T_n.
S_n= waiting time until the next occurrence.

$$\begin{aligned} &P(S_n > t) = P(N_{T_n+t} - N_{T_n} = 0) \\ &P(S_n > t) = P(N_t = 0) \quad \text{(hypothesis of temporal independence)} \\ &P(S_n > t) = e^{-\lambda t} \\ &P(S_n \leq t) = 1 - e^{-\lambda t} \end{aligned}$$

So:

$$S_n \rightsquigarrow \mathcal{E}xp(\lambda)$$

3.3.3 Law of T_n

Let T_n be the moment on which the n^{th} event occurs. $T_n - T_{n-1} \rightsquigarrow \mathcal{E}xp(\lambda)$ n>0, $T_0 = 0$.
So T_n is the sum of n exponential parameter variables λ and it is the Gamma law noted $\Gamma(n, \lambda)$.
The density $f_{T_n}(t)$ of the r.v T_n is written:

$$f_{T_n}(t) = \lambda e^{-\lambda t} \frac{(\lambda t)^{n-1}}{(n-1)!} \quad t \geq 0$$

$$\mathbb{E}(T_n) = \frac{n}{\lambda} \qquad \mathbb{V}(T_n) = \frac{n}{\lambda^2}$$

Let's code!

```
#Code303.py

import numpy as np
import matplotlib.pyplot as plt

# poisson_Proc: generates sampling for Poisson SP for different parameters
def poisson_Proc(N,lambdas):
    X_T = [np.random.poisson(lam, size=N) for lam in lambdas]
    return [np.cumsum(X) for X in X_T]

def plot_Poisson(N,S,lambdas):
    X = np.linspace(0, N, N)
    G = [plt.step(X, S[i], label="Lambda = %d"%lambdas[i])[0] for i in range(len(lambdas))]
    plt.legend(handles=G, loc=2)
    plt.title("Poisson Process", fontdict={’fontname’: ’Times New Roman’, ’fontsize’: 21}, y=1.03)
    plt.ylim(0);  plt.xlim(0)
    plt.show()

N , lambdas = 20 , [4, 7, 13]
S = poisson_Proc(N,lambdas)
plot_Poisson(N,S,lambdas)
```

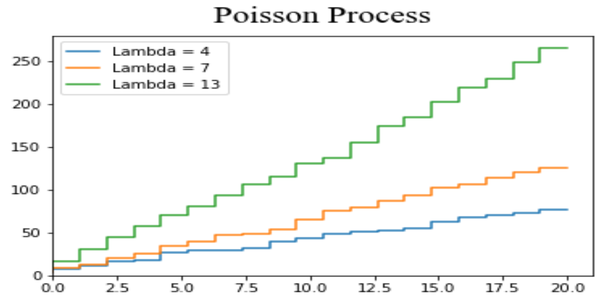


Figure 3.6: Poisson process

Example 5. *A volcano in a given city is estimated to erupt, on average, once every 400 years. What is the probability that a newly born person living in this city will not witness an eruption during his first 20 years?*
Solution
Poisson's law can be used to model this event. $\lambda = 1/400$ *(one eruption in 400 years).*
$N_t \rightsquigarrow Pois(\lambda t)$ *and the time between two successive eruptions* $S_n \rightsquigarrow \mathcal{E}xp(\lambda)$
The event that a newly born person does not witness an eruption during his first 20 years is equivalent to $\{S_n > 20\}$.

$$P(S_n > 20) = 1 - P(S_n \leq 20) = e^{-20\lambda} = e^{-\frac{1}{400}\times 20} = 0.9512$$

Another solution: during the first 20 years of a person's life, no eruption will occur $\equiv \{N_{20} = 0\}$

$$P(N_{20} = 0) = e^{-20\lambda} \frac{(20\lambda)^0}{0!} = e^{-20\lambda} = 0.9512$$

Example 6. *People immigrate to a given city following a Poisson law rate* $\lambda = 3$ *persons/day.*
1- What is the average time until the 10^{th} *person arrives?*

2- What is the probability that the time between the arrival of the 10^{th} *and the* 11^{th} *person exceeds* 1 *day?*

Solution

1- $\mathbb{E}(T_{10}) = 10/\lambda = 3.33$ *days*

2- $P(S_{10} > 1) = e^{-3\times 1} \approx 0.049$

Let's code!

```
#Code304.py

import numpy as np
from numpy.random import exponential

# Example3.6
N, lamda, dixieme, dixOnze =10000, 3, [], []
for j in range(N):
    T_n = exponential(1/lamda)
    for i in range(12):
        S_n = exponential(1/lamda)
        T_n += S_n
        if(i==9): dixOnze.append(S_n)
        if(i==8): dixieme.append(T_n)

# question1
avg = np.array(dixieme).mean()
print("Average time until 10th arrival:",np.round(avg,3),"\n")

# question2
plusdunjour=[1 if i>1 else 0 for i in dixOnze]
proba = np.array(plusdunjour).sum()/len(plusdunjour)
print("Probability more than a day:",np.round(proba,3))

#______________________________   Output  ______________________________________
# Average time until 10th arrival: 3.325
# Probability more than a day: 0.053
```

3.3.4 Sum and composition of Poisson processes

Proposition 2.
Let X *and* Y *be two independent r.vs following the Poisson distribution with parameters* λ *and* μ *respectively. The r.v* $X+Y$ *follows the Poisson distribution of parameter* $\lambda+\mu$*:*

$$P(X=k) = e^{-\lambda}\frac{\lambda^k}{k!} \text{ and } P(Y=k) = e^{-\mu}\frac{\mu^k}{k!} \implies P(X+Y=k) = e^{-(\lambda+\mu)}\frac{(\lambda+\mu)^k}{k!}$$

$$X \sim \mathcal{P}ois(\lambda) \text{ and } Y \sim \mathcal{P}ois(\mu) \implies X+Y \sim \mathcal{P}ois(\lambda+\mu)$$

Demonstration.

$$\begin{aligned} P(X+Y=k) &= \sum_{j=0}^{k} P(X=j)P(Y=k-j) \\ &= \sum_{j=0}^{k} e^{-\lambda}\frac{\lambda^j}{j!}e^{-\mu}\frac{\mu^{k-j}}{(k-j)!} \\ &= \frac{e^{\lambda+\mu}}{k!}\sum_{j=0}^{k} C_k^j \lambda^j \mu^{k-j} \\ &= e^{\lambda+\mu}\frac{(\lambda+\mu)^k}{k!} \quad \text{for } k = 0, 1, \dots \end{aligned}$$

Newton's Binome: $(a+b)^k = \sum_{j=0}^{k} C_k^j a^j b^{k-j}$ is used in the last step of the demonstration.

Proposition 3.
Let $(N_t)_{t\geq 0}$ *be a Poisson process of rate* λ*, the events that occur are of two types: type 1*

arrive with probability p and type 2 with probability $1-p$. Each event is independent of all the others.
Let $N_t^{(1)}$ and $N_t^{(2)}$ be the number of events of type1 and type2, respectively, occurring between 0 and t.

$$N_t = N_t^{(1)} + N_t^{(2)}$$

$(N_t^{(1)})_{t\geq 0}$ and $(N_t^{(2)})_{t\geq 0}$ are Poisson processes having respectively rate λp and $\lambda(1-p)$ and are independents.

Proof.

$$\begin{aligned}
P(N_t^{(1)} = j, N_t^{(2)} = k) &= P(N_t^{(1)} = j, N_t^{(2)} = k, N_t = j+k) \\
&= P(N_t^{(1)} = j, N_t^{(2)} = k | N_t = j+k) P(N_t = j+k) \\
&= C_{j+k}^{k} p^j (1-p)^k e^{-\lambda} \frac{\lambda^{j+k}}{(j+k)!} \\
&= e^{-\lambda p} \frac{(\lambda p)^j}{j!} \times e^{-\lambda(1-p)} \frac{(\lambda(1-p))^k}{k!} \text{ for } j,k \geq 0
\end{aligned}$$

$$\begin{aligned}
P(N_t^{(1)} = j) &= \sum_{k=0}^{\infty} P(N_t^{(1)} = j, N_t^{(2)} = k) = e^{-\lambda p} \frac{(\lambda p)^j}{j!} \\
P(N_t^{(2)} = k) &= \sum_{j=0}^{\infty} P(N_t^{(1)} = j, N_t^{(2)} = k) = e^{-\lambda(1-p)} \frac{(\lambda(1-p))^k}{k!}
\end{aligned}$$

Which proves that $(N_t^{(1)})_{t\geq 0}$ and $(N_t^{(2)})_{t\geq 0}$ are Poisson processes having respectively rate λp and $\lambda(1-p)$ and are independents.

3.4 Exercises

Exercise 1. *Suppose that the time a person spends in the bank is exponentially distributed with an average of 10 minutes.*
1- What is the probability that a customer spends more than 15 minutes in the bank?
2- What is the probability that the customer will spend more than 15 minutes given that he has already spent 10 minutes in the bank?

Exercise 2. *In a quay, on average, five ships arrive per day. Let us model the arrivals of ships by a Poisson process. Let X be the random variable which measures the time between midnight until the arrival of the first ship.*
1- What is the expectation of X?
2- What is the probability that more than two ships will arrive between midnight and 6 a.m.?

Exercise 3. *A server receives requests following a Poisson process with a rate of 10 requests per minute. There is a probability $1/12$ that the query is erroneous.*
What is the probability that the server does not receive any bad request for 5 minutes?

Exercise 4. *[21] A scientific theory assumes that division defects of a cell occur following a Poisson process of rate 2.5 per year, and that an individual dies when 196 such defects occur. Find:*
(a) The average lifespan of a person,
(b) The variance of a person's lifespan.
(c) What is the probability that a person will die before the age of 67.2,

Exercise 5. *[10] In a football match between two teams A and B, goals are scored following Poisson process with rate $1/30$ per minute. The full time of the match is 90 minutes. Each*

scored goal comes from team A with probability $12/25$ and team B with probability $13/25$.
1. What is the probability that the total number of goals will be 3 or more during the match.
2. What is the probability of having exactly two goals in the first half and 1 goal in the second half.

3.5 Solutions

Solution 1.
1/ $X \rightsquigarrow Exp(\lambda)$, $\lambda = 1/10$ $P(X > 15) = 1 - P(X \leq 15) = 1 - (1 - e^{-\frac{1}{10}*15}) = e^{-\frac{3}{2}} = 0.22$
2/ Since the exponential distribution has the memoryless property, the probability that the customer spends more than 15mns given that he has already spent 10mns in the bank is equal to the probability that the customer spends at least 5mns in the bank, i.e $P(X > 5)$, $P(X > 5) = e^{-5/10} = e^{-1/2} = 0.604$

Solution 2.
$X \rightsquigarrow Exp(\lambda)$, $\lambda = \frac{5}{24}$ *(one time unit = 1 hour)*
$\mathbb{E}(X) = \frac{1}{\lambda} = 4.8$ *hours*
$P(N_6 > 2) = 1 - P(N_6 \leq 2) = 1 - (P(N_6 = 0) + P(N_6 = 1) + P(N_6 = 2))$
$P(N_6 > 2) = 1 - (e^{-6\lambda} + 6\lambda e^{-6\lambda} + (-6\lambda)^2 \frac{e^{-6\lambda}}{2!}) = 0.1315$

Solution 3.
Let T_1 be the r.v representing the time elapsed until the next bad request arrives.
The Poisson process of receiving requests (of rate $\lambda = 10$) is composed of two types of independent events: erroneous request (with probability $p_1 = 1/12$) and non-erroneous request (with probability $p_2 = 11/12$). According to Proposition 3, each type is a Poisson process of rate λp_i. So, $T_1 \rightsquigarrow \mathcal{E}xp(\lambda p_1)$. The desired probability is equal to $P(T_1 > 5)$.

$$P(T_1 > 5) = 1 - P(T_1 \leq 5) = e^{-10*\frac{1}{12}*5} = e^{-\frac{25}{6}}$$

Solution 4.
Let $N_t \sim \mathcal{P}ois(2.5)$
a- T_n *is the r.v representing the arrival time of n^{th} event, it follows $\mathcal{G}amma(\lambda)$ distribution.* $\mathbb{E}(T_{196}) = \frac{196}{\lambda} = 78,4$ *years.*
b- $\mathbb{V}(T_{196}) = \frac{196}{\lambda^2} = 31,36$.
c- We apply CLT:

$$\begin{aligned} P(T_{196} < 67,2) &= P(\sum_{i=1}^{196} S_i < 67,2) \\ &= P(\frac{\sum_{i=1}^{196} S_i - 196/2.5}{\sqrt{196}/2.5} < \frac{67.2 - 196/2.5}{\sqrt{196}/2.5}) \\ &\approx \phi(2.035) = 0.9788 \end{aligned}$$

Solution 5.
Let $N(t)$ be the r.v representing the number of goals at time t. $N(t) \rightsquigarrow \mathcal{P}ois(\lambda)$, $\lambda = 90 \times \frac{1}{30} = 3$ *(time unit is 90mns)*

1. $P(N_1 \geq 3) = 1 - \sum_{k=0}^{2} e^{-3} \times \frac{3^k}{k!} = 0.5768$
2. $P(N_{\frac{1}{2}} = 2) \times P(N_{\frac{1}{2}} = 1) = e^{-1.5} \frac{(1.5)^2}{2!} \times e^{(-1.5)} \times 1.5 = 0.084$

Chapter 4

Markov Chains

4.1 Introduction

In epidemiological science, for instance, when a disease is spreading in a community, the number of infected people at any time may increase or decrease depending on infections of healthy individuals and recovery of those infected at the present time. Any person can be in one of three states: healthy, infected, or recovered. Its future interaction with others changes its current state and consequently the number of infected people in the population. In this phenomenon, the state of the community and of individuals depends only on their current state. This behavior controls the number of future infections, and its study can predict the trend of transmissible diseases. In the real world, many systems exhibit similar behavior: the future state depends only on the present state.

The mathematical model that allows to model and study this kind of systems is the Markovian stochastic process. In the case where the system has a countable number of states, this process is called a *Markov chain "MC"*.

4.2 Discrete Time Markov Chains

Definition 1.
A discrete time MC "DTMC" $(X_n)_{n\geq 0}$ is a discrete-time Markovian stochastic process (which satisfies the memoryless property):

$$\forall i, j, i_{n-1}, i_{n-2}, ...i_0 \in S$$

$$P(X_{n+1} = j|X_n = i) = P(X_{n+1} = j|X_n = i, X_{n-1} = i_{n-1}, X_{n-2} = i_{n-2}, ..., X_0 = i_0)$$

S is the set of all possible states.
If S is finite then the MC is said to be finite otherwise it is infinite.

$P(X_{n+1} = j|X_n = i)$ is *the transition probability* from state i to state j at the n^{th} step.
The MC is said to be *homogeneous* if the transition probabilities do not depend on the instant n in consideration:

$$P(X_{n+1} = j|X_n = i) = P(X_n = j|X_{n-1} = i) = ... = P(X_1 = j|X_0 = i) = p_{ij}$$

p_{ij} is sometimes noted $\pi(i,j) \quad or \quad \pi_{ij}$. Note that p_{ij} does not denote the joint probability but the conditional probability of being in state j given that we were in state i. It is also called the one-step transition probability. In the rest of this book we only deal with homogeneous DTMCs. An homogeneous DTMC is completely determined by:

- *the states set* S.

- the *initial probabilities vector* $\boldsymbol{\pi}_0$ which contains the probability distribution of being initially in each of the states: $\boldsymbol{\pi}_0(j) \quad \forall j \in S$ satisfying : $\boldsymbol{\pi} \times \mathbf{1} = \sum_{j \in S} \boldsymbol{\pi}_0(j) = 1$ ($\mathbf{1}$ is a vector that has the same size as $\boldsymbol{\pi}_0$ and all its elements are one).

- The transition probabilities p_{ij} in one step as a square matrix $\mathbf{P} = (p_{ij})$.

4.2.1 Transition matrix and graph

The transition matrix $\mathbf{P}$ of an homogeneous DTMC is a stochastic matrix composed of the elements p_{ij}, the probabilities of going from state i to state j in a single step. This matrix is of size $|S| \times |S|$; all of its elements are probabilities, so between 0 and 1, and the sum of each row is equal to 1.
The *transition graph* corresponding to matrix $\mathbf{P}$ is composed of vertices representing the states and of arcs corresponding to the possible transitions (having non-zero probabilities).

Example 1. *Consider the DTMC modeling the weather condition of a given city, having the set of states $S = \{sun, rain\}$, the vector of initial probabilities $\boldsymbol{\pi_0} = [0, 1]$ and the transition matrix* $\mathbf{P}$.

$$\mathbf{P} = \begin{bmatrix} 0.7 & 0.3 \\ 0.5 & 0.5 \end{bmatrix}$$

The elements p_{ij} of the matrix are the probabilities of going from a day of type i to a day of type j, $\quad i, j \in S$. The following graph is the transition graph of the DTMC.

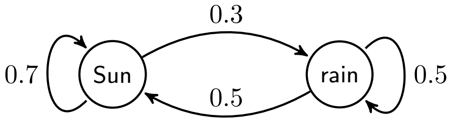


Figure 4.1: Transition Graphe

Example 2. *We have two boxes A and B, each contains r balls. Altogether there are: r red balls and r green balls. At each step, a ball is randomly chosen from A and another one is randomly chosen from B, and each of them is moved from its box to the other box.*
1- Define a DTMC (with its transition matrix) that describes the number of red and green balls in each box.
Solution
Let X_n be the number of red balls in box A after the n^{th} transfer. The process $(X_n)_{n \geq 0}$ is a DTMC with the states set $S = \{0, 1, 2, ..., r\}$. The transition matrix $\mathbf{P}(p_{ij})$ is defined as:

1. $p_{i,i+1} = \frac{(r-i)^2}{r^2}$
2. $p_{i,i-1} = \frac{i^2}{r^2}$
3. $p_{i,i} = \frac{2i(r-i)}{r^2}$
4. $p_{i,j} = 0$ *otherwise*

Case 1 occurs when a green ball is chosen from A with probability $(r - i)/r$ and a red ball is chosen from B with probability $(r - i)/r$.
Case 2 occurs when a red ball is chosen from A with probability i/r and a green ball is chosen from B with probability i/r.
Case 3 occurs when a red ball is chosen from A and B or a green ball is chosen from A and B.
Case 4 cannot occur.

Let's code!

In the following code, we propose a class *DTMC* that implements a DTMC whose attributes are: the set of states S, the transition matrix P and the initial distribution $pi0$. The main methods of the class will be implemented progressively in the next sections of this chapter.

```
#Code401.py (Continuation)

    # nSteps probabilities
    def nSteps(self,n):
        return self.pi0.dot(np.linalg.matrix_power(self.P,n))
```

4.2.2 Probability law of X_n

Let $\boldsymbol{\pi}_n(j) = P(X_n = j) = P_{X_n}(j)$ for $n \geq 0$ and $j \in S$ be the probability that the process is in state j at instant (step) n. The vector $\boldsymbol{\pi}_n = [\pi_n(1), \pi_n(2), \pi_n(3), ...]$ is a probability distribution (the sum of the terms is equal to 1).

Knowing $\boldsymbol{\pi}_0$ the vector of initial probabilities, we want to determine $\boldsymbol{\pi}_n$ $\forall n$. We have:

$$\begin{aligned}
\boldsymbol{\pi}_1(j) &= \sum_{k \in S} P(X_1 = j \cap X_0 = k) \quad \text{total probability theorem,} \\
&= \sum_{k \in S} P(X_1 = j | X_0 = k) \times P(X_0 = k) \quad \text{chain rule} \\
&= \sum_{k \in S} \boldsymbol{\pi}_0(k) \times p_{kj}
\end{aligned}$$

$\boldsymbol{\pi}_1 = \boldsymbol{\pi}_0 \times \mathbf{P}$

In the same way: $\boldsymbol{\pi}_2 = \boldsymbol{\pi}_1 \mathbf{P} = \boldsymbol{\pi}_0 \mathbf{P}^2$

In general: $\boldsymbol{\pi}_n = \boldsymbol{\pi}_{(n-1)} \mathbf{P}$

$$\boldsymbol{\pi}_n = \boldsymbol{\pi}_0 \times \mathbf{P}^n$$

$\boldsymbol{\pi}_n$ is the *Probability law* of X_n.

4.2.3 n steps transition probabilities

$p_{ij}^{(2)}$ is the probability of going from i to j in 2 steps.

$$\begin{aligned}
p_{ij}^{(2)} &= P(X_2 = j | X_0 = i) = \sum_{k \in S} P(X_2 = j | X_1 = k, X_0 = i) \times P(X_1 = k | X_0 = i \\
&= P(X_2 = j | X_0 = i) = \sum_{k \in S} P(X_2 = j | X_1 = k) \times P(X_1 = k | X_0 = i) \quad \text{Markov property} \\
&= P(X_2 = j | X_0 = i) = \sum_{k \in S} p_{ik} \times p_{kj}
\end{aligned}$$

In general: $p_{ij}^{(n)} = P(X_n = j | X_0 = i)$ is the element i, j of matrix $\mathbf{P}^{(n)}$ which is the *n steps transition Probability*.

4.2.4 Chapman Kolmogorov equation

For $n \geq 2$

$$p_{ij}^{(n)} = \sum_{k \in S} p_{ik}^{(n-1)} \times p_{kj} \quad \forall i, j \in S$$

The probability of going from i to j in n transitions is obtained by summing (for all states k) the probabilities of the mutually exclusive events of going from i to a certain state k in $n-1$ steps then going from k to j at the n^{th} step.

Matrix form: let $\mathbf{P}^{(n)}$ be the transition probability matrix in n steps.

$$\mathbf{P^{(n)}} = \mathbf{P^{(n-1)}} \times \mathbf{P} = \mathbf{P} \times \mathbf{P^{(n-1)}}$$

This relation is called the *Chapman Kolmogorov equation*.
We have $\mathbf{P}^{(1)} = \mathbf{P}$, by iterating the formula, we get:

$$\mathbf{P^{(n)}} = \mathbf{P} \times \mathbf{P} \times ... \times \mathbf{P} = \mathbf{P^n}$$

Example 3. *In each chocolate packet we purchase there is a collectible figurine. The complete serie includes 4 figurines. Initially, we have one purchased packet. We are trying to get the complete collection.*
1. Represent the collection process by a Markov chain.
2. Establish the transition matrix. Calculate the probability distribution vector after three purchases.
Solution
Let X_n be the number of different collected figurines after the n^{th} purchase. The process $(X_n)_{n\geq 0}$ is a MC defined by $S = \{1, 2, 3, 4\}$ such that each state represents the number of the different collected figurines. The transition probabilities are as follows:

- $p_{i,i+1} = \frac{4-i}{4} \quad for \quad i = 1, .., 3$
- $p_{i,i} = \frac{i}{4} \quad for \quad i = 1, .., 4$
- $p_{i,j} = 0 \quad otherwise$

$$\mathbf{P} = \begin{bmatrix} 0.25 & 0.75 & 0 & 0 \\ 0 & 0.5 & 0.5 & 0 \\ 0 & 0 & 0.75 & 0.25 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The initial distribution (probability vector): $\boldsymbol{\pi}_{0=}(1, 0, 0, 0)$. *After three purchases:*

$$\mathbf{P^3} = \begin{bmatrix} 0.0156 & 0.3281 & 0.5625 & 0.0938 \\ 0 & 0.125 & 0.5938 & 0.2813 \\ 0 & 0 & 0.4219 & 0.5781 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

So $\boldsymbol{\pi}_3 = \boldsymbol{\pi}_0\mathbf{P}^3 = (0.016, 0.33, 0.56, 0.09)$

Let's code!
We have added the following method that calculates the probabilities in n steps to the DTMC class.

```
#Code401.py (Continuation)

    # nSteps probabilities
    def nSteps(self,n):
        return self.pi0.dot(np.linalg.matrix_power(self.P,n))
```

Let's code!
This code shows its application to the precedent example.

```
#Code401.py (Continuation)

    # nSteps probabilities
    def nSteps(self,n):
        return self.pi0.dot(np.linalg.matrix_power(self.P,n))
```

Example 4. *In a given city, the weather can be sunny, cloudy, or rainy. The weather for the next day depends only on today's weather and not on the previous days' weather. If some day the weather is sunny, the next day will be sunny, cloudy, rainy with the respective probabilities* 0.7, 0.1 *and* 0.2*; when it is cloudy, the probabilities are* 0.5, 0.25, 0.25 *and the probabilities are* 0.4, 0.3, 0.3 *for a rainy day.*
If on some given day the weather is rainy, what is the probability that it will be sunny after three days?
Solution
We consider a MC with three states: 1 (sunny), 2 (cloudy) and 3 (rainy). Let X_n be the r.v representing the weather at the n^{th} day. The stochastic process $X_1, X_2, ...X_n$ is a DTMC with states $S = \{1, 2, 3\}$ and the transition matrix $\mathbf{P}$*:*

$$\mathbf{P} = \begin{bmatrix} 0.70 & 0.10 & 0.20 \\ 0.50 & 0.25 & 0.25 \\ 0.40 & 0.30 & 0.30 \end{bmatrix}$$

To calculate the probability of sunny weather after three days of a rainy day, we can calculate the matrix $\mathbf{P^3}$.

$$\mathbf{P^3} = \begin{bmatrix} 0.60150 & 0.16825 & 0.23025 \\ 0.59125 & 0.17525 & 0.23125 \\ 0.58550 & 0.17975 & 0.23475 \end{bmatrix}$$

So the requested probability is: 0.5855

Another way to find this probability is to calculate $\boldsymbol{\pi}_3$ *starting with* $\boldsymbol{\pi}_0 = (0, 0, 1)$ *using the relationship:*
$\boldsymbol{\pi}_3 = \boldsymbol{\pi}_2\mathbf{P}$ *such that :*
$\boldsymbol{\pi}_1 = \boldsymbol{\pi}_0\mathbf{P} = (0.4000, 0.3000, 0.3000)$
$\boldsymbol{\pi}_2 = \boldsymbol{\pi}_1\mathbf{P} = (0.5500, 0.2050, 0.2450)$
$\boldsymbol{\pi}_3 = \boldsymbol{\pi}_2\mathbf{P} = (0.5855, 0.1797, 0.2347)$
and we get the same value: 0.5855

4.2.5 Classification of states

Let $(X_n)_{n\geq 0}$ be a DTMC having the state space S, the initial probability vector $\boldsymbol{\pi}_0$ and the transition matrix $\mathbf{P}$.

- We say that state j is *accessible* from state i if the probability of going from i to j is non-zero:

$$i \to j \Leftrightarrow \exists n > 0 : p_{ij}^{(n)} > 0$$

  This means that in the transition graph there is a path between i and j.

- We say that states i and j *communicate* if each of them is accessible from the other:

$$i \leftrightarrow j \Leftrightarrow i \to j \wedge j \to i$$

The communication relationship between two states is:

- reflexive (by convention $\forall i \quad \boldsymbol{\pi}_0(i, i) = 1$),
- symmetric (by definition)
- transitive (by Chapman Kolmogorov equation)

It is therefore an equivalence relationship and it is possible to construct a partition of the MC's states into equivalence classes such that all the states of a class communicate with each other and that two states belonging to two different classes do not communicate. These classes are pairwise disjoint and their union gives the set of all the states.
Two types of classes exist:

- *Recurrent* class: it is impossible to leave it.

$$C \text{ is recurrent } \equiv \forall i \in C, \forall j \in S, i \to j \implies j \in C$$

- *Transient* class: it is possible to get out of it without ever being able to come back.

$$C \text{ is transient } \equiv \exists i \in C, \exists j \in S, i \to j \wedge j \notin C$$

Example 5. *Consider the DTMC defined by the following transition graph:*

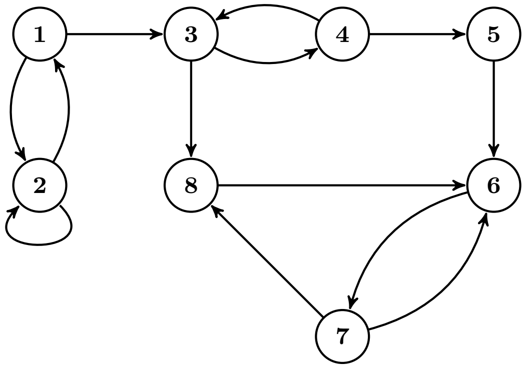


Figure 4.2: States classification

In this example, there are 4 classes:
C1={1,2} transient
C2={3,4} transient
C3={5} transient
C4={6,7,8} recurrent

If the recurrent class is made up of a single state, this state is said to be *absorbing*.
A Markov chain for which there is only one class which is recurrent (equal to the states set) is said *irreducible*.

Let's code!

The method *classify()* has been added to the CMTD class as follows:

```
#Code401.py (Continuation)

    # classify
    def classify(self):
        graf, succ = makeGraf(self.S,self.P)
        cfcs=list(nx.strongly_connected_components(graf))
        classes={"transitoire":[],"reccurente":[]}
        for i in range(len(cfcs)):
            cfc , voisins = cfcs[i] ,set()
            for s in cfc: voisins = voisins|succ[s]-cfc
            classes["transitoire" if(len(voisins)>0) else "reccurente"].append(cfc)
        return classes
```

Let's code!

The following method of the CMTD class allows us to test if a MC is irreducible.

```
#Code401.py (Continuation)

    # is_irreducible
    def is_irreducible(self):
        return(nx.is_strongly_connected(makeGraf(self.S,self.P)[0]))
```

Let’s code!

Its application to the previous example is in the following code.

```
#Code403.py

from CMTD import CMTD

# Classification example
P = [[0.00, 1/2 , 1/2 , 0.00, 0.00, 0.00, 0.00, 0.00],
     [1/3 , 2/3 , 0.00, 0.00, 0.00, 0.00, 0.00, 0.00],
     [0.00, 0.00, 0.00, 1/4 , 0.00, 0.00, 0.00, 3/4 ],
     [0.00, 0.00, 1/2 , 0.00, 1/2 , 0.00, 0.00, 0.00],
     [0.00, 0.00, 0.00, 0.00, 0.00, 1.00, 0.00, 0.00],
     [0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 1.00, 0.00],
     [0.00, 0.00, 0.00, 0.00, 0.00, 2/3 , 0.00, 1/3 ],
     [0.00, 0.00, 0.00, 0.00, 0.00, 1.00, 0.00, 0.00]]

print(CMTD(P).classify())
print(CMTD(P).is_irreducible())
#_____________________________   Output   _____________________________________
#{’transitoire’: [{5}, {3, 4}, {1, 2}], ’reccurente’: [{8, 6, 7}]}
```

4.2.5.1 Recurrent and transient states

It is often useful to know if, starting from state i, the process will return to this state or not.

Consider the r.v $T_i = Min\{n \geq 0 : X_n = i\}$ representing the time of the first passage through state i.

Consider the r.v $T_{ji} = T_i | X_0 = j$ representing the time of the first passage through state i given that it was initially in j

Let the r.v $T_{ii} = T_i | X_0 = i$ be the time of the first return to state i.
The value $\mathbb{E}(T_{ii}) = \sum_{k=0}^{\infty} kP(T_{ii} = k)$ represents the average return time to i .

Let $f_{ii} = P(T_{ii} < \infty)$ be the probability that the process returns to state i given that it was initially in i.

Definition 2.
A state i of a DTMC is said:
*1- **recurrent** if $f_{ii} = P(T_{ii} < \infty) = 1$*
*2- **transient** if $f_{ii} = P(T_{ii} < \infty) < 1$.*
*3- **positive recurrent** if $\mathbb{E}(T_{ii}) < \infty$*
*4- **null recurrent** if $\mathbb{E}(T_{ii}) = \infty$*

Proposition 1.

Let N_i be the number of visits to state i of a DTMC before leaving it definitively. The average number N_i satisfies:

$$\mathbb{E}(N_i | X_0 = i) = \sum_{n=1}^{\infty} p_{ii}^{(n)}$$

We can express the r.v N_i as follows: $N_i = \sum_{n=1}^{\infty} \mathbb{1}_{X_n = i}$ given that $X_0 = i$ such that its average is given by:

$$\mathbb{E}(N_i | X_0 = i) = \mathbb{E}(\sum_{n=1}^{\infty} \mathbb{1}_{X_n = i} | X_0 = i) = \sum_{n=1}^{\infty} \mathbb{E}(\mathbb{1}_{X_n = i} | X_0 = i)$$

$$= \sum_{n=1}^{\infty} 1 \times P(X_n = i | X_0 = i) + 0 \times P(X_n \neq i | X_0 = i)$$

$$\mathbb{E}(N_i | X_0 = i) = \sum_{n=1}^{\infty} P(X_n = i | X_0 = i)$$

$$= \sum_{n=1}^{\infty} p_{ii}^{(n)}$$

if i is:

1- **Recurrent**:
Starting from this state, the probability of returning back is 1 and therefore the probability of returning back again is always 1, this means that the state i will be visited an infinity of times.

$$P(N_i < \infty | X_0 = i) = 0$$
$$P(N_i = \infty | X_0 = i) = 1$$

So a recurrent state has the property that the average number of passages through i is infinite.

$$\mathbb{E}(N_i | X_0 = i) = \sum_{n=1}^{\infty} p_{ii}^{(n)} = 0 \times \mathbb{E}(N_i < \infty | X_0 = i) + 1 \times \mathbb{E}(N_i = \infty | X_0 = i) = \infty$$

so i is a recursive state if and only if $\sum_{n=1}^{\infty} p_{ii}^{(n)} = \infty$

Recurrent states of a finite-state MC are all positive, which is not the case for a infinite-state MC.

2- **Transient**
The probability of going back f_{ii} is less than 1 and therefore the probability of never going back there is $1 - f_{ii}$. Let $T_{ii}^{(h)}$ be the h^{th} return to state i

> **Proposition 2.**
> *If i is a transient state then the r.v $N_i | X_0 = i$ follows the geometric law with parameter $1 - f_{ii}$*

$$\begin{aligned} P(N_i = k | X_0 = i) &= P((\bigcap_{h=1}^{k-1} T_{ii}^{(h)} < \infty) \cap (T_{ii}^{(k)} = \infty)) \\ &= \prod_{h=1}^{k-1} P(T_{ii}^{(h)} < \infty) \times P(T_{ii}^{(k)} = \infty) \\ &= (P(T_{ii} < \infty))^{k-1} \times (1 - P(T_{ii} < \infty)) \\ &= f_{ii}^{k-1} \times (1 - f_{ii}) \end{aligned}$$

This means that:
$$N_i | X_0 = i \rightsquigarrow Geo(1 - f_{ii})$$

And then
$$\mathbb{E}(N_i | X_0 = i) = \sum_{n=1}^{\infty} p_{ii}^{(n)} = \frac{1}{1 - f_{ii}} < \infty$$

We have shown that the average number that the process visits i is finite and equal to $\frac{1}{1-f_{ii}}$
So the average return time is infinite
$$\mathbb{E}(T_{ii}) = \infty$$

Recurrence is a class property, which means that all states belonging to a class are all transient or all recurrent.

4.2.6 Case of infinite MCs

1- **A finite DTMC** has the property that the set of recurrent states is non-empty and that the average return time to each recurrent state is finite $\mathbb{E}(T_{ii}) < \infty$.

2- **An infinite DTMC** does not necessarily have the same property as a finite DTMC.
1- If we take for example the DTMC with the following states and transitions:
1- $S = \mathbb{N}$ and $p_{i,i+1} = 1$,
$$\forall i \in S, \forall k \geq 1, P(T_{ii} = k) = 0 < 1$$

All the states are transient and we have $\mathbb{E}(T_{ii}) = \infty$.

2- Consider the infinite discrete Markov chain described by the following transition graph:

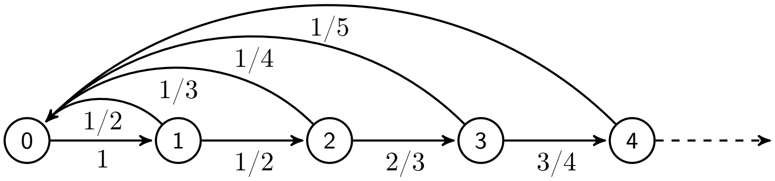


Figure 4.3: Transition Graph

T_0 is the d.r.v of the time of the first passage through state 0.
$T_{i0} = T_0 | X_0 = i$ is the r.v of the first passage through state 0 starting from state i (T_{00} time of first return to 0).

To determine the type of recurrence of the chain (positive or null), we must calculate:
a- The probability that T_{00} has a finite value ($P(T_{00} < \infty)$).
b- The average time to return to 0 ($\mathbb{E}(T_{00})$).

a- The probability that $T_{00} < \infty$:
If returning to 0 is impossible then $T_{00} = \infty$ is a possible event ($P(T_{00} = \infty) > 0$).

$$\begin{aligned}
P(T_{00} \geq 1) &= 1 && \\
P(T_{00} \geq 2) &= 1 && (\ 0 \to 1) \\
P(T_{00} \geq 3) &= 1 \cdot \tfrac{1}{2} && (\ 0 \to 1 \to 2) \\
P(T_{00} \geq 4) &= 1 \cdot \tfrac{1}{2} \cdot \tfrac{2}{3} = \tfrac{1}{3} && (\ 0 \to 1 \to 2 \to 3) \\
P(T_{00} \geq k+1) &= 1 \cdot \tfrac{1}{2} \cdots \tfrac{k-1}{k} = \tfrac{1}{k} && (\ 0 \to 1 \to \cdots \to k)
\end{aligned}$$

Since $\lim_{k\to\infty} P(T_{00} \geq k) = \lim_{k\to\infty} \frac{1}{k-1} = 0$, then $P(T_{00}{=}\infty) = 0$
and we have $P(T_{00}{<}\infty) = f_{00} = 1$
We conclude that the chain is recurrent.

b- The average time to return to 0 ($\mathbb{E}(T_{00})$).

$$\begin{aligned}
\mathbb{E}[T_{00}] &= \sum_{k=1}^{\infty} k \cdot P(T_{00} = k) \\
&= \sum_{k=1}^{\infty} \sum_{j=1}^{k} P(T_{00} = k) = \sum_{j=1}^{\infty} \sum_{k=j}^{\infty} P(T_{00} = k) \\
&= \sum_{j=1}^{\infty} P(T_{00} \geq j) = 1 + 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \cdots = \infty
\end{aligned}$$

So this Markov chain is null recurrent
It returns to 0 with a probability 1, but the average return time is infinite, $\mathbb{E}(T_{ii}) = \infty$

4.2.7 Periodic states

A state i is said to be *periodic*, of period t, if a return to this state can only happen at steps: t, $2t$, $3t$, ... s.t $t = PGCD\{n, p_{ii}^{(n)} \geq 0, t \geq 1\}$. If $t = 1$ we say that the state i is *aperiodic*.

Two communicating states have the same period, so the period is constant within the communication classes. The common period of the elements of a class is called *period of the class*. If the period of a class is equal to 1, it is an *aperiodic class*.

Let's code!

The following method of the CMTD class allows us to test whether a MC is aperiodic.

```
#Code401.py (Continuation)

    # is_aperiodic
    def is_aperiodic(self):
        return(nx.is_aperiodic(makeGraf(self.S,self.P)[0]))
```

4.2.8 Mean of the first passage time (first hitting time)

Let t_{ij} be the average number of required steps to reach, for the first time, state j from state i. If $T_j = Min\{n \geq 0 : X_n = j\}$ is the time (number of steps) it takes for the chain to visit state j for the first time then, $t_{ij} = \mathbb{E}[T_j|X_0 = i]$. We have: $t_{ii} = 0$ and $\forall j \neq i$:

$$t_{ij} = 1 + \sum_{k \in S/\{i\}} p_{ik} t_{kj}$$

In matrix form $\mathbf{T}_i = [t_{ji}]_{j \neq i}$, $\hat{\mathbf{P}}_{(i)} = [p_{jk}]_{k \neq i, j \neq i}$:

$$\mathbf{T}_i = \mathbf{1} + \hat{\mathbf{P}}_{(i)} \mathbf{T}_i$$

Let's code!

```
#Code401.py (Continuation)

    # hitting_time : average time to hit for the first time the state given in the argument
    def hitting_time(self,state):
        n, i = len(self.S), self.S.index(state)
        I = np.identity(n); I[i,i] = 0;
        g = np.ones(n); g[i]=0;
        for k in range(n):
            if(self.P[k,k] == 1): I[k,k], g[k] = 0, 0
        return np.matmul(np.linalg.inv(np.identity(n)-np.matmul(I,self.P)), g)
```

4.2.9 Mean return time

Starting from state i, the average number of required steps to return to it is noted r_i. Consider $R_i = Min\{n \geq 1 : X_n = i\}$ therefore $r_i = \mathbb{E}[R_i|X_0 = i]$ We have: $r_i = 1$ if i is absorbing, otherwise :

$$r_i = 1 + \sum_{k \in S} p_{ik} t_{ki}$$

t_{ki} is the mean of the first hitting time to reach state i starting from k.

Let's code!

```
#Code401.py (Continuation)

    # return_time : average return time to the state given in the argument
    def return_time(self,state):
        i = self.S.index(state)
        return 1 + ( 0 if self.P[i,i] == 1 else  np.dot(self.P[i], self.hitting_time(state)))
```

Example 6. *Consider the DTMC defined by $S = \{1, 2, 3\}$ and the following transition matrix:*

$$\mathbf{P} = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 0 & 1/3 & 2/3 \\ 1/2 & 1/2 & 0 \end{bmatrix}$$

Let t_{k1} be the average time of the first passage by state 1 starting from state k and r_1 the average time to return to state 1. Calculate t_{21}, t_{31} and r_1.
t_{21} and t_{31} are obtained by solving the system of equations such that $t_{11} = 0$:

$$\mathbf{T}_1 = \mathbf{1} + \hat{\mathbf{P}}_{(1)}\mathbf{T}_1 = \begin{bmatrix} t_{21} \\ t_{31} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1/3 & 2/3 \\ 1/2 & 0 \end{bmatrix} \begin{bmatrix} t_{21} \\ t_{31} \end{bmatrix} = \begin{cases} t_{21} = 1 + \frac{1}{3}t_{21} + \frac{2}{3}t_{31} \\ t_{31} = 1 + \frac{1}{2}t_{21} + 0t_{31} \end{cases}$$

We get : $[t_{21}, t_{31}]^T = [5, 7/2]$.
We notice that if $X_0 = 1$ then $X_1 = 1$ with probability 1/2 or $X_1 = 2$ with probability 1/2, so: $r_1 = 1 + 1/2t_{11} + 1/2t_{21} = 7/2$.

Let's code!

```
#Code404.py

from CMTD import CMTD

P = [[0.5, 0.5, 0.0],
     [0.0, 1/3, 2/3],
     [0.5, 0.5, 0.0]]

print(CMTD(P).hitting_time(3))
print(CMTD(P).return_time(1))

#______________________________   Output  ______________________________________
# [3.5 1.5 0. ]
# 3.4999999999999999
```

4.2.10 Absorbing probabilities

In the following MC, 0 and 3 are absorbing states.

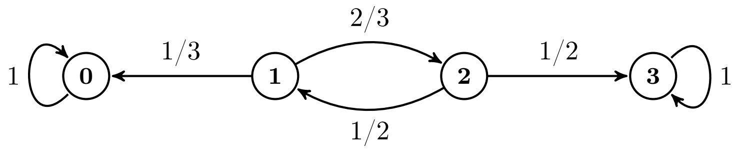


Figure 4.4: Absorbing probabilities

Let a_{i0} be the probability of being absorbed by state 0 starting from state i. $a_{00} = 1$ and $a_{30} = 0$ (because 3 is another absorbing state).

By applying the total probability law, we will have $a_{i0} = \sum_k p_{ik} \times a_{k0}$ this gives the following system of equations such that $a_{00} = 1$ and $a_{30} = 0$:

$$\begin{cases} a_{10} = \frac{1}{3}a_{00} + \frac{2}{3}a_{20} \\ a_{20} = \frac{1}{2}a_{10} + \frac{1}{2}a_{30} \end{cases}$$

Its resolution gives the values of a_{i0} : $a_{10} = 1/2, \quad a_{20} = 1/4$.
In general, for all absorbing state j:

$$a_{ij} = \sum_k p_{ik} \times a_{kj}$$

Let's code!
The method of calculating absorption probabilities.

```
#Code401.py (Continuation)

    # absorption probability
    def absorbing_proba(self,state):
        n, j = len(self.S) , self.S.index(state)
        absorb = [i for i in range(n) if(self.P[i][i] == 1) ]
        if(j not in absorb):
            print("The state must be absorbant")
            sys.exit(0)
        A, B = np.zeros((n,n)), np.zeros(n)
        A[j][j], B[j] = 1 , 1
        for i in range(n):
            if(i != j):
                if(i in absorb): A[i][i] = 1
                else:
                    for k in range(n):
                        A[i][k] = self.P[i][k] if(k != i) else A[i][i]-1
        return np.linalg.inv(A).dot(B)
```

Let's code!
Code of the example.

```
#Code405.py

from CMTD import CMTD

P = [[1.0, 0.0, 0.0, 0.0],
     [1/3, 0.0, 2/3, 0.0],
     [0.0, 1/2, 0.0, 1/2],
     [0.0, 0.0, 0.0, 1.0]]

print(CMTD(P).absorbing_proba(1))
#______________________________   Output   ______________________________________
#[1.   0.5  0.25 0.   ]
```

4.2.11 Mean absorbing time

Let n_i be the average time until the absorption of the MC starting from state i. It represents the required average time for the process to be absorbed given that its initial state is i. $n_i = \mathbb{E}[T|X_0 = i]$ such that T is the time (number of steps) required for the chain to reach an absorbing state.
The values of n_i are obtained by solving the following system (A is the set of absorbing states), if $i \in A$, then $n_i = 0$, otherwise:

$$n_i = 1 + \sum_{j \notin A} p_{ij} n_j$$

This system can be written as follows :

$$\mathbf{N}_{\bar{A}} = \mathbf{1} + \hat{\mathbf{P}}_{\bar{A}} \mathbf{N}_{\bar{A}}$$

Such that $\mathbf{N}_{\bar{A}} = [n_i]_{i \in \bar{A}}$ and $\hat{\mathbf{P}}_{\bar{A}} = [p_{ij}]_{i,j \in \bar{A}}$

Let's code!
The method of calculating the mean absorption time.

```
#Code401.py (Continuation)

    # average absorption time
    def absorbing_time(self):
        n = len(self.S)
        absorb = [i  for i in range(n) if(self.P[i][i] == 1)]
        if(len(absorb) == 0):
            print("No absorbant state exists!")
            sys.exit(0)
        A, B = np.zeros((n,n)), np.zeros(n)
        for i in range(n):
            if(i in absorb): A[i][i]=1
            else:
                B[i] = -1
                for j in range(n):
                    if(j not in absorb): A[i][j]=self.P[i][j]
                A[i][i] -= 1
        return np.linalg.inv(A).dot(B)
```

Example 7.
In Example 3, state 4 is absorbing and it is the winning state. If we want to calculate the

absorption probabilities by this state (probability of winning), we simply have to solve the system of equations with $A = \{4\}$:

$$\mathbf{N}_{\bar{A}} = \mathbf{1} + \hat{\mathbf{P}}_{\bar{A}}\mathbf{N}_{\bar{A}} = \begin{bmatrix}1\\1\\1\end{bmatrix} + \begin{bmatrix}1/4 & 3/4 & 0\\ 0 & 1/2 & 1/2\\ 0 & 0 & 3/4\end{bmatrix}\begin{bmatrix}n_1\\n_2\\n_3\end{bmatrix} = \begin{cases} n_1 = 1 + 1/4 \times n_1 + 3/4 \times n_2 \\ n_2 = 1 + 1/2 \times n_2 + 1/2 \times n_3 \\ n_3 = 1 + 3/4 \times n_3 \end{cases}$$

Which gives : $n_1 = 22/3$, $n_2 = 6$, $n_3 = 4$ and $n_4 = 0$

Let's code!

The method of calculating absorption probabilities.

```
#Code406.py

from CMTD import CMTD

P = [[1/4, 3/4, 0.0, 0.0],
     [0.0, 1/2, 1/2, 0.0],
     [0.0, 0.0, 3/4, 1/4],
     [0.0, 0.0, 0.0, 1.0]]

print(CMTD(P).absorbing_time())

#______________________________ Output _____________________________________
#[7.33333333 6.         4.         0.        ]
```

4.2.12 Stationary and limit distributions

If we take example 4 again and we calculate the successive powers of matrix $\mathbf{P}$, we will realize that there is a limiting probability for the system to be in a some state after a large number of transitions and this probability is independent of the initial state. In the example we have:

$$\mathbf{P} = \begin{bmatrix}0.7 & 0.1 & 0.2\\ 0.5 & 0.25 & 0.25\\ 0.4 & 0.3 & 0.3\end{bmatrix} \qquad \mathbf{P^2} = \begin{bmatrix}0.62 & 0.155 & 0.225\\ 0.575 & 0.1875 & 0.2375\\ 0.55 & 0.205 & 0.245\end{bmatrix}$$

$$\mathbf{P^5} = \begin{bmatrix}0.596 & 0.171 & 0.231\\ 0.595 & 0.172 & 0.231\\ 0.595 & 0.172 & 0.231\end{bmatrix} \qquad \mathbf{P^8} = \mathbf{P^9} = ... = \begin{bmatrix}0.596 & 0.172 & 0.231\\ 0.596 & 0.172 & 0.231\\ 0.596 & 0.172 & 0.231\end{bmatrix}$$

We notice that from power 8 and on, the three lines have identical values, this implies that the probabilities of having a given weather after 8 days are independent of the weather we had in the initial day.

In general, when a MC contains transient and recurrent states, after a large number of steps, the process moves to a recurrent class and never leaves it. So if i is transient, $\lim\limits_{n\to+\infty} p_{ij}^{(n)} = 0 \quad \forall j$. Also, if i and j belong to two different recurrent classes, then $p_{ij}^{(n)} = 0 \quad \forall n$. So the analysis of long-term behavior makes sense only for MCs without absorbing states; for this reason we are interested in this section in irreducible MCs where all the states belong to the same recurrent class. If in addition to being irreducible, the finite MC (a finite number of states) is aperiodic, then it is said *ergodic*; and for any ergodic MC, the $\lim\limits_{n\to+\infty} p_{ij}^{n} \quad \forall i,j$ exists and is independent of i.

Let's code!

The following method of the DTMC class allows us to test whether a MC is ergodic.

```
#Code401.py (Continuation)

    # is_ergodic
    def is_ergodic(self):
        return (self.is_irreducible() and self.is_aperiodic())
```

The aperiodicity property is necessary because if we consider for example the CM described by the following transition matrix:

$$\mathbf{P} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

This chain is irreducible but periodic, this means that as a function of n, $p_{ij}^{(n)}$ takes the values 0 and 1 , therefore, it has no limit as n tends to infinity.

Any finite and irreducible DTMC has a unique *stationary distribution* $\boldsymbol{\pi} = (\pi_1, \pi_2, \pi_3, .., \pi_{|S|})$

1. The π_j are independent of the initial state i and are the unique solution of the system of the linear equations:

$$\pi_j = \sum_{k \in S} \pi_k \times p_{kj} \quad \forall j \in S \quad \textit{and} \quad \sum_{j \in S} \pi_j = 1.$$

2. If in addition, the MC is aperiodic, then, $\lim\limits_{n \to +\infty} p_{ij}^{(n)} = \pi_j \quad \forall i, j \in S.$

The previous system of equations can be written in the form:

$$\begin{cases} \boldsymbol{\pi} \times \mathbf{P} = \boldsymbol{\pi} \\ \boldsymbol{\pi} \times \mathbf{1} = 1 \end{cases}$$

Let's code!

The following code implements the method that calculates the steady-state probabilities.

```
#Code401.py (Continuation)

    # steady state probabilities
    def steady_prob(self):
        if(not self.is_ergodic()):return None
        n = len(self.S)
        A = np.vstack([self.P.T - np.identity(n),np.ones(n)])
        B = np.append(np.zeros(n),1)
        return np.linalg.lstsq(A,B)[0]
```

Example 8. *The stationary distribution in Example 10 is obtained by solving the system of equations:*

$$\begin{cases} 0.7\pi_1 + 0.5\pi_2 + 0.4\pi_3 = \pi_1 \\ 0.1\pi_1 + 0.25\pi_2 + 0.3\pi_3 = \pi_2 \\ 0.2\pi_1 + 0.25\pi_2 + 0.3\pi_3 = \pi_3 \\ \pi_1 + \pi_2 + \pi_3 = 1 \end{cases}$$

We get: $\pi_1 = 0.596 \quad \pi_2 = 0.172 \quad \pi_3 = 0.231$. *Since the chain is aperiodic, these probabilities correspond to the limit probabilities. In this case, the* π_i *can have two possible interpretations:*

1. *As probabilities: in the distant future, a day will be sunny with probability* 0.596*, cloudy with probability* 0.172*, and rainy with probability* 0.231*.*

2. *As proportions: In the long run,* 59.6% *of the days will be sunny,* 17.2% *cloudy, and* 23.1% *rainy.*

However, if the MC is periodic, the π_i *will have only the second interpretation.*

Let's code!

```
#Code407.py

from CMTD import CMTD

P = [[0.70, 0.10, 0.20],
     [0.50, 0.25, 0.25],
     [0.40, 0.30, 0.30]]

print(CMTD(P).steady_prob())

#______________________________   Output   _______________________________________
#[0.59602649 0.17218543 0.23178808]
```

In an irreducible finite state MC, if r_j is the mean time to return to state j, then: $r_j = \frac{1}{\pi_j}$. This result can be explained by the fact that any recurrent state j can serve as a regeneration state, i.e a state from which the process can recycle. If we take a cycle as the time between two successive visits to state j, then the average length of a cycle is r_j, so in the long term, the average rate of visits to state j per unit of time is $\frac{1}{r_j}$ which represents the proportion of time that the process spends in state j and which is given by π_j.

4.3 Continuous Time Markov Chains

In a DTMC, the time parameter t is discrete (it takes the values 0,1,2, ..) and each time the process reaches a state, it spends a unit of time there before making another transition. The time that the process spends in state i is equal to 1 if $p_{ii} = 0$ and follows a Geometric distribution of parameter (1 - p_{ii}) otherwise.
In a continuous time MC the time that the process spends in state i is an exponential r.v which has the memoryless property.

Definition 3.
A Continuous Time MC "CTMC" $(X_t)_{t\geq 0}$ is a continuous time Markovian stochastic process (satisfies the memoryless property):

$$\forall j, i, i_{n-1}, i_{n-2}, ..., i_0 \in S \qquad t > s > t_{n-1} > t_{n-2} > ... > t_0$$

$$P(X_t = j|X_s = i) = P(X_t = j|X_s = i, X_{t_{n-1}} = i_{n-1}, X_{t_{n-2}} = i_{n-2}, ..., X_{t_0} = i_0)$$

Such that S is the discrete set of all possible states.
If S is finite then the MC is said finite otherwise it is said infinite.

4.3.1 Transition probabilities

Let $(X_t)_{t\geq 0}$ be a CMTC, and S its states set:
- X_t: is the r.v representing the state of the system at time t.
- $p_{ij}(t, s)$: is *the continuous-time transition probability function*. It represents the probability that the process passes from state i to state j in a period of time equal to t starting from s.

$$p_{ij}(t, s) = P(X_{(t+s)} = j|X_s = i)$$

If the transition probabilities are independent of s, they are said *homogeneous*

$$p_{ij}(t) = p_{ij}(t, s) = P(X_{(t+s)} = j|X_s = i) = P(X_t = j|X_0 = i) \qquad \forall s \geq 0$$

The matrix $\mathbf{P}(t) = [p_{ij}(t)]$ is the transition matrix in a duration equal to t (the sum of each row of the matrix equals 1 and the elements are between 0 and 1, $\forall t >= 0$).
When t tends towards 0, the limit of the probability that the process changes its state is zero, i.e the probability that the process changes its state in an infinitely small amount of time is almost zero.

$$\lim_{t\to 0} p_{ij}(t) = \begin{cases} 1 & \text{si i=j} \\ 0 & \text{si } i \neq j \end{cases}$$

Therefore, $\mathbf{P}(0) = \mathbf{I}$ s.t $\mathbf{I}$ is the identity matrix.

Example 9. *Consider the following matrix:*

$$\mathbf{N}(t) = 1/2 \begin{bmatrix} 1 + e^{-t} & 1 - e^{-t} \\ 1 - e^{-t} & 1 + e^{-t} \end{bmatrix}$$

For $\mathbf{N}(t)$ to be a transition matrix, the sum of each row of the matrix must be equal to 1 and the elements are between 0 and 1:

$$1/2(1 + e^{-t} + 1 - e^{-t}) = 1 \qquad \forall t \geq 0$$

Since $0 \leq e^{-t} \leq 1$, so: $0 \leq \frac{1+e^{-t}}{2} \leq 1$ and $0 \leq \frac{1-e^{-t}}{2} \leq 1$, then, $\mathbf{N}(t)$ is a transition matrix.

4.3.2 Sojourn time

The time that the process spends in state i is called the *sojourn time*, it is a r.v noted T_i.

> **Proposition 3.** *The sojourn time T_i follows an exponential distribution.*

Proof

$$\begin{aligned}P(T_i > s+t|T_i > s) &= P(X_r = i \quad for \quad 0 \le r \le s+t|X_r = i \quad for \quad 0 \le r \le s)\\ &= P(X_r = i \quad for \quad s \le r \le s+t|X_r = i \quad for \quad 0 \le r \le s)\\ &= P(X_r = i \quad for \quad s \le r \le s+t|X_s = i) \text{ (Markov property)}\\ &= P(X_r = i \quad for \quad 0 \le r \le t|X_0 = i) \text{ (homogeneity)}\\ &= P(T_i > t)\end{aligned}$$

So T_i follows an exponential distribution.
We denote by λ_i, the parameter of the distribution of T_i.
When the process enters state i, the time it spends in i before going to state $j \neq i$ (if no transition to another state occurs before the transition towards state j) is also an exponential rv denoted by T_{ij} of rate λ_{ij}. T_i, the sojourn time in i is the minimum of T_{ij}, $\forall j \neq i$. If i is an absorbing state then $p_{ii}(t) = 1$ and $\lambda_i = 0$ which means that the process will spend an infinite time in this state.

- When the process leaves state i, it moves to state j with probability $p_{ij}(t)$ satisfying:

$$p_{ii}(t) = 0 \quad \forall i \in S \text{ and } \sum_{j \in S} p_{ij}(t) = 1$$

- The state visited after i is independent of the time spent in i.

4.3.3 Chapman Kolmogorov equation

For $t > 0$, the transition from i to j in a time $t+s$ is given by:

$$p_{ij}(t+s) = \sum_{k \in S} p_{ik}(t) \times p_{kj}(s)$$

We have:

$$\begin{aligned}p_{ij}(t+s) &= P(X_{t+s} = j|X_0 = i)\\ &= \sum_{k \in S} P(X_{t+s} = j|X_t = k, X_0 = i) \times P(X_t = k|X_0 = i) \text{ total probability}\\ &= \sum_{k \in S} P(X_{t+s} = j|X_t = k) \times P(X_t = k|X_0 = i) \text{ Markov property}\\ &= \sum_{k \in S} P(X_s = j|X_0 = k) \times P(X_t = k|X_0 = i) \text{ homogeneity}\\ &= \sum_{k \in S} P_{ik}(t) \times P_{kj}(s)\end{aligned}$$

In matrix form:

$$\mathbf{P}(t+s) = \mathbf{P}(t)\mathbf{P}(s), \forall s, t \ge 0$$

4.3.4 Generator matrix

$$p_{ij}(0) = \begin{cases} 1 & \text{si i=j} \\ 0 & \text{si } i \neq j \end{cases}$$

Let h be an infinitely small time, we define the following quantities:

$$q_{ij} = p'_{ij}(0) = \lim_{h\to 0} \frac{p_{ij}(h) - p_{ij}(0)}{h} = \lim_{h\to 0} \frac{p_{ij}(h) - \mathbb{1}_{i=j}}{h}$$

So we have :

$$p_{ij}(h) = \mathbb{1}_{i=j} + q_{ij}h + o(h)$$

Since : $\sum_{j=0}^{\infty} p_{ij}(h) = 1$

$$1 = \sum_{j=0}^{\infty} p_{ij}(h) = 1 + \sum_{j=0}^{\infty} q_{ij}h + o(h) = 1 + q_{ii}h + \sum_{j=0, j\neq i}^{\infty} q_{ij}h + o(h)$$

$$q_{ii} = -\sum_{j=0, j\neq i}^{\infty} q_{ij}$$

The matrix $\mathbf{Q}$ built of $[q_{ij}]$ is called the *generator* matrix (the sum of each row is equal to 0).

$$\mathbf{Q} = \lim_{h\to 0} \frac{\mathbf{P}(h) - \mathbf{I}}{h} = \mathbf{Q}'(0)$$

Such that $\mathbf{I}$ is the identity matrix. Since h is small enough then:

$$P(X_h = i|X_0 = i) = P(T_i > h) = e^{-\lambda_i h} = 1 - \lambda_i h + o(h)$$

$$\lambda_i = \lim_{h\to 0} \frac{1 - P(X_h = i|X_0 = i)}{h} = \lim_{h\to 0} \frac{1 - p_{ii}(h)}{h} = -q_{ii}$$

• The *transition rate* from state i is $\lambda_i = -q_{ii}$, it represents the average number of times the process leaves state i in a unit of time spent in i.

$$-q_{ii} = \lambda_i = \frac{1}{\mathbb{E}(T_i)}$$

• Similarly, q_{ij} is the transition rate from state i to state j, it is the average number of times the process leaves state i to state j in a unit of time.

Example 10. *Consider a CMTC whose transition matrix is as follows:*

$$\mathbf{P}(t) = 1/2 \begin{bmatrix} 1+e^{-t} & 1-e^{-t} \\ 1-e^{-t} & 1+e^{-t} \end{bmatrix}$$

$$\mathbf{P}'(t) = 1/2 \begin{bmatrix} -e^{-t} & e^{-t} \\ e^{-t} & -e^{-t} \end{bmatrix}$$

$$\mathbf{Q} = \mathbf{P}'(0) = 1/2 \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix}$$

4.3.5 Forward and backward equations

Let $\mathbf{P}'(t)$ be the derivative of matrix $\mathbf{P}(t)$ and $\mathbf{Q}$ the generator matrix of the CTMC.

- The *Forward equations* given by: $\mathbf{P}'(t) = \mathbf{P}(t)\mathbf{Q}$ equivalent to :

$$p'_{ij}(t) = \sum_{k\in S} p_{ik}(t)q_{kj} \quad \forall i, j \in S$$

- The *Backward equations* are given by: $\mathbf{P}'(t) = \mathbf{Q}\mathbf{P}(t)$ equivalent to :

$$p'_{ij}(t) = \sum_{k\in S} q_{ik}p_{kj}(t) \quad \forall i,j \in S$$

Forward equations:

$$\begin{aligned} p_{ij}(t+s) &= \sum_{k\in S} p_{ik}(t)p_{kj}(s) \\ &= p_{ij}(t)p_{jj}(s) + \sum_{k\neq j} p_{ik}(t)p_{kj}(s) \\ &\approx p_{ij}(t)(1+q_{jj}s) + \sum_{k\neq j} p_{ik}(t)q_{kj}s \\ &= p_{ij}(t) + p_{ij}(t)q_{jj}s + s\sum_{k\neq j} p_{ik}(t)q_{kj} \\ &= p_{ij}(t) + s\sum_{k\in S} p_{ik}(t)q_{kj} \end{aligned}$$

So,

$$\frac{p_{ij}(t+s)-p_{ij}(t)}{s} = p'_{ij}(t) = \sum_{k\in S} p_{ik}(t)q_{kj}$$

$$p'_{ij}(t) = -\lambda_j p_{ij}(t) + \sum_{k\neq j} p_{ik}(t)q_{kj}$$

Backward equations:

$$\begin{aligned} p_{ij}(t+s) &= \sum_{k\in S} p_{ik}(s)p_{kj}(t) \\ &= p_{ij}(t)p_{ii}(s) + \sum_{k\neq i} p_{ik}(s)p_{kj}(t) \\ &\approx p_{ij}(t)(1+q_{ii}s) + \sum_{k\neq i} q_{ik}sp_{kj}(t) \\ &= p_{ij}(t) + p_{ij}(t)q_{ii}s + \sum_{k\neq i} q_{ik}sp_{kj}(t) \\ &= p_{ij}(t) + s\sum_{k\in S} q_{ik}(t)p_{kj}(t) \end{aligned}$$

Therefore,

$$\frac{p_{ij}(t+s)-p_{ij}(t)}{s} = p'_{ij}(t) = \sum_{k\in S} q_{ik}p_{kj}(t)$$

$$p'_{ij}(t) = -\lambda_i p_{ij}(t) + \sum_{k\neq i} q_{ik}p_{kj}(t)$$

The solution of the system of differential equations

$$\mathbf{P}'(t) = \mathbf{Q}\mathbf{P}(t) = \mathbf{P}(t)\mathbf{Q}$$

such that $\mathbf{P}(0) = \mathbf{I}$ is given by:

$$\mathbf{P}(t) = \mathbf{e}^{\mathbf{Q}t} + \mathbf{C}$$

$\mathbf{e}^{\mathbf{Q}t}$ is the exponential matrix:

$$\mathbf{e}^{\mathbf{Q}t} = \sum_{k=0}^{\infty} \mathbf{Q}^k \frac{t^k}{k!}$$

In case $\mathbf{Q}$ is diagonalizable ($\mathbf{Q} = \mathbf{UDU}^{-1}$), $\mathbf{e^{Qt}}$ can be calculated as follows:

$$\mathbf{e}^{\mathbf{Q}t} = \mathbf{U}\mathbf{e}^{\mathbf{Dt}}\mathbf{U}^{-1} = \mathbf{U}diag(e^{d_1 t}, e^{d_2 t}, ..., e^{d_n t})\mathbf{U}^{-1}$$

Such that $d_i, \quad 1 \leq i \leq n$ are the elements of the diagonal matrix $\mathbf{D}$ and $diag(e^{d_1 t}, e^{d_2 t}, ..., e^{d_n t})$ is the matrix obtained by applying the function e^x to the elements of the diagonal D.

Example 11. *Let $(X_t)_{t\geq 0}$ be a CMTC whose generator matrix is:*

$$\mathbf{Q} = \begin{bmatrix} -1/2 & 1/2 \\ 1/2 & -1/2 \end{bmatrix}$$

The system solution $\mathbf{P}'(t) = \mathbf{P}(t)\mathbf{Q}$ is $\mathbf{e^{Qt}} + \mathbf{C}$:
Since $\mathbf{Q}$ is diagonalizable, then

$$\mathbf{Q} = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix} \frac{1}{2} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$$

$$\mathbf{e^{Qt}} = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & e^{-t} \end{bmatrix} \frac{1}{2} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} = \frac{e^{-t}}{2} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

So, $\mathbf{P}(t) = \frac{e^{-t}}{2} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} + \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$
We know that $\mathbf{P}(0) = I$

$$\mathbf{P}(0) = \frac{1}{2} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} + \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\mathbf{P}(t) = \frac{e^{-t}}{2} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} + \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

The generator matrix allows us to find the stationary distribution of the MC using the following proposition:

> **Proposition 4.**
> *The probability vector π is the stationary distribution of $X(t)$ if and only if:*
>
> $$\boldsymbol{\pi}\mathbf{Q} = 0$$

Proof
1. If $\boldsymbol{\pi}$ is a stationary distribution, then $\boldsymbol{\pi} = \boldsymbol{\pi}\mathbf{P}(t)$.
By differentiating on the two sides we have :

$$0 = \boldsymbol{\pi}\mathbf{P}'(t) = \boldsymbol{\pi}\mathbf{Q}\mathbf{P}(t)$$

Since $\mathbf{P}(t) \neq 0$ then $\boldsymbol{\pi}\mathbf{Q} = 0$

2. Let $\boldsymbol{\pi}$ be a probability vector that satisfies $\boldsymbol{\pi}\mathbf{Q} = 0$.
We have $\mathbf{P}'(t) = \mathbf{Q}\mathbf{P}(t)$, so $\boldsymbol{\pi}\mathbf{P}'(t) = \boldsymbol{\pi}\mathbf{Q}\mathbf{P}(t) = 0$.
Knowing that $\boldsymbol{\pi}\mathbf{P}'(t)$ is the derivative of $\boldsymbol{\pi}\mathbf{P}(t)$, so $\boldsymbol{\pi}\mathbf{P}(t)$ does not depend on t.
that is: $\boldsymbol{\pi}\mathbf{P}(t) = \boldsymbol{\pi}\mathbf{P}(0) = \boldsymbol{\pi}$, (because $\mathbf{P}(\mathbf{0}) = \mathbf{I}$), so $\boldsymbol{\pi}$ is a stationary distribution.

4.3.6 Stationary probabilities

A CTMC is *irreducible* if:

$$\forall i, j \in S \quad \exists\, t > 0 \ / \ p_{ij}(t) > 0$$

Irreducibility is satisfied in almost all real-world applications.
For CTMCs, periodicity is not a problem because the transition times between states are continuous r.vs.

If a CTMC is irreducible, then: $\lim_{t\to\infty} p_{ij}(t) = \pi_j$ always exists and is independent of the initial state i. The values π_j are the stationary probabilities of the MC and satisfy:

$$\pi_j = \sum_{i\in S} \pi_i p_{ij}(t) \quad \forall j \in S \quad \textit{and} \quad t \geq 0$$

To calculate stationary probabilities, it is more convenient to solve the system given by:

$$\begin{cases} \boldsymbol{\pi} \times \mathbf{Q} = 0 \\ \boldsymbol{\pi} \times \mathbb{1} = 1 \end{cases}$$

and which is equivalent to:

$$\begin{cases} \pi_j q_j = \sum_{i\neq j} \pi_i q_{ij} \quad \forall j \in S \\ \sum_{j\in S} \pi_j = 1 \end{cases}$$

The equations obtained by the first line of this system, are often called *the equilibrium equations* in the sense that the left side of the equation $\pi_j q_j$ represents the rate that the process leaves state j in the long run (since π_j is the stationary probability to be in state j and q_j is the rate that the process leaves state j), while the right side $\pi_i q_{ij}$ represents the rate that the process enters into state j (the sum from all the other states i) which means that the two rates of exit and entry are equal.

4.3.7 Embedded Markov Chain

When the process leaves state i after an exponential time T_i, it passes to another state j with probability p_{ij} called the jump probability. This probability is given by $p_{ij} = \frac{q_{ij}}{\lambda_i}$ for $j \neq i$ and $p_{ii} = 0$.
The matrix $\mathbf{P}$ whose elements are (p_{ij}) is a stochastic matrix which gives the one-step transition probabilities of a discrete Markov chain called the *embedded* Markov chain (integrated Markov chain) of the CTMC. The transition matrix $\mathbf{P}$ determines the probabilistic behavior of the integrated CTMD, but cannot represent the behavior of the continuous-time process because it does not specify the transition rates of the process.
The states classification of a CTMC is the same as the classification of its embedded CTMD.

Let's code!

```
#CMTC.py

import sys
import numpy as np
from CMTD import CMTD

# CMTD subclass of CMTD
class CMTC(CMTD):

    # constructor
    def __init__(self,P, lambdas, S = None, pi0 = None ):
        super().__init__(P ,S ,pi0)
        n = len(P)
        self.lambdas=np.array(lambdas)
        if((len(self.lambdas) != n)or(len(list(filter(lambda x:x<0,self.lambdas))) != 0)):
            print("lambdas should have ",n," positive values")
            sys.exit(0)
        copyP = self.P.copy()
        np.fill_diagonal(copyP,-1)
        self.Q = np.matmul(self.lambdas * np.identity(n),copyP)

    # steady state probabilities
    def steady_prob(self):
        n = len(self.S)
        A = np.vstack([self.Q.T,np.ones(n)])
        B = np.append(np.zeros(n),1)
        return np.linalg.lstsq(A,B)[0]
```

Example 12. *Consider the CMTC $(X_t)_{t\geq 0}$ whose integrated CM have the following transition graph: We assume that: $\lambda_1 = 2$,$\lambda_2 = 1$ and $\lambda_3 = 3$.*

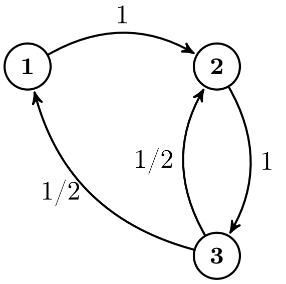


Figure 4.5: Transition graph

1. *Find the generator matrix of the CTMC.*
2. *Find the corresponding limit distribution.*

The generator matrix $\mathbf{Q}$ is defined by:

$$q_{ij} = \begin{cases} \lambda_i p_{ij} & \text{si } i \neq j \\ -\lambda_i & \text{si } i = j \end{cases}$$

$$\mathbf{Q} = \begin{bmatrix} -2 & 2 & 0 \\ 0 & -1 & 1 \\ 3/2 & 3/2 & -3 \end{bmatrix}$$

The limiting distribution is obtained by solving:

$$\boldsymbol{\pi}\mathbf{Q} = 0 \quad \text{and} \quad \boldsymbol{\pi}\mathbb{1} = 1$$

Which gives:

$$\boldsymbol{\pi} = (\frac{3}{19}, \frac{12}{19}, \frac{4}{19})$$

Let's code!

```
#Code408.py

from CMTC import CMTC

p = [[0.0, 1.0, 0.0],
     [0.0, 0.0, 1.0],
     [1/2, 1/2, 0.0]]
lambdas = [2,1,3]

continuous = CMTC(p, lambdas)
print(continuous.steady_prob())

#______________________________   Output   ______________________________________
#[0.15789474 0.63157895 0.21052632]
```

4.3.8 First hitting time

Consider the CTMC $(X_t)_{t\geq 0}$ and let k be a state accessible from the other states. We are interested in the time of the first passage (visit) through state k from state i.
We define T_i the r.v of the time of the first passage by k from i.

$$H_i(t) = P(T_i > t) = P(X_u \neq k \quad for \quad 0 \leq u \leq t | X_0 = i) \quad for \quad t \geq 0, i \neq k$$

We define the CTMC $(\bar{X}_t)_{t\geq 0}$ such that state k is an absorbing state and its generator matrix: $\bar{q}_{ij} = q_{ij}\mathbb{1}_{\{i\neq k\}} \quad \forall j \in S, i \neq k$.
Its transition probability matrix: $\bar{p}_{ij}(t) = P(\bar{X}_t = j|\bar{X}_0 = i)$ and $\bar{p}_{kk}(t) = 1$.
So we have, $H_i(t) = 1 - \bar{p}_{ik}(t)$.
Applying the backward equation:

$$\begin{aligned}
H_i'(t) = -\bar{p'}_{ik}(t) &= -(\bar{q}_{ii}\bar{p}_{ik}(t) + \sum_{l\neq i} \bar{q}_{il}\bar{p}_{lk}(t)) \\
&= -\bar{q}_{ii}\bar{p}_{ik}(t) - \sum_{l\neq i,k} \bar{q}_{il}\bar{p}_{lk}(t) - \bar{q}_{ik}\bar{p}_{kk}(t) \\
&= -q_{ii}(1 - H_i(t)) - \sum_{l\neq i,k} q_{il}(1 - H_l(t)) - q_{ik} \\
&= q_{ii}H_i(t) + \sum_{l\neq i,k} q_{il}H_l(t) - q_{ii} - \sum_{l\neq i,k} q_{il} - q_{ik} \\
&= q_{ii}H_i(t) + \sum_{l\neq i,k} q_{il}H_l(t) - (q_{ii} + \sum_{l\neq i,k} q_{il} + q_{ik})
\end{aligned}$$

$$H_i'(t) = q_{ii}H_i(t) + \sum_{l\neq i,k} q_{il}H_l(t)$$

This system can be written as :

$$\mathbf{H}'(t) = \mathbf{W}\mathbf{H}(t)$$

$\mathbf{W}$ is the matrix whose elements are (q_{ij}) s.t $i \neq k$ and $j \neq k$.

The solution of this differential equation gives us the complement of the cumulative function of the r.v T_i $(1 - H_i(t) = F_i(t))$. To obtain the average time of the first passage, we calculate its expectation:

$$\mathbb{E}(T_i) = \int_0^\infty tf_i(t)dt = -\int_0^\infty tH_i'(t)dt = -tH_i|_0^\infty + \int_0^\infty H_i dt$$

Example 13. *Consider the CTMC $(X_t)_{t\geq 0}$ whose generator matrix is:*

$$\mathbf{Q} = \begin{bmatrix} -2 & 1 & 1 \\ 1 & -2 & 1 \\ 0 & 2 & -2 \end{bmatrix}$$

Find the average passage time from all states to state 2.

$$\mathbf{H}'(t) = \mathbf{W}\mathbf{H}(t)$$

$$\mathbf{W} = \begin{bmatrix} -2 & 1 \\ 1 & -2 \end{bmatrix}$$

The solution of this system:

$$\mathbf{H}(t) = c_1 e^{\lambda_1 t}\mathbf{V}_1 + c_2 e^{\lambda_2 t}\mathbf{V}_2$$

such that λ_1, λ_2, $\mathbf{V}_1$, $\mathbf{V}_2$ are the eigenvalues and eigenvectors of $\mathbf{W}$.

$\lambda_1 = -1: \quad \mathbf{V}_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$

$\lambda_2 = -3: \quad \mathbf{V}_2 = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$

$$\mathbf{H}(t) = c_1 e^{-t} \begin{pmatrix} 1 \\ 1 \end{pmatrix} + c_2 e^{-3t} \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

To calculate c_1 and c_2, we consider the initial conditions: $\mathbf{H}(0) = \mathbf{1}$. We therefore have: $c_1 = 1$ and $c_2 = 0$
To calculate $\mathbb{E}(T_i)$, we have $H_0(t) = H_1(t) = e^{-t}$:

$$\mathbb{E}(T_i) = -tH_i(t)|_0^\infty + \int_0^\infty H_i(t)dt = -(t+1)e^{-t}|_0^\infty = 1 \quad i = 0, 1$$

A simpler method to calculate the average first hitting time is to apply recursive reasoning. Let μ_i be the value of the average transition time through state k starting from state i. The sojourn time in state i is equal to $1/\lambda_i$, the process will then pass to another state j with probability p_{ij}. From j, it reaches state k in an average time equal to μ_j. μ_i can therefore be obtained by the following recursive expression :

$$\mu_i = \frac{1}{\lambda_i} + \sum_{j \neq i,k} p_{ij}\mu_j \quad for \quad i \neq k.$$

Its matrix form can be written as:

$$\mathbf{M}_k = -diag^{-1}(\mathbf{Q})\mathbf{1} + \mathbf{P}^{(k)}\mathbf{M}_k$$

Such that $\mathbf{Q}$ is the generator matrix of the CTMC, $\mathbf{M}_k$ the vector of the mean first hitting times from the other states to state k and $\mathbf{P}^{(k)}$ is the probability matrix of the integrated MC without row and column k. The solution of this system is:

$$\mathbf{M}_k = -(\mathbf{I} - \mathbf{P^{(k)}})^{-1} diag^{-1}(\mathbf{Q})\mathbf{1}$$

$diag(\mathbf{Q})$ is a function that returns a matrix whose diagonal is the same as $\mathbf{Q}$ and the other elements are zero.
The result obtained by this method is: $\mu_0 = 1$ and $\mu_1 = 1$.

This equation expresses the fact that this time is composed of two parts, the first represents the sojourn time in the current state and the second represents the average time to reach k from the other states to which the process has transited after the expiration of his stay in state i.

Let's code!

```
#CMTC.py (Continuation)

# hitting_time : average time to hit for the first time the state given in the argument
def hitting_time(self,state):
    i = self.S.index(state)
    I = np.identity(len(self.S)-1);
    lambdas_inv = np.delete(np.reciprocal(self.Q.diagonal()),i)
    PK = np.delete(np.delete(self.P,i,0),i,1)
    return -np.matmul(np.linalg.inv(I-PK), lambdas_inv)
```

Let's code!

```
#Code409.py

from CMTC import CMTC

p = [[0.0, 0.5, 0.5],
     [0.5, 0.0, 0.5],
     [0, 1, 0.0]]
lambdas = [2,2,2]

continuous = CMTC(p, lambdas)
print("Hitting time: ",continuous.hitting_time(3))

#______________________________     Output  ______________________________________
# Hitting time:  [1. 1.]
```

4.4 Application: PageRank Algorithm

PageRank is an algorithm that analyses hyperlinks to rank web pages. It is used by Google search engine to rank web pages.
PageRank models the web by a directed graph whose vertices are the set of web pages and the arcs are the hyperlinks between them, this structure is called the web graph. The inbound (I_i) and outbound (O_i) links of the page determine the importance of that page. Links from important pages carry more weight than links from less important pages, and a page fairly shares its weight over its successors. Consider:
N the total number of pages.
I_i the set of the pages pointing to i.
O_i the set of the pages to which i points.

In order to represent the navigation process in the web graph, Google modeled it by a DTMC whose states (the set S) are the set of pages, and the transition matrix is called the Google matrix ($\mathbf{G}$), and it is constructed as follows:

$$\mathbf{G} = d\mathbf{H} + (1-d)\mathbf{E}$$

Such that:
$0 < d < 1$ a constant which represents the probability of following the links of the pages (not restarting the navigation).
$\mathbf{H}$ is a matrix of size $N \times N$ defined by: $h_{ij} = \frac{1}{|O_i|}$ if $i \in I_j$ and 0 otherwise.
$\mathbf{E}$ is a matrix of size $N \times N$ defined by: $e_{ij} = \frac{1}{N} \quad \forall\ 1 \leq i, j \leq N$

$\mathbf{G}$ is a stochastic matrix because:
1. Each element of row i of matrix $d\mathbf{H}$ is equal to $\frac{d}{|O_i|}\mathbb{1}_{j\in O_i}$, its sum is equal to d.
2. The sum of each row of $(1-d)\mathbf{E}$ is equal to $1-d$.
So all elements of $\mathbf{G}$ are between 0 and 1 and the sum of each row is 1.

The DTMC defined by $\mathbf{G}$ is aperiodic and irreducible, it therefore admits a unique stationary probability vector
$\boldsymbol{\pi}$, the stationary probability vector of the chain is obtained by the system of equations:

$$\begin{cases} \pi_i = \sum_{j=1}^{N} \pi_j g_{ji} \\ \sum_{i=1}^{N} \pi_i = 1 \end{cases}$$

We have:

$$\pi_i = \sum_{j=1}^{N} \pi_j g_{ji} = \sum_{j=1}^{N} \pi_j \frac{1-d}{N} + \sum_{j \in I_i} \pi_j \frac{d}{|O_j|} = \frac{1-d}{N} + \sum_{j \in I_i} \pi_j \frac{d}{|O_j|}$$

We define R_i the rank of page i as the quantity $N\pi_i$ which verifies the recurring relationship used by Google to determine the ranking of web pages:

$$R_i = (1-d) + \sum_{j \in I_i} R_j \frac{d}{|O_j|}$$

Let's code!

```
#Code410.py

import numpy as np
from CMTD import CMTD

def getRankPage(G,d):
    N = len(G)
    _output = list( map(lambda x: 0 if list(x).count(1) == 0 else  1/list(x).count(1) , G))
    M = np.identity(N) - d * np.multiply(np.array(_output), G.T).T
    M = np.vstack([M.T ,np.ones(N)])
    B = np.hstack([np.ones(N)*(1-d), N])
    return np.linalg.lstsq(M,B)[0]

# Test with getRankPage
d = 0.85
A = np.array([[0, 1, 1],
              [1, 0, 0],
              [0, 1, 0]])
print('R : ', getRankPage(A,d))

# Test with the steady state computation
G = [[0.050, 0.475, 0.475],
     [0.900, 0.050, 0.050],
     [0.050, 0.900, 0.050]]
print( 'R : ', CMTD(G).steady_prob()*3)

#______________________________   Output  ______________________________________
# R : [1.16336914 1.19219898 0.64443188]
# R : [1.16336914 1.19219898 0.64443188]
```

4.5 Exercises

Exercise 1. *For the temperature forecast of a given city, at the end of each day the temperature is recorded. We noticed that if it has increased, the probability that it will increase tomorrow is 0.6, if it has decreased, the probability that it will increase tomorrow is 0.7.*
1. Represent this system by a DTMC.
2. Suppose that the model has changed so that the temperature of tomorrow depends on that of yesterday and today: it increases with probability 0.8 if it has increased today and yesterday. If it increases today and has decreased yesterday, it increases tomorrow with probability 0.4. If it decreases today and has increased yesterday then it increases tomorrow with probability 0.5 and if it decreases today and decreased yesterday then it increases tomorrow with probability 0.3. Represent the system by a DTMC.
3. Code the example in Python.

Exercise 2. *Consider a player who has 2 dinars initially; at each stage of the game, he can win 1 dinar with probability p or lose 1 dinar with probability* $1 - p$. *He stops when he has won* 4 *dinars or when he has lost everything.*
1. Represent this experiment by a DTMC.
2. Classify this DTMC.
3. With $p = 1/3$, *calculate the probability that the player loses everything.*
4. Determine the value of p such that the probability of losing is equal to 0.5.
5. Give the Python code that determines the classes of this DTMC for $p = 1/3$.

Exercise 3. *To obtain a certificate, each candidate must pass two tests (level 1 test and level 2 test). To pass level 2 test, one must pass level 1 test. In case of failure, it is possible to redo each test only once. Someone who retakes a test and fails is automatically excluded.*
Each participant has a probability p of passing each test ($q = 1 - p$ *is the probability of failing).*
1. Using a DTMC, model the path of a candidate.
2. After passing two tests, what are the possible states in which a candidate can be?
3. With $p = 2/3$, *what is the probability that a candidate will obtain his certificate.*
4. Write the Python code that determines the average time of success.

Exercise 4. *With friends you play the following game: three candles numbered 1 to 3 are on a table. A die is rolled repeatedly. If the outcome is 1 or 6 then the first candle is lit if it is unlit or unlit if it is lit. The same is done with the second candle if the obtained outcome is 2 or 5 and with candle 3 if the outcome is 3 or 4. Initially, the three candles are extinguished, and the game is over when all the candles are lit. Let N be the number of the required die rolls until the three candles are lit. Calculate* $\mathbb{E}(N)$.

Exercise 5. *Let* $(X_n)_{n\geq 0}$ *be a DTMC defined by:*
$S = \{1, 2, 3, 4, 5\}$

$$\mathbf{P} = \begin{bmatrix} 1/4 & 0 & 3/4 & 0 & 0 \\ 1/4 & 1/2 & 1/4 & 0 & 0 \\ 1/3 & 0 & 1/3 & 0 & 1/3 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

1. Classify the states of this chain and calculate their period.
2. What is the mean first passage time through state 4?
3. Give the Python code that determines the classification of the states of this chain and the average of the first return time to state 4.

Exercise 6. *Let $(X_n)_{n\geq 0}$ be a DTMC defined by $S = \{1, 2, 3, 4\}$ and $\mathbf{P}$:*

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1/4 & 0 & 3/4 & 0 \\ 0 & 1/3 & 0 & 2/3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

1. Draw the transition graph and classify the states of this chain.
2. Calculate the probability and the mean absorption time starting from state 2.
3. Write the Python code for the 2^{nd} question.

Exercise 7. *Consider the DTMC be described by the following transition matrix.*

$$\mathbf{P} = \begin{bmatrix} 1/4 & 3/4 & 0 \\ 0 & 1/2 & 1/2 \\ 1/3 & 2/3 & 0 \end{bmatrix}$$

1. Find t_{k1}, the average time of the first passage through state 1 starting from state k (for $k = 2, 3$).
2. Find r_1, the average return time to state 1.
3. Write the Python code that determines the average return time to state 1.

Exercise 8. *It is assumed that the income of a person depends on the income of his parents, the relationship is given by the following matrix. For example if the father has a high income, his son will have an average income with a probability 0.4. $S = \{L, M, H\}$:*

$$\mathbf{P} = \begin{bmatrix} 0.5 & 0.3 & 0.2 \\ 0.2 & 0.6 & 0.2 \\ 0.1 & 0.4 & 0.5 \end{bmatrix}$$

L: low, M: medium, H: high
1. What is the probability that the income of a grandson in a low-income family to be high?.
2. In the long term what is the percentage of the population with low income.
3. Write the Python code of question 2.

Exercise 9. *Consider a client / server computer system:*
- the client sends requests and the server returns the corresponding response.
- the server can have a buffer memory to store requests whose size can be unlimited. The system has three components Client (Request), Buffer and Server will be noted: $[\lambda, m = 1, k = 1, \mu, n = 1]$
- λ : requests arrival rate.
- m : number of sent requests (default 1)
- k : size buffer k (infinite) (default 1)*
- n : number of servers (default)
- μ : service rate.
When an element is omitted, it takes the default value. In the case of a memory of size 0, the requests received when the server is busy are lost. Consider the following cases:

$01 - [\lambda, , 0, \mu,]$ $02 - [\lambda, , , \mu,]$ $03 - [\lambda, , 2, \mu,]$ $04 - [\lambda, , k-1, \mu,]$ $05 - [\lambda, , *, \mu,]$

$06 - [\lambda, , 0, \mu, 2]$ $07 - [\lambda, , , \mu, 2]$ $08 - [\lambda, , 2, \mu, 2]$ $09 - [\lambda, , k, \mu, 2]$ $10 - [\lambda, , *, \mu, 2]$

$11 - [\lambda, , 0, \mu, n]$ $12 - [\lambda, , k, \mu, n]$ $13 - [\lambda, , *, \mu, n]$

1- Model each case as a CTMC by giving the states and the transition diagram.

Exercise 10. *Consider a production system whose elements are:*
- Producer (P): is the entity that produces k units of a product A (by default 1)
- Storage (S): is an intermediate storage area of size M where the product A is deposited or withdrawn.
- Consumer (C): is the entity that consumes A by removing h units (by default 1)
The system is denoted $[\lambda, k, S, \mu, h]$
- k: number of units of A produced simultaneously by P (a batch).
- λ*: production rate.*
- S: the maximum capacity of the temporary storage (by default 1).
- h: number of units A withdrawn and consumed simultaneously by C (by default 1).
- μ*: consumption rate.*
Consider the following cases :

$$1-[\lambda,1,1,\mu,1] \quad 2-[\lambda],1,4,\mu,1] \quad 3-[\lambda,2,6,\mu,1] \quad 4-[\lambda,2,8,\mu,3]$$

1- Model each of these cases by a CTMC by giving the transition diagram.

Exercise 11. *Consider the following matrix* $\mathbf{P}(t)$ *:*

$$\mathbf{P}(t) = \mathbf{D} + f(t)\mathbf{A}$$

- $\mathbf{D}$ *is a constant transition matrix of the same size as P.*
- $\mathbf{A}$ *is a constant matrix of the same size as P.*
- f is a positive function .
1- Given $\mathbf{D}$*, What are the conditions on f and* $\mathbf{A}$ *so that* $\mathbf{P}(t)$ *is a transition matrix for a CTMC*
2- Find non-null, integer-valued matrices $\mathbf{A}$ *for* $f(t) = e^{-t}/2$ *and*

$$\mathbf{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \qquad \mathbf{D} = \begin{bmatrix} 1/2 & 1/2 \\ 1/2 & 1/2 \end{bmatrix}$$

Exercise 12. *Consider a population of n individuals. It is initially considered that there is one person infected with a contagious disease, and that the times between contact of any two individuals in the population follows an exponential distribution of rate* λ *and that the times between the different contacts are independent of each other. If a healthy person contacts an infected person, he/she becomes infected too. Once infected, a person always remains so. Model the spread of the disease by a CTMC.*

Exercise 13. *Customers arrive at a beverage vending machine following a Poisson process of rate* λ*. Each customer takes a single unit. When the vending machine is empty, all incoming requests are lost. The vending machine is only filled when it is empty; and its filling happens following a Poisson process rate* μ *which is independent of that of the incoming requests. The filling time is negligible and the quantity filled is always equal to N units. Propose a CMTC to model this description.*

Exercise 14. *An electronic system uses one operating unit, and has* $n-1$ *other replacement units. The lifetime of each unit is exponentially distributed with rate* λ*. If the operating unit fails, it is immediately replaced with another operating unit if available. There are enough repairers, and every broken unit goes straight for repair. The repair time is exponentially distributed of rate* μ*. A repaired unit is as good as a new unit. Model the number of units under repair by a CTMC.*

Exercise 15. *Let* $(X_t)_{t\geq 0}$ *be the CTMC associated to the embedded MC with the following transition graph : We assume that:* $\lambda_1 = 1$*,* $\lambda_2 = 2$*,* $\lambda_3 = 1$ *and* $\lambda_4 = 1$*.*

1. *Find the generator matrix of the CTMC.*
2. *Find the associated limit distribution.*
3. *Determine the average first return time to state 4.*
4. *Write the Python code for questions 2 and 3.*

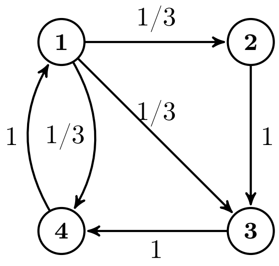


Figure 4.6: Transition graph

4.6 Solutions

Solution 1.
1- Let $S = \{0, 1\}$ be the set of states. State 0 represents the increase in temperature and state 1 represents the decrease in temperature. Thus, the transition matrix is defined by:

$$\mathbf{P} = \begin{bmatrix} 0.6 & 0.4 \\ 0.7 & 0.3 \end{bmatrix}$$

2- If we keep the same set of states as question 1, the memoryless property will be lost because the future will depend on the present and also on the past. To find this property back, we define a new set of states $S = \{0, 1, 2, 3\}$ such that:
- • *state 0: the temperature has increased today and yesterday*
- • *state 1: the temperature has increased today and has dropped yesterday*
- • *state 2: the temperature went down today and increased yesterday*
- • *state 3: the temperature went down today and went down yesterday*

We get the following transition matrix:

$$\mathbf{P} = \begin{bmatrix} 0.8 & 0 & 0.2 & 0 \\ 0.4 & 0 & 0.6 & 0 \\ 0 & 0.5 & 0 & 0.5 \\ 0 & 0.3 & 0 & 0.7 \end{bmatrix}$$

The first element of the second row, for example, represents the probability of increasing tomorrow and today given that it has increased today and has decreased yesterday, it is simply the probability of increasing tomorrow given that it has increased today and has decreased yesterday which equals 0.6. The probability is equal to 0 in the case of inconsistent events.

Solution 2.
1- Let $(X_n)_{n\geq 0}$ be the r.v representing the number of dinars that the player has at step n. The set of states is: $S = \{0, 1, 2, 3, 4\}$. The transition matrix is as follows:

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1-p & 0 & p & 0 & 0 \\ 0 & 1-p & 0 & p & 0 \\ 0 & 0 & 1-p & 0 & p \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

2- The classes of the DTMC are:
- $\{0\}$ *a recurrent class (0 is an absorbing state)*
- $\{4\}$ *a recurrent class (4 is an absorbing state)*

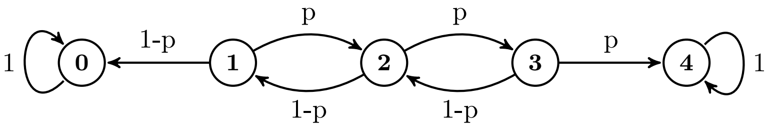


Figure 4.7: Transition graph

- $\{1, 2, 3\}$ *is a transient class (there is a non zero probability to go from the states of this class to the absorbing states).*

3- The probability that the player loses everything is the probability of being absorbed by state 0 starting from state 2: a_{20} *and which can be calculated by the following system:*

$$\begin{cases} a_{10} = \frac{2}{3} + \frac{1}{3} \times a_{20} \\ a_{20} = \frac{2}{3} \times a_{10} + \frac{1}{3} \times a_{30} \\ a_{30} = \frac{2}{3} \times a_{20} \end{cases}$$

$a_{20} = \frac{4}{5}$
4- We solve the system of equations of question 3 ($q = 1 - p$):

$$\begin{cases} a_{10} = q + pa_{20} \\ a_{20} = qa_{10} + pa_{30} \\ a_{30} = qa_{20} \end{cases}$$

So: $a_{20} = \frac{p^2}{2p^2-2p+1}$
For: $a_{20} = \frac{1}{2} \implies \frac{p^2}{2p^2-2p+1} = \frac{1}{2}$
We have: $p = \frac{1}{2}$

Solution 3.
1- $S = \{t_1, r_1, t_2, r_2, R, E\}$
1. t_1*: takes the first test the first time.*
2. r_1*: repeats the first test.*
3. t_2*: takes the second test the first time.*
4. r_2*: repeats the second test.*
5. E*: exclusion*
6. R *: passes (obtaining the certificate)*
The transition matrix is:

$$\mathbf{P} = \begin{bmatrix} 0 & q & p & 0 & 0 & 0 \\ 0 & 0 & p & 0 & q & 0 \\ 0 & 0 & 0 & q & 0 & p \\ 0 & 0 & 0 & 0 & q & p \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

2- $\boldsymbol{\pi}_0 = (1, 0, 0, 0, 0, 0)$, $\boldsymbol{\pi}_2 = \boldsymbol{\pi}_0 \mathbf{P}^2$
After two tests, the candidate can be in:
R *with probability* p^2,
E *with probability* q^2,
t_2 *with probability* qp,
r_2 *with probability* pq

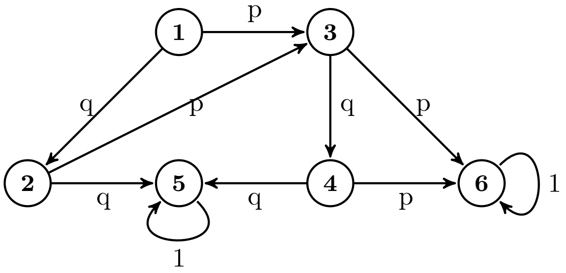


Figure 4.8: Transition graph

3- The following system should be solved such that $a_{ER} = 0$ *and* $a_{RR} = 1$ *:*

$$\begin{cases} a_{t_1R} = \frac{1}{3}a_{r_1R} + \frac{2}{3}a_{t_2R} \\ a_{r_1R} = \frac{2}{3}a_{t_2R} + \frac{1}{3}a_{ER} \\ a_{t_2R} = \frac{1}{3}a_{r_2R} + \frac{2}{3}a_{RR} \\ a_{r_2R} = \frac{1}{3}a_{ER} + \frac{2}{3}a_{RR} \end{cases}$$

$a_{t_1R} = \frac{16}{27}$

Solution 4. *We can define a MC with the set of states* $S = \{0, 1, 2, 3\}$*, each state represents the number of lit candles,* 3 *is therefore an absorbing state, and the transition matrix is as follows:*

$$\mathbf{P} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1/3 & 0 & 2/3 & 0 \\ 0 & 2/3 & 0 & 1/3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

To find $\mathbb{E}(N)$*, which is, in fact, the average absorption time of the chain by state 3 starting from state 0; we must solve the following system such that* $n_3 = 0$*:*

$$\begin{cases} n_0 = 1 + n_1 \\ n_1 = 1 + \frac{1}{3}n_0 + \frac{2}{3}n_2 \\ n_2 = 1 + \frac{2}{3}n_1 + \frac{1}{3}n_3 \end{cases}$$

$\mathbb{E}(N) = n_0 = 10$

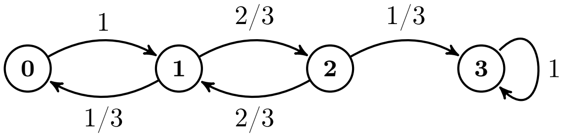


Figure 4.9: Transition graph

Solution 5.

1.

$C1 = \{1, 3\}$ *: transient, period=1*

$C2 = \{2\}$*: transient, period=1*

$C3 = \{4, 5\}$*: recurrent, period=2*

2. To find the average time of the first passage, we must solve the following system of equations:

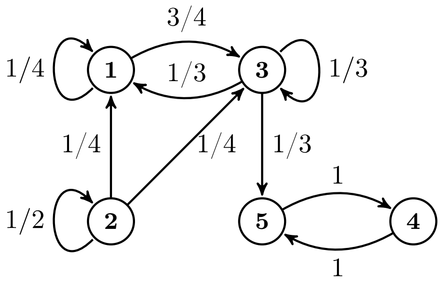


Figure 4.10: Transition graph

$$t_{i4} = 1 + \sum p_{ik} t_{k4}$$

such that $t_{44} = 0$ *and* $t_{54} = 1 + t_{44} = 1$

$$\begin{cases} t_{14} & = 1 + \frac{1}{4}t_4 + \frac{3}{4}t_{34} \\ t_{24} & = 1 + \frac{1}{4}t_{14} + \frac{1}{2}t_{24} + \frac{1}{4}t_{34} \\ t_{34} & = 1 + \frac{1}{3}t_{14} + \frac{1}{3}t_{34} + \frac{1}{3}t_{54} \end{cases}$$

So, we have: $t_{14} = \frac{20}{3}$ $\quad t_{24} = 8$ $\quad t_{34} = \frac{16}{3}$ $\quad t_{54} = 1$

Solution 6.

1.The classes of this DTMC are:

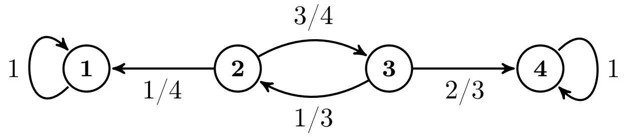


Figure 4.11: Transition graph

$C1 = \{1\}$*: absorbing,* $C2 = \{2,3\}$*: transient,* $C4 = \{4\}$*: absorbing*

2. This MC contains two absorbing states: 1 *and* 4*. The probability of absorption from state* 2 *is therefore the sum of the probabilities of absorption by* 1 *and by* 4 *starting from* 2*. We must solve the following two systems of equations such that* $a_1 = 1$*,* $a_4 = 0$*,* $b_1 = 0$ *and* $b_4 = 1$ *:*

$$\begin{cases} a_2 & = \frac{1}{4}a_1 + \frac{3}{4}a_3 \\ a_3 & = \frac{1}{3}a_2 + \frac{2}{3}a_4 \end{cases}$$

The solution is: $a_1 = 1$ $\quad a_2 = \frac{1}{3}$ $\quad a_3 = \frac{1}{9}$ $\quad a_4 = 0$

$$\begin{cases} b_2 & = \frac{1}{4}b_1 + \frac{3}{4}b_3 \\ b_3 & = \frac{1}{3}b_2 + \frac{2}{3}b_4 \end{cases}$$

The solution is: $b_1 = 0$ $\quad b_2 = \frac{2}{3}$ $\quad b_3 = \frac{8}{9}$ $\quad b_4 = 1$

$a_2 + b_2 = 1$*, therefore starting from* 2*, the chain will certainly be absorbed either by state* 1 *with probability* $1/3$ *or state* 4 *with probability* $2/3$*.*

Solution 7.

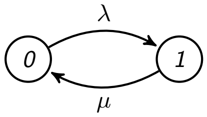
1- To calculate t_{21} *and* t_{31} *we use the total probability law with recursion such that* $t_{11} = 0$*:*

$$\begin{cases} t_{21} = 1 + \frac{1}{2}t_{21} + \frac{1}{2}t_{31} \\ t_{31} = 1 + \frac{1}{3}t_{11} + \frac{2}{3}t_{21} \end{cases}$$

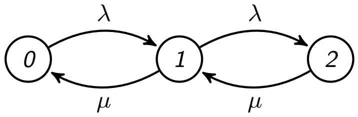
The solution is $t_{21} = 9$ *and* $t_{31} = 7$
2- $r_1 = 1 + \frac{1}{4}t_{11} + \frac{3}{4}t_{21} = 1 + \frac{3}{4} \times 9 = \frac{31}{4}$

Solution 8.
1. We have $S = \{F, M, E\}$, $\boldsymbol{\pi_0} = (1, 0, 0)$ *and* $\boldsymbol{\pi_2 = \pi_0}P^2 = (0.33, 0.41, 0.26)$. *The probability that the income of a grandson in a low-income family is high is: 0.26*
2. This MC is ergodic: 1) irreducible because all the states communicate, and 2) aperiodic because the period is equal to 1 ($p_{00} > 0$*). It therefore admits a limit stationary distribution which can be calculated by the system:* $\boldsymbol{\pi}\mathbf{P} = \boldsymbol{\pi}$ *and* $\boldsymbol{\pi}\mathbf{1} = 1$
which gives the solution: $\boldsymbol{\pi} = (\frac{12}{49}, \frac{23}{49}, \frac{2}{7})$
The percentage of the population with low income is : 24.48%

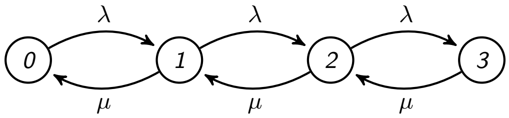
Solution 9.
The CTMC states represent the number of requests in the system.



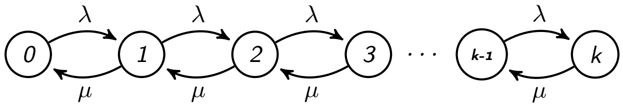
1. $[\lambda, , 0, \mu,]$



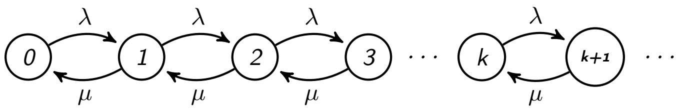
2. $[\lambda, , , \mu,]$



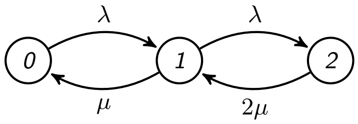
3. $[\lambda, , 2, \mu,]$



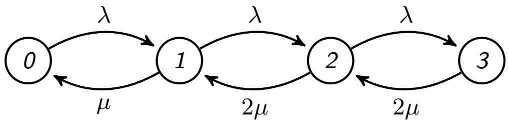
4. $[\lambda, , k-1, \mu,]$



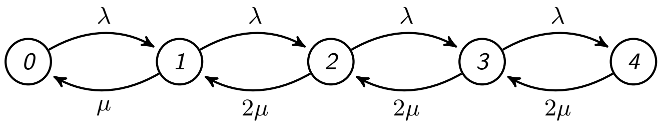
5. $[\lambda, , *, \mu,]$



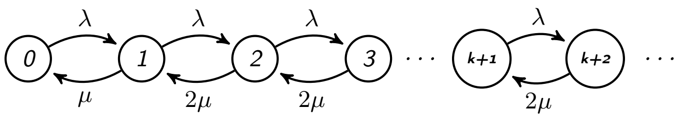
6. $[\lambda, , 0, \mu, 2]$



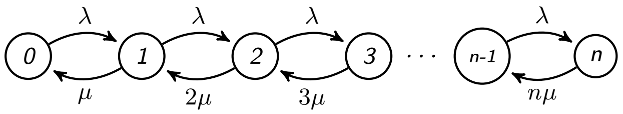
7. $[\lambda, , , \mu, 2]$



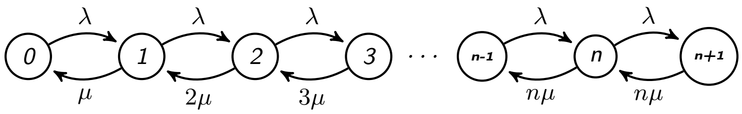
8. $[\lambda, , 2, \mu, 2]$



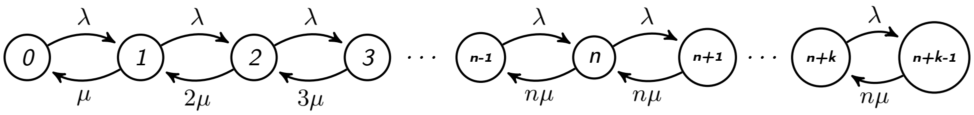
9. $[\lambda, , k, \mu, 2]$



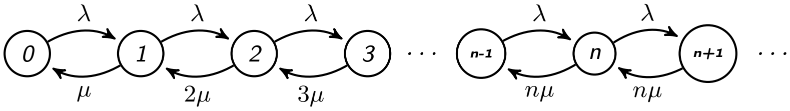
10. $[\lambda, , *, \mu, 2]$



11. $[\lambda, , 0, \mu, n]$

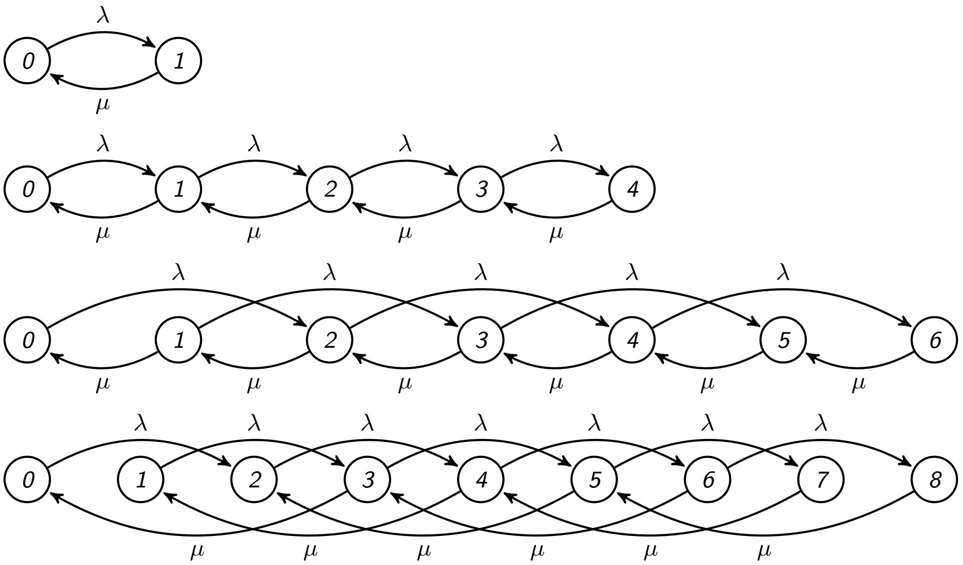


12. $[\lambda, , k, \mu, n]$



13. $[\lambda, , *, \mu, n]$

Solution 10.



Solution 11.
1- For $\mathbf{P}(t)$ to be a transition matrix, it must be stochastic. $\mathbf{A}$ and f must therefore satisfy the following conditions:
1.1- The sum of each row of $\mathbf{A}$ is equal to 0.
1.2- The elements of $\mathbf{P}(t)$ must be values between 0 and 1, therefore:

$$0 \leq d_{ij} + f(t)a_{ij} \leq 1 \implies -\frac{d_{ij}}{a_{ij}} \leq f(t) \leq \frac{(1-d_{ij})}{a_{ij}} \qquad \textit{if } a_{ij} \neq 0$$

2- From 1.1 and 1.2 we can deduce that $a = -b$, $c = -d$ therefore $-\frac{1}{2|a|} \leq \frac{e^{-t}}{2} \leq \frac{1}{2|a|}$ then $0 \leq e^{-t} \leq \frac{1}{|a|} = 1$ so $|a| = 1 \implies a = \pm 1$
In the same way we get $c = \pm 1$

$$\mathbf{A}_1 = \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}$$

For the other matrices we exchange the values of the same row ($\mathbf{A}_2, \mathbf{A}_3, \mathbf{A}_4$).
3- We have $-\frac{1}{2|a|} \leq f(t) \leq \frac{1}{2|a|} \implies |a| = 1$ and $-\frac{1}{3|c|} \leq f(t) \leq \frac{2}{3|c|}$
and $-\frac{2}{3|c|} \leq f(t) \leq \frac{1}{3|c|} \implies 0 < e^{-t} < \frac{2}{3|c|} = 1 \implies |c| = \frac{2}{3}$

$$\mathbf{A}_1 = \begin{bmatrix} 1 & -1 \\ \frac{2}{3} & -\frac{2}{3} \end{bmatrix}$$

For the other matrices we exchange the values of the same row (A_2, A_3, A_4).

Solution 12.
Let $(X_t)_{t>0}$ *be the number of infected individuals at time* t. $S = \{1, 2, \cdots, n\}$
If we have i *infected people then the time until one of the* $n-i$ *people to become infected is the min of* $(n-i) * i$ *r.vs of rate* λ.
$q_{i,i+1} = (n-i) \times i \times \lambda$ *for* $i = 1, 2, ..., n-i$.

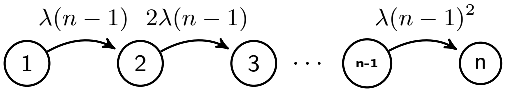


Figure 4.12: Transition graph

Solution 13.
Let $(X_t)_{t>0}$ *be the number of drink units in the vending machine at time* t. $S = \{0, 1, ..., N\}$.
The Markov property is satisfied by the fact that the Poisson processes are memoryless, i.e. at any time, the time elapsed since the arrival of the last event, does not affect the arrival of future events.
$q_{i,i-1} = \lambda$, $q_{i,j} = 0$ *for* $j \neq i-1$ *and* $q_{0,N} = \mu$, $q_{0,j} = 0$ *for* $j \neq N$.

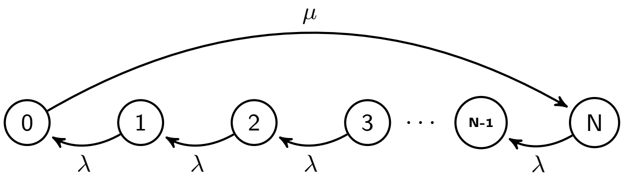


Figure 4.13: Transition graph

Solution 14. *Let* $(X_t)_{t>0}$ *be the number of units under repair at time* t. $S = \{0, 1, ..., n\}$.
The minimum of i *exponential r.vs of rate* μ *follows an exponential distribution of rate* $i \times \mu$.
If at any given time t, *there are* i *units under repair, then* $q_{i,i-1} = i \times \mu$. *and* $q_{i,i+1} = \lambda$ *for* $0 \leq i \leq n-1$.

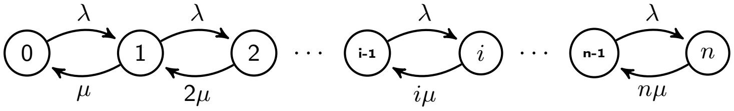


Figure 4.14: Transition graph

Solution 15.
1. The generator matrix $\mathbf{Q}$ *is defined by:*

$$q_{ij} = \begin{cases} \lambda_i p_{ij} & \text{if } i \neq j \\ -\lambda_i & \text{if } i = j \end{cases}$$

$$\mathbf{Q} = \begin{bmatrix} -1 & 1/3 & 1/3 & 1/3 \\ 0 & -2 & 2 & 0 \\ 0 & 0 & -1 & 1 \\ 1 & 0 & 0 & -1 \end{bmatrix}$$

2. The limit distribution is obtained by solving the following system of equations:

$$\boldsymbol{\pi}\mathbf{Q} = \mathbf{0} \quad \text{and} \quad \boldsymbol{\pi}\mathbf{1} = 1$$

We get:

$$\boldsymbol{\pi} = (\frac{6}{17}, \frac{1}{17}, \frac{4}{17}, \frac{6}{17})$$

3. We calculate the first hitting time of state 4 using the recursive method:

$$\mu_i = \frac{1}{\lambda_i} + \sum_{j \neq i,k} p_{ij}\mu_j \quad for \quad i \neq 4.$$

$$\begin{cases} \mu_1 = 1 + \frac{1}{3}\mu_2 + \frac{1}{3}\mu_3 \\ \mu_2 = \frac{1}{2} + \mu_3 \\ \mu_3 = 1 \end{cases}$$

The solution is : $(\mu_1, \mu_2, \mu_3) = (\frac{11}{6}, \frac{3}{2}, 1)$

Chapter 5

Application to queuing models

5.1 Introduction

Queuing is a class of models for studying systems that offer services. The creation of a queue begins when the demand for the service exceeds the capacity of the *servers* (entities offering the *service*). Given the non-deterministic (random) aspect of the arrival of requests, decision-makers often face the problem of predicting the time that will be necessary to serve the requests, which is an information that makes it possible to decide on the required number and/or frequency of servers. Being able to make these decisions helps, on the one hand, to avoid offering an excess of service, something that generates unnecessary costs (costs of unoccupied servers); and on the other hand, it avoids too long waits for the services which generates the loss of *requests* (often *clients*) or sometimes unnecessary social costs. The objective is therefore to find the right compromise between the cost of service and the cost of waiting.

Using queue models helps calculate some metrics of the system's *performance*, such as the client's average waiting time used by decision-makers to find this trade-off.

In this class of models the system to be studied is called *station* where clients (requests) arrive to receive a service. If the servers are busy, the clients wait their turn to be served; and when served, they leave the system.

A common example of this type of system is the emergency department in a hospital, where patients (clients) arrive to receive treatment (the service) by a doctor (the server). When the doctor is treating a patient, the patients who come must wait their turn (queue). The hospital director must decide of the number of doctors to be put into service in order to limit the waiting time for patients while limiting the idle hours of the doctors. Figure 5.1 schematizes a queuing system. The same description applies to many other systems such as administrative, commercial, transport, IT service systems, etc.

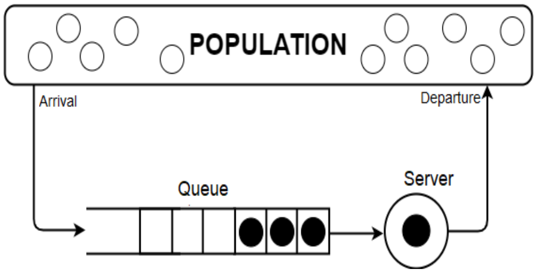


Figure 5.1: An elementary queuing system

5.2 Components of a queuing system

The elements that make up a queuing system are:

- *The queue*: can be finite or infinite. When a customer tries to enter the station whose queue is full and limited, he is considered lost.

- *Client*: is the entity that receives the service offered by the system.
- *Server*: when speaking of a single server, there is one server that processes clients one after the other. When speaking of multiservers, several clients are served at the same time by several servers.
- *Discipline of service*: this is the order in which customers in a queue will be served. The default discipline is FIFO (First In First Out: first come first served). If another discipline is used, it must be specified (Round Robin, LIFO, ...)
- *Arrival process*: describes the time between two successive arrivals of customers. This time can be deterministic (it is necessary to give its value) or random (it is necessary to specify the law and the parameters that make it possible to define it)
- *Service process*: describes the time it takes for a server to process a client. (same remark as the arrival process)

5.3 Kendall notation A/B/C/K/P/D

The different models of queuing systems can be classified and characterized by the properties of the systems they model and the type and context of the application. In 1958, Kendall proposed a notation to represent each type of these models which was subsequently standardized. The current notation scheme has 6 classification references and has the following form A/B/C/K/P/D (P and D are often omitted) such that:

- A and B: arrival and service processes. Can take values:
  - M: *Exponential* distribution (Markovian)
  - D: Degenerate distribution (taking a constant value).
  - E_k: *Erlang* distribution (of parameter k).
  - G: General distribution.
- C: Number of servers.
- K: System capacity (Queue and currently served customers omitted if infinite).
- P: Population (omitted if infinite).
- D: Service policy (FIFO by default).

The model **M/M/s** for example considers that the inter-arrival time and the service time follow exponential distribution, the number of servers is s and the size of the queue is infinite. The model **M/G/1** has an exponential inter-arrival time, but there is no restriction on the service time distribution, only the expectation and the variance of this distribution are necessary to know (or to estimate), there is only one server and the queue is of infinite size. In the case of **M/D/s**, the inter-arrival time is exponentially distributed, the service time is constant and the system has s servers.

5.4 Station study

The study of a given station makes it possible to determine its performances. In order to find them analytically, we use the following notations:

λ : *Average arrival rate (average number of arrivals per unit of time)*

μ : *Average service rate (average number of customers completing the service per unit of time)*

π_n : *Probability of having n clients in the system.*

L : *Average number of customers in the station (waiting customer + customers being served).*

L_q : *Average number of customers in the queue (excluding customers being served).*

W : *Average customer response time (between entering and leaving the station).*

W_q : *Average waiting time in the queue.*

ρ : *Utilization factor. i.e the fraction of the average time during which individual servers are busy.*

$$\rho = \frac{B}{B+I} = \frac{\lambda}{s\mu}$$

B is the length of time the server is occupied during the operating period.
I is the length of time the server is idle.
If, in a unit of time, each server can serve μ clients and the system receives λ clients, then each of the servers will have an average occupancy rate equal to $\frac{\lambda/s \text{ clients}}{\mu \text{ clients}} = \rho$.

5.4.1 Little law

Consider the following processes:
$L(t)$ the number of clients in the system at time t.
$A(t)$ the number of clients who have arrived in the system at time t.
$D(t)$ the number of clients who left the system (after receiving the service) at time t.
W_k the total time spent by client k in the system.

$$L(t) = A(t) - D(t)$$

$$\lambda = \lim_{t\to\infty} \frac{A(t)}{t}$$

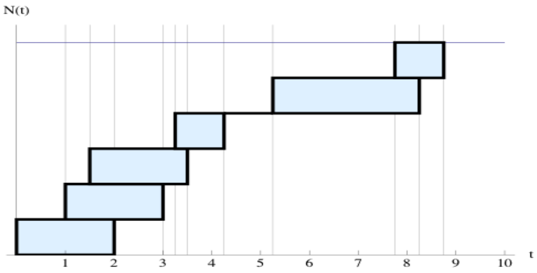


Figure 5.2: Little law

$H(t)$ is the area between the curve $A(t)$ and that of $D(t)$. It is equal to the sum of the total time spent by the clients who arrived before time t.

$$H(t) = \int_0^t L(s)ds = \sum_{i=1}^{A(t)} W_i$$

Dividing by t we get:

$$\frac{1}{t}\int_0^t L(s)ds = \frac{1}{t}\sum_{i=1}^{A(t)} W_i = \frac{A(t)}{t} \times \frac{\sum_{i=1}^{A(t)} W_i}{A(t)}$$

$$\lim_{t\to\infty} \frac{1}{t}\int_0^t L(s)ds = \lim_{t\to\infty} \left(\frac{A(t)}{t} \times \frac{\sum_{i=1}^{A(t)} W_i}{A(t)}\right)$$

When these limits exist, we get

$$L = \lambda \times W$$

In the same way we can show that:

$$L_q = \lambda \times W_q$$

$$W = W_q + \frac{1}{\mu}$$

($\frac{1}{\mu}$ is the average service time)

To know the performances L, L_q, W and W_q, it is necessary and sufficient to calculate π_n for $n \in \mathbb{N}$: the probability of having n customers in the station.

Relationships between performance indicators.

- The average number of clients in the system:

$$L = \sum_{n=0}^{\infty} n\pi_n$$

- The average number of clients in the queue (there are s servers in the system) :

$$L_q = \sum_{n=s}^{\infty}(n-s)\pi_n = \sum_{n=s}^{\infty} n\pi_n - s\sum_{n=s}^{\infty}\pi_n = L - (1 - s\sum_{n=0}^{s-1}\pi_n)$$

- The average response time:

$$W = \frac{L}{\lambda}$$

- The average waiting time in the queue (response time - service time):

$$W_q = W - \frac{1}{\mu}$$

Let’s code!

```
#Code501.py

from math import factorial

# Queueing Class
class Queueing(object):

    # constructor
    def __init__(self, model=’MM1’,  A=’M’ ,B=’M’ ,C=1, K=-1, P=None, D=’FIFO’, laws =None):
        self.model =  model

        self.s, self.k = C, K
        self.lamda, self.mu = A[’params’][’lambda’], B[’params’][’mu’]
        self.rho = 1.0 * self.lamda/(self.s*self.mu)

        self.p0 = laws[self.model][’p0’](rho1=self.rho, s=self.s , k=self.k)
        self.pk = self.pn(self.k) if self.k > 0 else 0;
        self.lamdae = (1 - ( 0 if self.k == -1 else self.pk)) * self.lamda
        self.rhoe = 1.0 * self.lamdae/(self.s* self.mu)

        #
        self.Lq = laws[self.model][’Lq’](rho1=self.rhoe, s=self.s , k=self.k, p0=self.p0)
        self.L  = self.Lq + self.s * self.rhoe
        self.W = lambda q : (self.L if q == ’’ else self.Lq)/self.lamdae

    # isErgodic
    def isErgodic(self):
        return self.rho <1
    # getStationaryProb
    def pn(self,n):
        pn0 = lambda n: self.s**min(self.s,n)/factorial(min(self.s,n)) * self.rho**n * self.p0
        if n == 0 : return self.p0
        if self.k != -1 and n > self.k  : return 0
        return pn0(n)

    # show perforamnces
    def test(self):
        print('==================' + self.model + ' =======================')
        print("p0* :{:.5f}".format(self.p0))
        print("p1* :{:.5f}".format(self.pn(1)))
        print("Lq  :{:.2f}".format(self.Lq))
        print("L   :{:.2f}".format(self.L))
        print("W   :{:.2f}".format(self.W('')))
        print("Wq  :{:.2f}".format(self.W('q')))
```

5.4.2 M/M/1 system

In this system, arrivals follow Poisson distribution with rate λ (average number of clients arriving during a unit of time) and the service is exponential of rate μ (average number of customers served during a unit of time, so $\frac{1}{\mu}$ is the average service time of a client). There is only one server and the queue size is unlimited. It is assumed that the discipline is *FIFO*.

If $(X_t)_{t\geq 0}$ represents the number of clients in the system at time t, then $S = \{0, 1, 2, ...\}$. We move from state i to $i+1$ if a customer arrives, we move from state i to $i-1$ if a customer is served. This is described by the following graph. It corresponds to a CTMC whose generator matrix is:

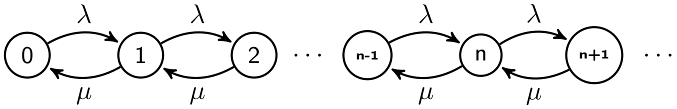


Figure 5.3: M/M/1 system

$$
\mathbf{Q} = \begin{array}{c|ccccccc}
 & \textit{0} & \textit{1} & \textit{2} & \textit{3} & \dots & \textit{i} & \dots \\
\textit{0} & -\lambda & \lambda & 0 & 0 & \dots & 0 & \dots \\
\textit{1} & \mu & -(\lambda+\mu) & \lambda & 0 & \dots & 0 & \dots \\
\textit{2} & 0 & \mu & -(\lambda+\mu) & \lambda & \dots & 0 & \dots \\
\vdots & & & & & & & \\
\textit{i} & 0 & 0 & 0 & \mu & -(\lambda+\mu) & \lambda & \dots \\
\vdots & & & & & & &
\end{array}
$$

We are looking for π_n, the probability of having n clients in the system at the steady state. We know that in this state:

$$
\begin{cases}
\boldsymbol{\pi}\mathbf{Q} = \mathbf{0} \\
\boldsymbol{\pi}\mathbf{1} = 1
\end{cases}
$$

Which gives the system of equations:

$$
\begin{cases}
\mu\pi_1 - \lambda\pi_0 = 0 \\
\lambda\pi_{i-1} + \mu\pi_{i+1} - (\lambda+\mu)\pi_i = 0 \quad \text{for } i \geq 1 \\
\sum_{i=0}^{\infty} \pi_i = 1
\end{cases}
$$

The solution of this system gives:

$$\pi_1 = \frac{\lambda}{\mu}\pi_0$$

$$\pi_{i+1} - \frac{\lambda}{\mu}\pi_i = \pi_i - \frac{\lambda}{\mu}\pi_{i-1} = \pi_1 - \frac{\lambda}{\mu}\pi_0 = 0 \quad \text{for } i \geq 1$$

so

$$\pi_{i+1} = \frac{\lambda}{\mu}\pi_i \quad \text{for } i \geq 1$$

We deduce that :

$$\pi_n = \rho^n \times \pi_0$$

such that $\rho = \frac{\lambda}{\mu}$. Which can be verified by induction.

To calculate π_0 we use the relation $\sum_{i=0}^{\infty} \pi_i = 1$. We get:

$$\pi_0 = \frac{1}{1 + \rho + \rho^2 + \rho^3 + ...}$$

The expression in the denominator is a geometric progression of reason ρ. It is convergent if $\rho < 1$ and equals to : $1/(1-\rho)$. The condition $\rho < 1$ is essential for the stability of the system, it is called the ergodicity condition of the system. $\rho < 1$ means that the arrival rate is lower than the departure rate, otherwise there will be an overflow of the number of customers in the system. We deduce that:

$$\pi_0 = 1 - \rho \quad \text{and} \quad \pi_n = \rho^n(1-\rho) = \rho^n\pi_0 \qquad \text{for } n \geq 1$$

Performances.

- *The average number of customers in the system:*

$$\begin{aligned} L = \sum_{n=0}^{\infty} n\pi_n = \sum_{n=0}^{\infty} n(1-\rho)\rho^n &= \rho(1-\rho)\sum_{n=1}^{\infty} n\rho^{n-1} \\ = \rho(1-\rho)\left(\sum_{n=1}^{\infty}\rho^n\right)' &= \rho(1-\rho)\frac{1}{(1-\rho)^2} \end{aligned}$$

  So:

$$L = \frac{\rho}{1-\rho} = \frac{\lambda}{\mu - \lambda} = \frac{\rho\pi_0}{(1-\rho)^2}$$

- *Average number of customers waiting in the queue:*
  L_q= average number of customers in the station - the average number of customers being served.

$$L_q = L - (1 - \pi_0) = L - \rho$$

- *The average response time:*

$$W = \frac{L}{\lambda} = \frac{1}{\mu - \lambda}$$

- *The average waiting time in the queue (response time - service time):*

$$W_q = W - \frac{1}{\mu}$$

Example:
For a M/M/1 system with parameters $\lambda = 4c/h$ and $\mu = 6c/h$, we have:
- Average service time : $1/\mu = 1/6 = 10$ min per client
- Server utilization rate : $\rho = \lambda/\mu = 4/6 = 2/3 = 0.666$
- Probability of having 0 clients in the system : $\pi_0 = (1 - \rho) = 1/3 = 0.333$
- Probability that the server is busy : $1 - \pi_0 = \rho = 0.666$
- Probability of having only one client in the queue : $\pi_1 = \rho\pi_0 = 2/3 \times 1/3 = 2/9 = 0.2222$
- Probability of having 0 clients in the queue : $\pi_0 + \pi_1 = 0.333 + 0.222 = 0.555$
- Average number of clients in the system : $L = \rho/(1 - \rho) = 2/3 \times 3/1 = 2$
- Average number of clients in the queue : $L_q = L - \rho = 2 - 0.666 = 1.333$
- Average waiting time in the system : $W = L/\lambda = 2/4 = 1/2h = 30$ min
- Average waiting time in the queue before service : $W_q = 0.333 = 20$ min

The parameters and performance indicators of this system indicate that :
- 66.6% of time the server is busy.
- 33.3% of time the system is idle.
- 22.2% of time one client is in the system.
- 55.5% of time the queue is empty.
- 4.3% of time there are 5 clients in the system.
- In average 2 clients in the system.
- In average 1.333 clients are in the queue.
- Each client waits in average 30 min.
- Each client waits in the queue in average 20 min.

For a M/M/1 system with an arrival rate $\lambda = 8c/h$:
For the utilization rate to be greater than 80%, its service rate must be: $\mu = \lambda/\rho = 8/0.8 = 10c/h$.

Let's code!

```
#Code502.py

from Code501 import Queueing

# getMM1
def getMM1(lamda, mu):
    #
    laws ={
    ’MM1’ : {
        ’p0’ : lambda **p: (1 - p[’rho1’]),                           # p0 = 1 - rho,
        ’Lq’ : lambda **p: p[’rho1’] / (1 - p[’rho1’]) - p[’rho1’],   # Lq = L - rho
        },
    }

    #==========================================================================
    # Tests
    qs = Queueing(
            model = ’MM1’,
            A ={’D’:’Pois’ , ’params’: { ’lambda’: lamda}},
            B ={’D’:’Expo’ , ’params’: { ’mu’: mu}},
            laws = laws)
    qs.test()


if __name__ == "__main__":  getMM1(4,6)
# ==================MM1 =======================
# p0* :0.33333
# p1* :0.22222
# Lq  :1.33
# L   :2.00
# W   :0.50
# Wq  :0.33
```

5.4.3 M/M/1/K system

In this system the arrivals follow Poisson distribution of rate λ (average number of clients arriving during a unit of time) and the service is exponential of rate μ (average number of customers served during a unit of time, therefore $\frac{1}{\mu}$ is the average customer service time). There is only one server and the queue size is limited to $K-1$ clients. We assume that the discipline is *FIFO*.

In this system, the arrival rate λ is significant during the period when the queue size is less than $K-1$, after this step all the clients that arrive at the system will be rejected, the system does not see these clients. The rate of arrivals perceived by an observer in the system is less than λ and equal to the real rate multiplied by the fraction of time when the system is not in its maximum capacity (K). This is called the effective rate and it will be noted λ_e .

If $(X_t)_{t\geq 0}$ represents the number of clients in the system at time t, then $S = \{0, 1, 2, ..., K\}$. We move from state i to $i+1$ if a client arrives, we move from state i to $i-1$ if a client is served. This is described by the following graph. It corresponds to a CTMC whose generator matrix is as follows:

$$\mathbf{Q} = \begin{array}{cccccccc} & \textbf{\textit{0}} & \textbf{\textit{1}} & \textbf{\textit{2}} & \textbf{\textit{3}} & \ldots & \textbf{\textit{K-2}} & \textbf{\textit{K-1}} & \textbf{\textit{K}} \\ \textbf{\textit{0}} & -\lambda & \lambda & 0 & 0 & \ldots & 0 & 0 & 0 \\ \textbf{\textit{1}} & \mu & -(\lambda+\mu) & \lambda & 0 & \ldots & 0 & 0 & 0 \\ \textbf{\textit{2}} & 0 & \mu & -(\lambda+\mu) & \lambda & \ldots & 0 & 0 & 0 \\ \vdots & & & & & & & & \\ \textbf{\textit{K-1}} & 0 & 0 & 0 & 0 & \ldots & \mu & -(\lambda+\mu) & \lambda \\ \textbf{\textit{K}} & 0 & 0 & 0 & 0 & \ldots & 0 & \mu & -\mu \end{array}$$

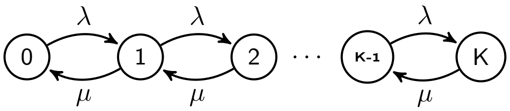


Figure 5.4: M/M/1/K system

We are looking for π_n, the probability of having n clients in the steady state system. The balance equations of this system are :

$$\begin{cases} \mu\pi_1 - \lambda\pi_0 = 0 \\ \lambda\pi_{i-1} + \mu\pi_{i+1} - (\lambda+\mu)\pi_i = 0 \quad \text{for } 2 \leq i \leq K-1 \\ \mu\pi_K - \lambda\pi_{K-1} = 0 \\ \sum_{i=0}^{\infty} \pi_i = 1 \end{cases}$$

The solution of this system gives (in the same way as M/M/1):

$$\pi_n = \rho^n \times \pi_0 \quad \text{ for } 1 \leq i \leq K$$

To calculate π_0 we use the relation $\sum_{i=0}^{\infty} \pi_i = 1$. We get:

$$\pi_0 = \frac{1}{1+\rho+\rho^2+\rho^3+...+\rho^K} = \frac{1-\rho}{1-\rho^{K+1}}$$

The denominator expression is a finite geometric progression of K terms and reason ρ .
We deduce that:

$$\pi_n = \rho^n \frac{1-\rho}{1-\rho^{K+1}} \qquad \text{for } 0 \leq n \leq K$$

Performances.

As explained above the system perceives another arrival rate: λ_e which will be used to determine the performance factors instead of λ. λ_e is equal to λ times the probability that the system is not in state K (π_K). So the *effective utilization rate* λ_e in the system is :

$$\lambda_e = \lambda(1-\pi_K).$$

$$\rho_e = \frac{\lambda_e}{\mu}$$

- *The average number of customers in the system:*

$$\begin{aligned} L = \sum_{n=0}^{K} n\pi_n = \sum_{n=0}^{K} n\frac{1-\rho}{1-\rho^{K+1}}\rho^n = \rho\frac{1-\rho}{1-\rho^{K+1}}\sum_{n=1}^{K} n\rho^{n-1} \\ = \rho\frac{1-\rho}{1-\rho^{K+1}}\left(\sum_{n=1}^{K}\rho^n\right)' = \rho\frac{1-\rho}{1-\rho^{K+1}}\left(\frac{1-\rho^{K+1}}{1-\rho}\right)' \\ = \rho\frac{1-\rho}{1-\rho^{K+1}}\left[\frac{-(1-\rho)(K+1)\rho^K+1-\rho^{K+1}}{(1-\rho)^2}\right] \end{aligned}$$

So:

$$L = \rho\frac{1-\rho}{(1-\rho)^2}\left[\frac{-(1-\rho)(K+1)\rho^K+1-\rho^{K+1}}{1-\rho^{K+1}}\right] = \frac{\rho}{(1-\rho)}\left[-\pi_0(K+1)\rho^K+1\right]$$

- *Average number of customers waiting in the queue:*
  $L_q =$ average number of customers in the station - the average number of customers being served.
  $$L_q = L - \rho_e$$

- *The average response time:*
  $$W = \frac{L}{\lambda_e} = \frac{1}{\mu-\lambda_e}$$

- *The average waiting time in the queue (response time - service time):*
  $$W_q = W - \frac{1}{\mu}$$

Example:
For a M/M/1/12 system with parameters $\lambda = 5c/h$ and $\mu = 10c/h$, we have :
- Average service time : $1/\mu = 1/10 = 6$ min per client
- Server utilization rate : $\rho = \lambda/\mu = 5/10 = 0.5$
- Probability of having 0 customers in the system : $\pi_0 = 0.5$
- Probability that the server is busy : $1-\pi_0 = 0.5$
- Probability of having only one client in the queue : $\pi_1 = 0.25$
- Probability of having no clients in the queue : $\pi_0+\pi_1 = 0.5+0.25 = 0.75$
- Average number of clients in the system : $L = 1$
- Average number of clients in the queue : $L_q = 0.5$
- Average waiting time in the system : $W = 0.2h = 12$ min
- Average waiting time in queue before service : $W_q = 0.10 = 6$ min

Particular case: $\rho = 1$

$$\pi_n = \pi_0 \quad \text{for } 1 \le i \le K$$

To calculate π_0 we use the relation $\sum_{i=0}^{K}\pi_0 = (K+1)\pi_0 = 1$. So:

$$\pi_0 = \frac{1}{K+1}$$

Let’s code!

```
#Code503.py

from Code501 import Queueing

# p0
def getP0(**p):
    rho, k = p[’rho1’], p[’k’]
    return (1 - rho)/(1 - rho**(k+1))

# Lq
def getLq(**p):
    rho, k, p0 = p[’rho1’], p[’k’], p[’p0’]    #
    return rho/(1-rho) * ( 1 - p0 * (k*rho**k + 1))

#
laws ={ ’MM1K’ : { ’p0’ : getP0, ’Lq’ : getLq, }}

#==========================================================================
# Tests
def getMM1K(mu, lamda, K):
    qs = Queueing(
            model = ’MM1K’,
            A ={’D’:’Pois’ , ’params’: { ’lambda’: lamda}},
            B ={’D’:’Expo’ , ’params’: { ’mu’: mu}},
            C = 1,
            K =  K,
            laws=laws)
    qs.test()

if __name__ == "__main__": getMM1K(10, 5, 12)
# ==================MMS =======================
# p0* :0.50006
# p1* :0.25003
# Lq  :0.50
# L   :1.00
# W   :0.20
# Wq  :0.10
```

5.4.4 M/M/s system

When the number of servers s is greater than 1 (multi-servers), the model diagram becomes as follows:

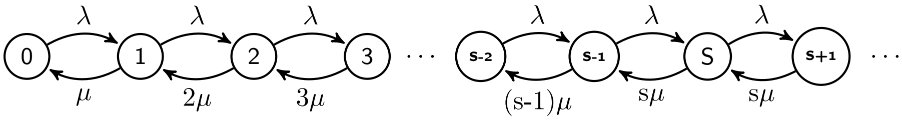


Figure 5.5: M/M/s system

In this diagram, the exit rate from state i is $i\mu$ for $i < s$ and $s\mu$ for $i \geq s$. This is because the transition from state i to state $i-1$ occurs when the first of the busy servers terminates its service. i.e, the r.v T of the time of passage from state i to state $i-1$ is the minimum of the r.vs of the service time of all the i busy servers and which (see exercise 6 of Chapter 2) follows an exponential distribution of rate $i\mu$. From state s, all servers are busy and therefore the service rate is $s\mu$.
The condition for the queuing system of this model to admit a stationary state (ergodicity

condition) is that $\rho = \frac{\lambda}{s\mu} < 1$

$$\mathbf{Q} = \begin{array}{cccccccccc}
 & \mathit{0} & \mathit{1} & \mathit{2} & \mathit{3} & \dots & \mathit{s\text{-}1} & \mathit{s} & \mathit{s+1} & \dots \\
\mathit{0} & -\lambda & \lambda & 0 & 0 & \dots & 0 & 0 & 0 & \dots \\
\mathit{1} & \mu & -(\lambda+\mu) & \lambda & 0 & \dots & 0 & 0 & 0 & \dots \\
\mathit{2} & 0 & 2\mu & -(\lambda+2\mu) & \lambda & \dots & 0 & 0 & 0 & \dots \\
\vdots & & & & & & & & & \\
\mathit{s\text{-}1} & 0 & 0 & 0 & 0 & \cdots & -(\lambda+(s-1)\mu) & \lambda & 0 & \dots \\
\mathit{s} & 0 & 0 & 0 & 0 & \cdots & s\mu & -(\lambda+s\mu) & \lambda & \dots \\
\mathit{s+1} & 0 & 0 & 0 & 0 & \cdots & .. & s\mu & -(\lambda+s\mu) & \dots \\
\vdots & & & & & & & & &
\end{array}$$

The balance equations of this system are :

$$\begin{cases} \mu\pi_1 - \lambda\pi_0 = 0 \\ \lambda\pi_{i-1} + min(s,i+1)\mu\pi_{i+1} - (\lambda + min(s,i)\mu)\pi_i = 0 \quad \text{for } i \geq 1 \\ \sum_{i=0}^{\infty} \pi_i = 1 \end{cases}$$

so

$$\begin{cases} \pi_1 = \frac{\lambda}{\mu}\pi_0 \\ \pi_{i+1} = \frac{(\lambda+min(s,i)\mu)\pi_i - \lambda\pi_{i-1}}{min(s,i+1)\mu} \quad \text{for } i \geq 1 \\ \sum_{i=0}^{\infty} \pi_i = 1 \end{cases}$$

If $i > s$, this part of the diagram is similar to M/M/1, and we apply the same analysis method with π_s as initial state.

$$\pi_n = \rho^{(n-s)} \times \pi_s$$

If $i \leq s$, we have :

$$\pi_{i+1} = \frac{(\lambda+i)\mu\pi_i - \lambda\pi_{i-1}}{(i+1)\mu} \quad \text{for } i \geq 1$$

$$\pi_{i+1} = \frac{\lambda}{(i+1)\mu}\pi_i + \frac{i}{(i+1)}\pi_i - \frac{\lambda}{(i+1)\mu}\pi_{i-1} \quad \text{for } i \geq 1$$

$$\pi_{i+1} - \frac{\lambda}{(i+1)\mu}\pi_i = \frac{i}{(i+1)}\left(\pi_i - \frac{\lambda}{i\mu}\pi_{i-1}\right) \quad \text{for } i \geq 1$$

By substituting the other terms :

$$\pi_{i+1} - \frac{\lambda}{(i+1)\mu}\pi_i = \frac{1}{(i+1)}\left(\pi_1 - \frac{\lambda}{\mu}\pi_0\right) = 0 \quad \text{for } i \geq 1$$

From the first balance equation, we have:

$$\pi_{i+1} = \frac{\lambda}{(i+1)\mu}\pi_i \quad \text{for } i \geq 1$$

so:

$$\pi_{i+1} = \frac{1}{(i+1)!}\left(\frac{\lambda}{\mu}\right)^{i+1}\pi_0 \quad \text{for } i \geq 1$$

$$\pi_n = \frac{1}{n!}\left(\frac{\lambda}{\mu}\right)^n\pi_0 = \frac{s^n}{n!}\rho^n\pi_0 \quad \text{for } 1 \leq n \leq s$$

$$\pi_n = \frac{s^{min(n,s)}}{min(n,s)!}\rho^n\pi_0 \quad \text{for } 1 \leq n \leq s$$

To calculate π_0 we use the relation $\sum_{i=0}^{\infty} \pi_i = 1$. We get:

$$\sum_{n=s+1}^{\infty} \rho^{(n-s)} \pi_s + \sum_{n=0}^{s} \frac{s^n}{n!} \rho^n \pi_0 = 1$$

$$\pi_s \sum_{n=s}^{\infty} \rho^{(n-s)} + \pi_0 \sum_{n=0}^{s-1} \frac{s^n}{n!} \rho^n = 1$$

We can deduce that:

$$\pi_0 = \frac{1}{\sum_{n=0}^{s-1} \frac{s^n}{n!} \rho^n + \frac{s^s}{s!} \rho^s \frac{1}{1-\rho}}$$

Performances.

- *Average number of customers waiting in the queue:*
  $L_q =$ average number of clients in the station - the average number of clients being served. Queue creation begins from the arrival of the $(s+1)^{th}$ client who will find all the servers busy so he joins the queue.

$$L_q = \sum_{n=s}^{\infty} (n-s)\pi_n = \sum_{n=s}^{\infty} (n-s) \frac{s^s}{s!} \rho^n \pi_0$$

$$= \frac{s^s}{s!} \pi_0 \rho^s \sum_{n=s}^{\infty} (n-s) \rho^{n-s} = \frac{s^s}{s!} \pi_0 \rho^{s+1} \sum_{k=0}^{\infty} k \rho^{k-1}$$

So:

$$L_q = \frac{s^s}{s!} \pi_0 \rho^{s+1} \frac{1}{(1-\rho)^2}$$

Example:
For a M/M/4 system with parameters $\lambda = 10c/h$ and $\mu = 5c/h$, we have :
- Average service time : $1/\mu = 1/5 = 0.2h/c = 12$ min per client
- Server utilization rate : $\rho = \lambda / s\mu = \frac{10}{4 \times 5} = 1/2 = 0.5$
- Probability of having 0 clients in the system : $\pi_0 = 0.13043$
- Probability that the server is busy : $1 - \pi_0 = 0.86957$
- Probability of having one client in the queue : $\pi_1 = 0.26087$
- Average number of clients in the system : $L = 2.17$
- Average number of clients in the queue : $L_q = 0.17$
- Average waiting time in the system : $W = 0.22$
- Average waiting time in the queue before service : $W_q = 0.02$

Let’s code!

```
#Code504.py

from Code501 import Queueing
from math import factorial
#
fsum01 = lambda rho,s, n : 1 if n == 0 else fsum01(rho,s, n-1) + (s**n/factorial(n))*rho**n

# p0 :
def getP0(**p):
    rho,s = p[’rho1’], p[’s’]
    return 1/( fsum01(rho,s , s-1) + (s**s/factorial(s)) * rho**s * (1/(1 - rho)))

# Lq
def getLq(**p):
    rho, s, p0 = p[’rho1’], p[’s’], p[’p0’]
    return (s**s /factorial(s)) * p0 * rho**(s+1)/(1-rho)**2

#
laws ={ ’MMS’ : { ’p0’ : getP0,  ’Lq’ : getLq }}

#==========================================================================
# Tests
def getMMS(mu, lamda, S):
    qs = Queueing(
            model = ’MMS’,
            A ={’D’:’Pois’ , ’params’: { ’lambda’: lamda}},
            B ={’D’:’Expo’ , ’params’: { ’mu’: mu}},
            C = S,
            laws=laws)
    qs.test()

if __name__ == "__main__":  getMMS(5,10,4)
# =================MMS =======================
# p0* :0.13043
# p1* :0.26087
# Lq  :0.17
# L   :2.17
# W   :0.22
# Wq  :0.02
```

5.4.5 M/M/s/K system

$$\mathbf{Q} = \begin{array}{c|cccccccccc}
 & \mathbf{0} & \mathbf{1} & \dots & \mathbf{s\text{-}1} & \mathbf{s} & \mathbf{s+1} & \dots & \mathbf{K\text{-}1} & \mathbf{K} \\
\mathbf{0} & -\lambda & \lambda & \cdots & 0 & \dots & 0 & \cdots & 0 & 0 \\
\mathbf{1} & \mu & -(\lambda+\mu) & \cdots & 0 & \dots & 0 & \cdots & 0 & 0 \\
\mathbf{2} & 0 & 2\mu & \cdots & 0 & \dots & 0 & \cdots & 0 & 0 \\
\vdots & & & & & & & & & \\
\mathbf{s\text{-}1} & 0 & 0 & \cdots & -(\lambda+(s-1)\mu) & \lambda & 0 & \cdots & 0 & 0 \\
\mathbf{s} & 0 & 0 & \cdots & s\mu & -(\lambda+s\mu) & \lambda & \cdots & 0 & 0 \\
\mathbf{s+1} & 0 & 0 & \cdots & 0 & s\mu & -(\lambda+s\mu) & \cdots & 0 & 0 \\
\vdots & & & & & & & & & \\
\mathbf{K\text{-}1} & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & -(\lambda+s\mu) & \lambda \\
\mathbf{K} & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & s\mu & -s\mu
\end{array}$$

When the number of servers s is greater than 1 (multi-servers) and the queue size is limited, the model diagram is as follows:

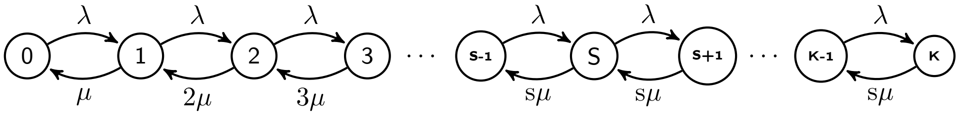


Figure 5.6: M/M/s/K system

In this diagram, the exit rate from state i is $i\mu$ for $i < s$ and $s\mu$ for $i \geq s$. This comes from the fact that the passage from state i to state $i-1$ when the first of the busy servers ends its service, (i.e the r.v T of the passage time from i to state $i-1$) is the minimum of the r.vs of the service time of all the i busy servers and which follow an exponential distribution of rate $i\mu$. From state s and on, all the servers are busy and therefore the transition rate is $s\mu$.
The same reasoning as for the M/M/s system is followed to obtain the performance of this queuing systems. The balance equations of this system are :

$$\begin{cases} \mu\pi_1 - \lambda\pi_0 = 0 \\ \lambda\pi_{i-1} + min(s,i+1)\mu\pi_{i+1} - (\lambda + min(s,i)\mu)\pi_i = 0 \quad \text{for } 1 \leq i \leq K \\ \sum_{i=0}^{\pi} i = 1 \end{cases}$$

By following the same reasoning as the M/M/S system, we get :

$$\pi_n = \frac{s^{min(n,s)}}{min(n,s)!}\rho^n \pi_0 \quad \text{for } 1 \leq n \leq K$$

We calculate π_0 :

$$\sum_{n=s+1}^{K} \rho^{(n-s)}\pi_s + \sum_{n=0}^{s} \frac{s^n}{n!}\rho^n \pi_0 = 1$$

$$\pi_s \sum_{n=s}^{K} \rho^{(n-s)} + \pi_0 \sum_{n=0}^{s-1} \frac{s^n}{n!}\rho^n = 1$$

We deduce that:

$$\pi_0 = \frac{1}{\sum_{n=0}^{s-1} \frac{s^n}{n!}\rho^n + \frac{s^s}{s!}\rho^s \frac{1-\rho^{K-s+1}}{1-\rho}}$$

$$\pi_K = \frac{s^s}{s!}\rho^K \pi_0$$

Performances.

- *Average number of clients waiting in the queue:*
  L_q= average number of clients in the station - the average number of clients being served.

$$L_q = \sum_{n=s}^{K} (n-s)\pi_n = \sum_{n=s}^{K}(n-s)\frac{s^s}{s!}\rho^n \pi_0$$

$$= \frac{s^s}{s!}\pi_0 \rho^s \sum_{n=s}^{K}(n-s)\rho^{n-s} = \frac{s^s}{s!}\pi_0 \rho^{s+1} \sum_{k=0}^{K-s} k\rho^{k-1} \quad = \frac{s^s}{s!}\pi_0 \rho^{s+1}\left(\frac{1-\rho^{K-s+1}}{1-\rho}\right)'$$

So:

$$L_q = \frac{s^s}{s!}\pi_0 \rho^{s+1}\left(\frac{1+(K-s)\rho^{K-s+1}-(K-s+1)\rho^{K-s}}{(1-\rho)^2}\right)$$

We can calculate the other performance indicators: L, W, W_q in the same way as for M/M/1/K.

Example:
For a M/M/2/15 system with parameters $\lambda = 5c/h$ and $\mu = 4c/h$, we have :
- Average service time : $1/\mu = 1/6 = 10$ min per client
- Server utilization rate : $\rho = \lambda/s\mu = \frac{4}{2\times5} = 1/5 = 0.2$
- Probability of having 0 clients in the system : $\pi_0 = 0.23092$
- Probability that the server is busy : $1 - \pi_0 = \rho = 0.76907$
- Probability of having one client in the queue : $\pi_1 = 0.28865$
- Probability of having 0 clients in the queue : $\pi_0 + \pi_1 = 0.51957$
- Average number of clients in the system : $L = 2.04$
- Average number of clients in the queue : $L_q = 0.79$
- Average waiting time in the system : $W = 0.41$
- Average waiting time in line queue service : $W_q = 0.16$

Particular case: $\rho = 1$

$$\pi_n = \frac{s^{min(n,s)}}{min(n,s)!}\pi_0 \quad \text{for } 1 \leq n \leq K$$

$$\pi_0 = \frac{1}{\sum_{n=0}^{s} \frac{s^n}{n!} + \frac{s^s}{s!}(K-s)}$$

Let’s code!

```
#Code505.py

from Code501 import Queueing
from math import factorial

#
fsum = lambda rho,s, n : 1 if n == 0 else fsum(rho,s, n-1) + (s**n/factorial(n))*rho**n

# p0
def getP0(**p):
    rho, s, k = p[’rho1’], p[’s’], p[’k’]
    return 1/( fsum(rho,s , s-1) + ((s**s/factorial(s)) * rho**s * (1/(1 - rho))) * (1 - rho**(k-s
     +1)))

# Lq
def getLq(**p):
    rho, s, k, p0 = p[’rho1’], p[’s’], p[’k’], p[’p0’]
    return (s**s /factorial(s)) * rho**(s+1) * p0 * ((1 + (k-s)*rho**(k-s+1) - (k-s+1)*rho**(k-s))
     /(1 - rho)**2)

#
laws ={ ’MMSK’ : { ’p0’ : getP0, ’Lq’ : getLq}}

#==========================================================================
# Tests
def getMMSK(mu, lamda, S, K):
    qs = Queueing(
            model = ’MMSK’,
            A ={’D’:’Pois’ , ’params’: { ’lambda’: lamda}},
            B ={’D’:’Expo’ , ’params’: { ’mu’: mu}},
            C = S,
            K = K,
            laws=laws)
    qs.test()

if __name__ == "__main__": getMMSK(4, 5, 2, 15)

# =================MMSK =======================
# p0* :0.23092
# p1* :0.28865
# Lq  :0.79
# L   :2.04
# W   :0.41
# Wq  :0.16
```

The following table summarizes the previous results
$\lambda_e = \lambda(1-\pi_K)$ and $\rho_e = (1-\pi_K)\rho$

	M/M/1	M/M/S (S>1)	M/M/1/K	M/M/S/K (S>1)
π_0	$1-\rho$	$\left(\sum_{n=0}^{s-1} \frac{s^n}{n!}\rho^n + \frac{s^s}{s!}\rho^s \frac{1}{1-\rho}\right)^{-1}$	$\begin{cases} \frac{1-\rho}{1-\rho^{K+1}} & \rho \neq 1 \\ \frac{1}{K+1} & \rho = 1 \end{cases}$	$\begin{cases} \left(\sum_{n=0}^{s-1} \frac{s^n}{n!}\rho^n + \frac{s^s}{s!}\rho^s \frac{1-\rho^{K-s+1}}{1-\rho}\right)^{-1} & \rho \neq 1 \\ \left(\sum_{n=0}^{s} \frac{s^n}{n!} + \frac{s^s}{s!}(K-s)\right)^{-1} & \rho = 1 \end{cases}$
π_n	$\rho^n \pi_0$	$\frac{s^{min(n,s)}}{min(n,s)!}\rho^n \pi_0$	$\rho^n \pi_0$	$\frac{s^{min(n,s)}}{min(n,s)!}\rho^n \pi_0$
L_q	$\pi_0 \frac{\rho^2}{(1-\rho)^2}$	$\frac{s^s}{s!}\pi_0 \frac{\rho^{s+1}}{(1-\rho)^2}$	$\begin{cases} \frac{\rho}{1-\rho}(1-\pi_0(K\rho^K+1)) & \rho \neq 1 \\ \pi_0 \frac{K(K-1)}{2} & \rho = 1 \end{cases}$	$\begin{cases} \frac{s^s}{s!}\pi_0 \frac{\rho^{s+1}}{(1-\rho)^2}(1+(K-s)\rho^{K-s+1}-(K-s+1)\rho^{K-s}) & \rho \neq 1 \\ \frac{s^s}{s!}\pi_0 \frac{(K-s+1)(K-s)}{2} & \rho = 1 \end{cases}$
L	$L_q + \rho$	$L_q + s\rho$	$L_q + \rho_e$	$L_q + s\rho_e$
W_q	$\frac{L_q}{\lambda}$	$\frac{L_q}{\lambda}$	$\frac{L_q}{\lambda_e}$	$\frac{L_q}{\lambda_e}$
W	$W_q + \frac{1}{\mu}$	$W_q + \frac{1}{\mu}$	$W_q + \frac{1}{\mu}$	$W_q + \frac{1}{\mu}$

5.5 Exercises

Exercise 1. *In a store with only one checkout, customers arrive (at checkout) following the Poisson law. Give Kendal's notation, if:*
1. The time to scan the purchased items by each customer follows an exponential distribution, the queue is infinite.
2. The service time is fixed, the queue has size n
3. The service time follows the uniform law, the queue has size 1.

Exercise 2. *In a university department, there are 60 students. During lunch between noon and 2:00 p.m., each student uses the drink dispensers on average twice. The average service time is 2.5 mns (The inter-arrival times for students and service follow an exponential distribution). There are three distributors.*
1.Calculate the utilization rate of the servers in this system and the average waiting time in the queue.
2. A new distributor with the same characteristics is acquired. Calculate the new utilization rate and the average waiting time in the queue.

Exercise 3. *In a store, customers arrive at the checkout following the Poisson law of rate λ. The average number of items in each customer's cart is 10, and the service is exponential of rate μ customers per minute. Let A be the average number of items entered per minute by the cashier. Determine the minimum value of A so that the average waiting time does not exceed T_0.*

Exercise 4. *A telecommunication system has a transmission line of speed $V = 1800 bit/s$. Each message has a size of L bits which is an exponential r.v of average 900bits. The arrival of messages is Poisson process. We are looking for the maximum arrival rate (messages/s) that the system can support so that the waiting time for each message in the queue is less than 2s.*

Exercise 5. *In a post office, there is only one counter that can serve an average of 6 customers per hour. Customers' arrivals are Poisson process of rate 5 customers per hour.*
1. Calculate the average number of customers and the average waiting time in the system.
2. The manager wants to add a second counter with an independent (separate) queue from that of the first one. Recalculate the same performance in the new system.
3. Suppose the customers choose the first counter with probability $p = 2/3$. Find the average number of customers and the average waiting time in this system.

Exercise 6. *In a hairdressing salon, there is only one hairdresser who operates. Customers arrive in a Poisson process with an average of 32 customers in 8 hours. The duration of a cut is exponential of average equal to 10 minutes for each customer. Customers are served in the order of their arrival and there is no limit on seatings in the hairdressing salon.*

1. *Calculate the probability that exactly 4 customers will arrive between 8 am and 9 am.*
2. *Calculate the probability that a customer will be served for more than 20 minutes.*
3. *Calculate π_n the probability of having n customers in the salon at a given time (in the steady state).*
4. *Calculate the average number of customers and the average waiting time in the salon.*

Exercise 7. *In a queuing system, customers arrive following a Poisson process of rate $\lambda = 10$ per hour and the service time is exponentially distributed of average 4 minutes if the number of customers in the system is less than 3 otherwise the average service time is 2 minutes.*
1. Draw the transition diagram of this system.
2. Find the steady-state probability.
3. Find the performances of this system.

Exercise 8. *Consider a station with Poisson arrivals of rate λ. The service is exponential of rate μ . If the number of customers in the station is less than s_L, the service rate is lowered to μ_L, and when it reaches s_H ($s_H \geq s_L$), the service rate is increased to μ_H.*
1. Draw the transition diagram.
2. Find the expression of π_n as a function of π_0.

Exercise 9. *Consider a queuing system with a single server and a queue of infinite size. The arrivals are Poissonian of rate* λ *and the service is exponential of rate* $k\mu$ *when the system is in state* k *for* $k = 0, 1, 2, ...$ *(the server increases the service rate according to the number of customers in the queue).*
Calculate the average number of customers and the average waiting time in this system.

Exercise 10. *A service station has only one gasoline pump. Cars arrive following a Poisson process of average 15 cars per hour. If the pump is busy, the customers may leave without receiving service. When there are* n *cars in the station, the probability that a customer leaves without service is* $n/3$*. The service time follows an exponential law of mean 4 minutes.* $X(t)$ *represents the number of cars in the station:*

1. *Build the transition graph of this queuing system.*
2. *Find the probabilities of the steady state.*
3. *Find the average number of customers in the station.*
4. *Find the average waiting time in the station.*

5.6 Solutions

Solution 1.
1- M/M/1
2- M/D/1/n+1
3- M/G/1/2

Solution 2.
$\lambda = 60c/h, \mu = 24c/h$
$1 - s = 3, \rho = 60/(3 \times 24) = 5/6 = 83.33\%, W_q = 3,6min$
$2 - s = 4, \rho = 60/(4 \times 24) = 5/8 = 62.5\%, W_q = 0.6min$

Solution 3.
The cashier can serve $A/10$ *clients per minutes* $(\mu = A/10)$.
$W = \frac{1}{\mu-\lambda} < T_0 \implies \mu > (T_0^{-1} + \lambda) \implies A > 10(T_0^{-1} + \lambda)$

Solution 4.
In this system the transmission line represents the server. On average, it requires L/V *seconds to be transmitted,* $\mu = V/L = 2messages/s$*. We want that* W_q *be less than* $2s$*, so:*
$\frac{1}{\mu-\lambda} - \frac{1}{\mu} < 2$ *which implies:* $\lambda < 8/5$.

Solution 5.
1. $\lambda = 5$, $\mu = 6$, *the system is* $M/M/1$, *so* $\rho = \frac{5}{6}$, $L = \frac{\rho}{1-\rho} = 5$, $W = \frac{1}{\mu-\lambda} = 1$.
2. In the new system, each customer chooses the queue with probability 1/2. So the arrival rate for each server is $\lambda/2$*. We calculate the performance of M/M/1 with this arrival rate, which gives:*
$\lambda = 5/2$, $\mu = 6$, *the system is* $M/M/1$, *so* $\rho = \frac{5}{12}$:
$L = 2 \times \frac{\rho}{1-\rho} = 10/7$.
$W = \frac{1}{2} \times \frac{1}{\mu-\lambda} + \frac{1}{2} \times \frac{1}{\mu-\lambda} = 2/7$.
3. $\lambda_1 = 10/3$, $\lambda_2 = 5/3$, *so:* $\rho_1 = \frac{5}{9}$ *and* $\rho_2 = 5/18$
$L_1 = \frac{\rho_1}{1-\rho_1} = 5/4$ *and* $L_2 = \frac{\rho_2}{1-\rho_2} = 5/13$; $L = L_1 + L_2 = 1.63$
$W_1 = \frac{1}{\mu-\lambda_1} = 3/8$ *and* $W_2 = \frac{1}{\mu-\lambda_2} = 3/13$; $W = 2/3 \times W_1 + 1/3 \times W_2 = 0.32$

Solution 6.
1- The station is $M/M/1$, $\lambda = 4$ *and* $\mu = 6$, *so* $\rho = \frac{2}{3}$.
$N(t)$ *is the number of customers arriving during a time interval* t. $P(N(t) = k) = e^{-\lambda t}\frac{(\lambda t)^k}{k!}$
$P(N(1) = 4) = e^{-4}\frac{4^4}{4!} = 0.195$.

2- Let T_s *be the service time :* $P(T_s > t) = e^{-\mu t}$
$P(T_s > \frac{1}{3}) = s^{-2} = 0.135$.

3- $\pi_n = \rho^n(1-\rho) = \frac{1}{3}(\frac{2}{3})^n$.

4- $L = \frac{\rho}{1-\rho} = 2$ *clients,* $W = \frac{1}{\mu-\lambda} = \frac{1}{2}$ *hour=30 mns.*

Solution 7.

$\lambda = 10,\ \mu_1 = 15,\ \mu_2 = 30$

1- 2. The balance equations :

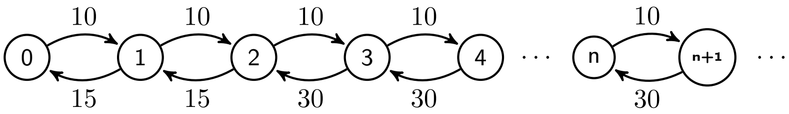


Figure 5.7: Diagram of exercise 7

$$\begin{aligned}
\mu_1\pi_1 &= \lambda\pi_0\\
\lambda\pi_0 + \mu_1\pi_2 &= (\lambda+\mu_1)\pi_1\\
\lambda\pi_1 + \mu_2\pi_3 &= (\lambda+\mu_1)\pi_2\\
\lambda\pi_{n-2} + \mu_2\pi_n &= (\lambda+\mu_2)\pi_{n-1}, \qquad \forall n > 3
\end{aligned}$$

So:

$$\pi_1 = \frac{\lambda}{\mu_1}\pi_0, \qquad \pi_n = (\frac{\lambda}{\mu_2})^{n-2}(\frac{\lambda}{\mu_1})^2\pi_0, \qquad \forall n > 1$$

We have: $\sum_{i=0}^{\infty}\pi_i = 1$

$$\pi_0 + \frac{\lambda}{\mu_1}\pi_0 + \sum_{n=2}^{\infty}(\frac{\lambda}{\mu_2})^{n-2}(\frac{\lambda}{\mu_1})^2\pi_0 = 1$$

$$\pi_0 = \frac{1}{1+\frac{\lambda}{\mu_1}+(\frac{\lambda}{\mu_1})^2(\frac{1}{1-\frac{\lambda}{\mu_2}})} = \frac{3}{7}$$

So: $\pi_1 = \frac{2}{7}$, $\pi_2 = \frac{4}{21}$ *and* $\pi_n = \frac{4}{7}(\frac{1}{3})^{n-1} \quad for \quad n > 2$

3. $L = \pi_1 + 2\pi_2 + \frac{4}{7}\sum_{n=3}^{\infty} n(\frac{1}{3})^{n-1} = \frac{2}{7} + \frac{8}{21} + \frac{7}{4}(\frac{1}{(1-\frac{1}{3})^2} - \frac{5}{3}) = 1$
$L_q = \sum_{n=1}^{\infty}(n-1)\pi_n = \sum_{n=1}^{\infty} n\pi_n - \sum_{n=1}^{\infty}\pi_n = L - (1-\pi_0) = \frac{3}{7}$
$W = \frac{L}{\lambda} = \frac{1}{10}$; $W_q = \frac{L_q}{\lambda} = \frac{3}{70}$

Solution 8.

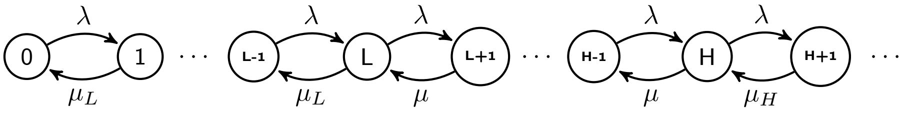


Figure 5.8: Diagram of exercise 8

$$\begin{aligned}
\lambda\pi_n &= \mu_L\pi_{n+1} \quad \textit{if } n < s_L\\
\lambda\pi_n &= \mu\pi_{n+1} \quad \textit{if } s_L \le n < s_H\\
\lambda\pi_n &= \mu_H\pi_{n+1} \quad \textit{if } n \ge s_H
\end{aligned}$$

$$\pi_n = \begin{cases}
(\frac{\lambda}{\mu_L})^n\pi_0 & \textit{if } n < s_L\\
(\frac{\lambda}{\mu})^{n-s_L+1}(\frac{\lambda}{\mu_L})^{s_L-1}\pi_0 & \textit{if } s_L \le n < s_H\\
(\frac{\lambda}{\mu_H})^{n-s_H+1}(\frac{\lambda}{\mu})^{s_H-s_L}(\frac{\lambda}{\mu_L})^{s_L-1}\pi_0 & \textit{if } n \ge s_H
\end{cases}$$

Solution 9.

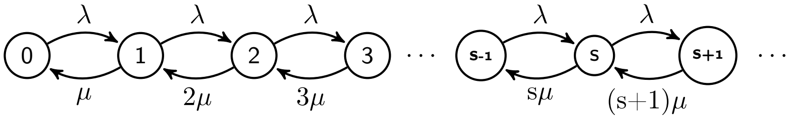


Figure 5.9: Diagram of exercise 9

The equations system in the stationary state is as follows: $\boldsymbol{\pi}\mathbf{Q} = \mathbf{0}$ *and* $\boldsymbol{\pi}\mathbf{1} = 1$

$$\begin{cases} -\lambda\pi_0 + \mu\pi_1 = 0 \\ \lambda\pi_{k-1} - (\lambda + k\mu)\pi_k + (k+1)\mu\pi_{k+1} = 0, \qquad \forall n > 0 \\ \sum_{i=0}^{\infty} \pi_i = 1 \end{cases}$$

From this system of equations we have the following relation which can be verified by induction.

$$\pi_i = \frac{\rho^i}{i!}\pi_0 \qquad \textit{such that } \rho = \frac{\lambda}{\mu}$$

From $\sum_{i=0}^{\infty} \pi_i = 1$ *and under the condition that* $\rho < 1$*, we have:*

$$\pi_0 = \frac{1}{1 + \rho + \frac{1}{2}\rho^2 + ... + \frac{1}{i!}\rho^i + ...}$$

So: $\pi_0 = e^{-\rho}$ *and* $\pi_n = \frac{\rho^i}{i!}e^{-\rho}$
We can notice that the expression of π_n *corresponds to the Poisson distribution of parameter* ρ*.*
We can therefore deduce that:

$$L = \sum_{i=0}^{\infty} i\pi_i = \rho \qquad \textit{the Poisson distribution expectation.}$$
$$W = \frac{L}{\lambda} = \frac{1}{\mu}$$

Solution 10.
The possible states set is $S = \{0, 1, 2, 3\}$*, each state represents the number of clients in the station. The transition rate from state* i *to state* $i + 1$ *is equal to :* $\lambda \times (1 - \frac{i}{3})$

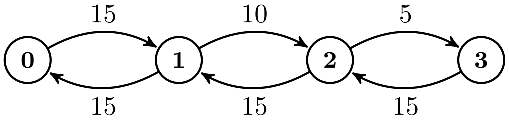


Figure 5.10: Diagram of Exercise 10

2. In the steady state $\pi \times Q = 0$ *and* $\sum_{i=0}^{3} \pi_i = 1$*, this system of equations gives:*
$\pi = (\frac{9}{26}, \frac{9}{26}, \frac{3}{13}, \frac{1}{13})$
3. The average number of customers in the station: $L = \sum_{n \in S} n \times \pi_n = \frac{27}{26}$
4. The average waiting time in the station. Since the arrival rate changes from state to state, Little's law must be applied using λ_e *which is equal to the expectation of* λ_i*:*
$\lambda_e = \sum_{i=0}^{3} \lambda_i\pi_i = 9.80$
$W = \frac{L}{\lambda_e} = 0.1059$

Part III

Simulation

Chapter 6

Random Variables Generation

6.1 Introduction

Random numbers generators are the basis of the simulation of any probability distribution. Unlike manual methods (coins, dice, cards, ...), tables of random numbers and physical processes; algorithmic generators produce pseudo-random numbers (because they come from deterministic algorithms), but they have the advantage of being simple to produce on a computer. They allow a very large number of experiments to be carried out in a very short time. They can repeat the same sequence multiple times which is important in the context of debugging and programs verification, comparing systems by simulation and reducing variance.

6.2 Generation of random numbers

> **Definition 1.** *[15]*
> *The pseudo-random number generator (PRNG) is a structure* (S, P_0, f, U, g) *such that:*
> *-* S *: a finite set of states (states space)* $s_0, s_1, \cdots, s_n$
> *-* P_0 *: probability distribution on* S *to select the initial state* s_0 *(seed)*
> *-* f *: transition function* $f : S \rightarrow S$
> *-* U *: space of generated numbers (output) (often* $[0, 1]$*)*
> *-* g *: output function* $g : S \rightarrow U$

The elements of this structure make it possible to perform the steps of random numbers generation, which are :
1- Select s_0 using P_0, then generate the first random number $u_0 = g(s_0)$
2- At each step $i \geq 1$, the transition function changes the state of the generator and then the output function gives the random number associated with the new state.

$$s_i = f(s_{i-1}) \ \& \ u_i = g(s_i)$$

These two functions express the dynamics of this system. We can express the current state according to the initial state s_0:

$$s_i = f^{(n)}(s_0) \ \& \ u_i = (g \circ f^{(n)})(s_0)$$

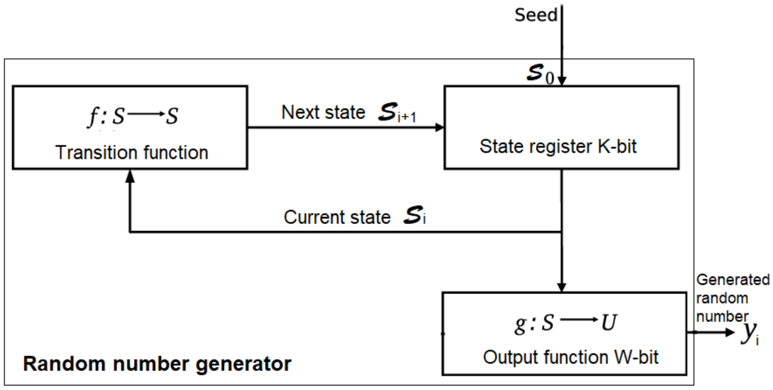


Figure 6.1: Random number generator

The sequence $u_0, u_1, \cdots$ returned by g is the sequence of random numbers generated by the RNG. Since S is finite and f is from S in S, then there exists $p \in \mathbb{N}$ and $s_k \in S$ so that $s_k = f^{(p)}(s_k)$ i.e $s_{k+p} = s_k$. The smallest number p which satisfies this property is called the period of the PRNG. From this state, the sequence becomes cyclic with cycle length p. This number is upper bounded by the cardinality of S ($p \leq |S|$ its upper bound). A long time period is better than a short one but it does not ensure the good quality of the generator.
Several methods of generating random numbers exist which are defined by the specification of the elements of the RNG structure. They all start from the initial value s_0: *seed* and calculate the successive values s_n recursively. The implementation of many generators is optimized for efficiency on the target platform on which it is going to be run. Often, the size of the memory word of this platform is considered to be an important factor in the choice of the parameters and in making the calculation optimal and more efficient. In the rest of this section we will present the most used RNGs.

6.2.1 Linear congruential generators

LCG(a,m) are used by most programming languages because of their simplicity and quickness. They are defined on the set ($\mathbb{Z}_m$) (*congruential generator CG*) with $m = |S|$ and they define f as a linear function defined on $\mathbb{Z}_m$, g is the quotient of the result of f by m :

$$x_n = f(x_{n-1}) = (ax_{n-1})[m] \qquad \text{and} \qquad u_n = g(x_n) = \frac{x_n}{m}$$

a is the *multiplier* and m is the *module*. The value u_n is the generated *pseudo random number*. It is considered to be the approximation of the value of the uniform random variable in [0,1].

6.2.2 Mixed congruential methods

MCM(a,c,m) is a congruential generator with f an affine function defined on $\mathbb{Z}_m$ with $m = |S|$ and $a, c \in \mathbb{Z}_m$:

$$x_n = f(x_{n-1}) = (ax_{n-1} + c)[m] \qquad \text{and} \qquad u_n = g(x_n) = \frac{x_n}{m}$$

c is called the *increment*.

Examples :
1- MCM(a,c,m)=$(128, 1, 2^{35})$, $x_n = f(x_{n-1}) = (ax_{n-1} + c)[m] = (2^7 x_{n-1} + 1)[2^{35}]$
2- MCM(a,c,m)=$(69069, 1327217885, 2^{32})$, $x_n = (69069x_{n-1} + 1327217885)[2^{32}]$
3- LCG(a,m)=$(2^{16}, 2^{31})$, $x_n = (2^{16}x_{n-1})[2^{31}]$
4- LCG(a,m)=$(7^5, 2^{31} - 1)$, $x_n = (7^5 x_{n-1})[2^{31} - 1]$ period $p = 2^{31} - 2$
5- MCM(a,c,m)=$(25214903917, 11, 48)$, $x_n = (25214903917x_{n-1} + 11)[48]$

Theorem 7. *Hull-Dobell Thoerem* [22]
The generator MCM specified by (a, c, m) *has the period* m *(complete period) iff :*
1- $pgcd(c, m) = 1$ *(c and m are relatively prime)*
2- $\forall p \in \mathcal{P}^*, p|m \implies p|(a-1)$
3- $4|m \implies 4|(a-1)$

$\mathcal{P}^*$ is the set of prime numbers and $x|y$ means that x is a divisor of y.
Let X be a uniform d.r.v on $\mathbb{Z}_m$ to be simulated by a CG. Its expectation and variance are :

$$\mathbb{E}[X] = \sum_{k \in \mathbb{Z}_m} k\frac{1}{m} = \frac{m-1}{2} \text{ and } \mathbb{V}[X] = \sum_{k \in \mathbb{Z}_m} k^2\frac{1}{m} - \frac{(m-1)^2}{4} = \frac{m^2-1}{12}$$

Let U be a c.r.v such that :

$$U = \frac{X}{m} \text{ and } \mathbb{E}[U] = \frac{1}{2}(1 - \frac{1}{m}) \text{ and } \mathbb{V}[U] = \frac{1}{12}(1 - \frac{1}{m^2})$$

when $m >>$, $U \sim \mathbb{U}[0, 1]$
The choice of a, c is crucial for the quality of the RNG.

6.2.3 Multiple recursive generators

MRG is a recursive equation of order greater than 1 (the current term is a function of previous terms).

$$x_n = (a_1x_{n-1} + a_2x_{n-2} + ... + a_kx_{n-k} + c)[m]$$

for $k > 1$, generation time increases and improves the period and the random properties.
For some cases, the MRG can be simplified to the form :

$$x_n = (a_rx_{n-r} + a_kx_{n-k})[m] \text{ such that } r < k$$

Examples :

RNG_1 = MRG(k=3, a=143580, b=-810728, m$_1 = 2^{32} - 209$)
$X_n = (143580X_{n-2} - 810728X_{n-3})[(2^{32} - 209)]$

RNG$_2$ = MRG(k=3, a=527612, b=-1370589, m$_2 = 2^{32} - 22853$)
$Y_n = (527612Y_{n-1} - 1370589Y_{n-3})[(2^{32} - 22853)]$

Let's code!

```
# Code601.py

from functools import partial

# number of generated values
N=200

# plot the random number points in a grid
def plotRNG(x,y, title):
    import matplotlib.pyplot as plt
    fig, axs = plt.subplots(1, 2, constrained_layout=True, figsize=(10,5))
    axs[0].scatter(x, y)
    axs[1].plot(range(N-1),x)
    fig.suptitle(title, fontsize=16)

# Congruential Generator
# conGen is a Python generator that yields a random number each time next() is called
def congGen(rng_function, seed, m, **params):
    x_n = seed
    while True:
        yield x_n[0]
        s = rng_function(x_n, **params) % m
        x_n = x_n[1:]; x_n.append(s)
```

```
# a_1 x_n-1 + a_2 x_n-2 + .... + a_k x_n-k + c
def sumRNG(x, **p):
    s = 0
    for i in range(p[’k’]): s += p[’a’ + str(i)] * x[i]
    return s + p[’c’]

# partial congruential generator
pcg = partial(congGen, lambda x, **p : sumRNG(x,**p))

# test RNG
def test(smas):
    for sma in smas:
        seed,m , abc = sma[0], sma[1], sma[2]
        cgi = pcg(seed, m, **abc); rs = [ next(cgi) for _ in range(N)]
        plotRNG(rs[:-1],rs[1:], sma[3])

# Random number generators
test([
      [[1], 2**10, {’k’:1,’a0’:99, ’c’:0}, ’Congruentiel Lineaire’],
      [[1], 2**10, {’k’:1,’a0’:99, ’c’:1}, ’Congruentiel Mixte’],
      [[1,2,3], 2**10, {’k’:3,’a0’:99, ’a1’:2, ’a2’:3,  ’c’:1}, ’Congruentiel Recursif’],

      [[1], 2**10, {’k’:1,’a0’:95, ’c’:0}, ’Congruentiel Lineaire’],
      [[1], 2**10, {’k’:1,’a0’:95, ’c’:1}, ’Congruentiel Mixte’],
      [[1,2,3], 2**10, {’k’:3,’a0’:95, ’a1’:2, ’a2’:3,  ’c’:1}, ’Congruentiel Recursif’]
      ])
```

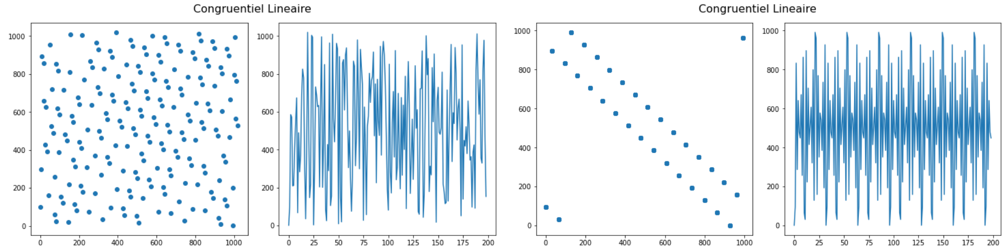


Figure 6.2: linear congruential generator. left a=99, and right a=95

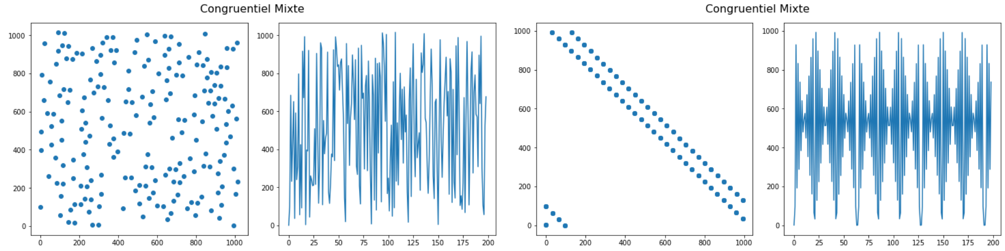


Figure 6.3: Mixed congruential generator. left a=99, and right a=95

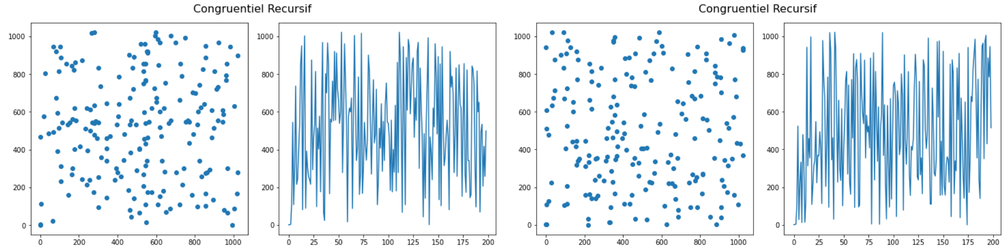


Figure 6.4: Recursive congruential generator. left a=99, and right a=95

The numbers generated by LCGs sometimes create a uniform structure called a lattice (which has the appearance of parallel lines, planes or hyper-planes) when the sequence of pairs (x_n, x_{n+1}) is plotted. It has been shown that the farther the planes are, the poorer the generator quality [8].

It can be noted that for this example, when the parameter a = 95, the recursive congruential generator gives a sequence of more apparent randomness than the other two generators. Also, for the three generators, the sequences generated with a value of a = 99 are more random than those generated with 95. The spectral test compares the distances between the planes of the lattice structure, the larger they are, the worse the generator is [5].

6.2.4 Mersene Twister RNG (MT)

Mersene Twister RNG (MT) [16] is a well designed and powerful RNG characterized by its long period (2^{19937}). It is a feedback shift register with twisting feature (TFSR). It has w bits as a word length and $\mathbb{Z}_{2^w-1}$ as output set. Its algorithm is based on recurrence relation for the seed initialization and for the state twisting. It consists of two steps, the first one tries to loop through the state and twist it recursively (state bit reflection), and the second generates the random number by tempering the current output. The Mersenne Twister algorithm manipulates a matrix of linear recurrence relations over the finite binary field $(\mathbb{F}_2, \oplus, \otimes)$ ($\oplus$ is xor, $\otimes$ is multiplication, over $\mathbb{Z}_2$) where each number is represented as a sequence of bits $(b_{w-1} \cdots b_1 b_0)$ and it could be written as a polynomial in this field $\sum_0^{w-1} b_i 2^i$. Its parameters :
- w: word size (number of bits)
- n: recurrence degree
- m: an offset used in the recurrence relation defining the series x, $1 \leq m < n$
- r: the number of bits of the lower bitmask, $0 \leq r \leq w-1$
- a: the rational normal form twist matrix
- b, c: tempering bitmasks
- s, t: tempering bit shifts
- u, d, l: additional Mersenne Twister tempering bit shifts/masks

In order to make computation and test easy these parameters are selected in such a way that $2^{nw-r} - 1$ is a Mersenne prime. Twisting recurrence relation:

$$x_{i+n} = x_{i+m} \oplus \overbrace{(\text{msb}(x_i, w-r) \quad | \quad \text{lsb}(x_{i+1}, r))}^{\text{(concat of } r{-}LSB\&(w-r){-}MSB)} A$$

such that LSB(respc MSB) are the r least(respc $w-r$ most) significant bits of the current word x_i and A is a matrix called twisting transformation matrix, defined as :

$$A = \begin{pmatrix} 0 & \mathbb{I}_{w-1} \\ a_w & a_{w-1} \dots a_1 \end{pmatrix}$$

The multiplication by the matrix A is the relational normal form matrix, it is simplified by the following relation :

$$xA = (x >> 1) \oplus si(x_0 = 1, a, 0^w)$$

Tempering function ϕ_g:

$$\begin{aligned} \phi_g(z, t, m) &= z \oplus ((z \gg t) \& m) \\ \phi_l(z, t, m) &= z \oplus ((z \ll t) \& m) \\ y = \phi(x) &= \phi_g(\phi_l(\phi_l(\phi_g(x_i, u, d), s, b), t, c), l, 1^w) \end{aligned}$$

where t and m are shift-number and bit-masks respectively, "&" is a bit-wise-and operation, and $\gg$ is the shift operator. $y = \phi(x)$ will be the output (random number) of the algorithm in $\mathbb{Z}_{2^w-1}$, in order to get a uniform random number in $[0, 1]$, we choose g as the function

$$u = \frac{\phi(x)}{2^w - 1}$$

There are two variants of this algorithm based on the platform of the implementation 32bits variant called MT19937 and 64bits called MT19937-64. Their parameters are respectively :

Implementation	MT19937-32bits	MT19937-64bits
(w, n, m, r)	(32, 624, 397, 31)	(64, 312, 156, 31)
(l,a)	(18, 9908B0DF16)	(43, B5026F5AA96619E916)
(u, d)	(11, FFFFFFFF16)	(29, 555555555555555516)
(s, b)	(7, 9D2C568016)	(17, 71D67FFFEDA6000016)
(t, c)	(15, EFC6000016)	(37, FFF7EEE00000000016)
f	1812433253	6364136223846793005

Given a specific seed, we start initializing the state of the algorithm (an array of n elements in $\mathbb{Z}_{2^w-1}$) using the following recurrence relation starting with $x_0 = seed$:

$$x_i = (f * (x_{i-1} \oplus x_{i-1} >> (w-2))) + i)$$

Algorithm 1 Mersene Twister Generator

```
Input: seed
$i, (x_i)_{i:0..n-1} = 0, \text{initialize}(seed)$
while  True:
    $y = (x_i \wedge u) \vee (x_{(i+1)[n]} \wedge l)$
    $x_i = (x_{(i+m)[n]} \oplus (y >> 1) \oplus si(LSN(y) = 0, 0, a)$
    yield tempering($x_i$)
    $i = (i+1)[n]$

function initialize($s$)
    $z_0 = s$
    for i $\in \{1..n-1\}$ : z$_i = (f * (z_{i-1} \oplus z_{i-1} >> (w-2))) + i)$
    return $(z_i)_{i:0..n-1}$

function tempering($x$)
    $z = x \oplus (x >> q)$
    $z = z \oplus ((z << s) \wedge b))$
    $z = z \oplus ((z << t) \wedge c))$
    return $z \oplus (z >> p)$
```

Let's code!

The following code gives the Python implementation of the MT19937 variant.

```python
# Code601_1.py

# algo MT19937
def algo_MT19937(seed=0):
    # coefficients for MT19937
    (w, n, m, r) = (32, 624, 397, 31)
    a = 0x9908B0DF
    (u, d) = (11, 0xFFFFFFFF)
    (s, b) = ( 7, 0x9D2C5680)
    (t, c) = (15, 0xEFC60000)
    (l, f) = (18, 1812433253)
    (lower_mask, upper_mask)= (0xFFFFFFFF, 0x00000000)

    # The state of the generator array of n elements
    MT = [0 for i in range(n)]
    index = n+1

    # initialize the generator from a seed
    def initialize_mt(seed):
        MT[0] = seed
        for i in range(1, n):
            MT[i]  = (f * (MT[i-1] ^ (MT[i-1] >> (w-2))) + i)  & 0xffffffff

    # Generate a random number
    def generate_number():
        # do a twist on every n numbers
        nonlocal index
        if index >= n:
            for i in range(0, n):
                x = (MT[i] & upper_mask) + (MT[(i + 1) % n] & lower_mask)
                xA = x >> 1 if (x % 2) == 0 else (x >> 1) ^ a
                MT[i] = MT[(i + m) % n] ^ xA
            index = 0

        # Extract a tempered value based on MT[index]
        y = MT[index]
        y = y ^ ((y >> u) & d)
        y = y ^ ((y << s) & b)
        y = y ^ ((y << t) & c)
        y = y ^ (y >> l)

        index += 1
        return y & 0xffffffff

    # Generator core
    initialize_mt(seed)
    while True:
        yield generate_number()

# generate 10 random numbers
if __name__ == '__main__':
    g_MT= algo_MT19937()
    for _ in range(10): print(next(g_MT))

# ======================================================================
2357136044
2546248239
3071714933
3626093760
3729171009
3684848379
3480577985
2632805477
679261451
3685339089
```

6.2.5 Combined RNG

A combined RNG is made up of several RNGs composed by the application of an operation which combines the different results of the LCGs. Addition, xor or any other operation can be used for this combination. This process makes it possible to improve the quality of these generators by creating an equivalent generator with long periods and possibly with better statistical properties.

6.2.5.1 Sum/Subtraction

This technique consists in summing the generated results by the CGs based on the following two propositions:

> **Proposition 1.** *If $(W_i)_{1\leq i\leq n}$ is a collection of independent d.r.v, such that W_1 is uniform over $\mathbb{Z}_d$, then the r.v of the sum of this collection in $\mathbb{Z}_d$ is uniform over this set.*
>
> $$W_1 \sim \mathcal{U}(\mathbb{Z}_d) \implies \sum_{1}^{n} W_i \sim \mathcal{U}(\mathbb{Z}_d)$$

> **Proposition 2.** *If $(L_i)_{1\leq i\leq n}$ is a family of CG such as L_j with period p_j and transition function $s_{j,i} = f_j(s_{j,i-1})$ and the starting seed is $s_0 = (s_{1,0}, s_{2,0}, \cdots, s_{n,0})$ and p is the period of the sequence $s_i = (s_{1,i}, s_{2,i}, \cdots, s_{n,i})$ generated by this family then*
>
> $$p = LCM(p_1, p_2, \cdots, p_n)$$

The LCM is the least common multiple for two integers.

6.2.5.2 Wichmann-Hill

It consists in generating two or more uniform real numbers $U_{j,i} = \frac{X_{j,i}}{m_j}$ on $[0, 1]$ using the CGs and taking the decimal part of the sum of $U_{j,i}$.

$$V_n = h(\frac{X_{1,n}}{m_1} + \frac{X_{2,n}}{m_2} + \cdots + \frac{X_{k,n}}{m_k})$$

Such that $h(x) = x - \lfloor x \rfloor$ is the function that gives the decimal part of a real number.

The Wichmann-Hill generator

L$_1$ = LCG(171, 30629) output X $_n$ = L$_1$(X$_{n-1}$)
L$_2$ = LCG(172, 30307) output Y $_n$ = L$_2$(Y$_{n-1}$)
L$_3$ = LCG(170, 30323) output Z $_n$ = L$_3$(Z$_{n-1}$)

$$U_n = h(\frac{X_n}{m_1} + \frac{Y_n}{m_2} + \frac{Z_n}{m_3})$$

These three generators L_1, L_2 and L_3 have the maximum period $(m_i - 1)$ and the respective periods are 30268, 30306 and 30322. The period of the combined generator is the smallest common multiple (LCM) of these three periods 6953607871644 [4].

6.2.6 Quality criteria

A good random number generator should:

1- have a very long period (which must be proven mathematically) to avoid looping quickly when used,

2- be efficient in terms of execution and memory usage,

3- the generated sequences must be reproducible,

4- and obviously the most important criterion of random number generators is the randomness of the generated sequences. In other words, the generated sequence must be uniform r.v s over (0,1), independent and identically distributed.

The numbers obtained from the algorithmic generators present in most programming languages are r.v's which follow the uniform distribution, either in the interval $[0, 1]$ or in $[0, Max]$ such that Max is the maximum limit given by the programmer. In the next section, we'll see how to turn those uniform r.v's into r.v's of other distributions.

6.3 Statistical tests of random number generators

The field of statistical tests of RNGs remains an active field of research, and to this day, several tests have been proposed, among these we will see the frequency test for uniformity and the autocorrelation test for independence. The uniformity and independence are the most tested properties for RNGs. The tests that we are going to see are based on *hypothesis tests* known in statistics. The idea is to have two hypotheses:

1. H_0: the null hypothesis, in the case of uniformity for example, this hypothesis states that the RNG is uniformly distributed.
2. H_1: the alternative hypothesis which states that the RNG is not uniformly distributed.

The result decides whether the test rejects H_0 or fails to reject it (we speak of failure of rejection instead of acceptance because to accept a hypothesis we must test an infinite number of cases. A failure of the rejection means that no evidence of non-compliance was detected).

Two types of errors must also be defined:

- Type I error: consists of rejecting H_0 when it is true.
- Type II error: consists of not rejecting H_0 when it is false.

The result of the test is always established with a significance level α which is the probability of rejecting H_0 while it is true (standard error I) P(rejecting $H_0 | H_0$ is true). We are looking for a small probability, but reducing α increases the probability of type II error, so a balance between the two is necessary.

6.3.1 Frequency test

Two frequency tests exist: the *Kolmogorov-Smirnov* test and the *Chi-square* test.

6.3.1.1 Kolmogorov-Smirnov (K-S) test

This test is used to decide whether a sequence follows a specific (theoretical) distribution.
Consider a sequence of N ordered data (from smallest to largest): $Y_1, Y_2, ..., Y_n$. The empirical cumulative distribution function is defined by: $E_n(Y_i) = n(i)/N$ such that $n(i)$ is the number of values less than or equal to Y_i. The $n(i)$ values are also ordered from the smallest to the largest.
The K-S test is based on the maximum distance between the theoretical distribution and the empirical distribution. We define the hypotheses:
H_0: the data follows a specific distribution
H_1: the data does not follow the specific distribution.
Consider the statistic:

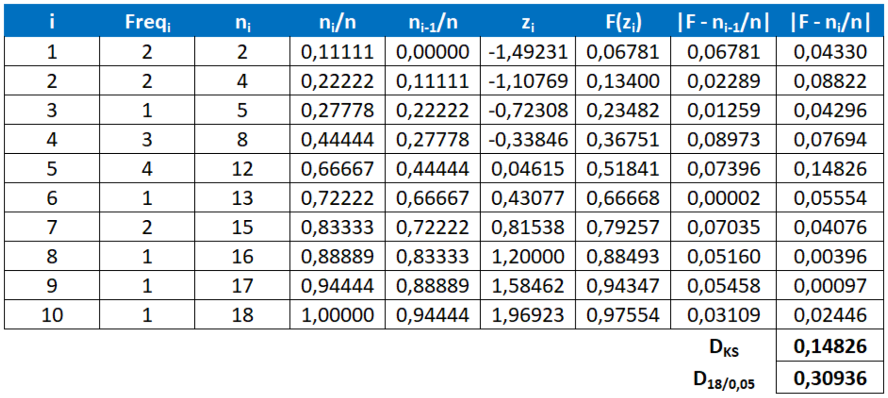
$$D_{KS} = \max_{i=1..N} |F(Y_i) - E_n(Y_i)|$$

F is the cumulative function of the theoretical distribution to be tested which must be continuous and completely specified (all its parameters must be known).
The value D_α is determined from the K-S statistics table (see appendix C) for the signification level α and N. If $D_{KS} > D_\alpha$, the null hypothesis H_0 is rejected otherwise there is no evidence to reject it.

Example 1. *We want to know (with $\alpha = 0.05$) if the below serie follows a Normal distribution:*

$$4, 5, 5, 1, 1, 3, 2, 2, 4, 10, 7, 5, 5, 4, 8, 9, 7, 6$$

The method consists of sorting the values in ascending order, then centering and reducing them, calculating the statistic D_{KS} described above and finally looking for the value in the K-S table (appendix KS-table C) corresponding to N and α to decide whether to reject the hypothesis or not.



i	$Freq_i$	n_i	n_i/n	n_{i-1}/n	z_i	$F(z_i)$	$\lvert F - n_{i-1}/n \rvert$	$\lvert F - n_i/n \rvert$
1	2	2	0,11111	0,00000	-1,49231	0,06781	0,06781	0,04330
2	2	4	0,22222	0,11111	-1,10769	0,13400	0,02289	0,08822
3	1	5	0,27778	0,22222	-0,72308	0,23482	0,01259	0,04296
4	3	8	0,44444	0,27778	-0,33846	0,36751	0,08973	0,07694
5	4	12	0,66667	0,44444	0,04615	0,51841	0,07396	0,14826
6	1	13	0,72222	0,66667	0,43077	0,66668	0,00002	0,05554
7	2	15	0,83333	0,72222	0,81538	0,79257	0,07035	0,04076
8	1	16	0,88889	0,83333	1,20000	0,88493	0,05160	0,00396
9	1	17	0,94444	0,88889	1,58462	0,94347	0,05458	0,00097
10	1	18	1,00000	0,94444	1,96923	0,97554	0,03109	0,02446
							D_{KS}	**0,14826**
							$D_{18/0,05}$	**0,30936**

$D_{KS} = 0.148 < D_{18,0.05} = 0.30936$, *so H_0 is not rejected.*

Let's code!

```
#Code602.py

from scipy import stats
from scipy.stats import ksone

# stats.kstest: returns the calculeted KS statistic
# ksone.ppf: returns the KS table value correspondent to alpha and n
def testKS(data,alpha,F,p):
    n = len(data)
    DKS = stats.kstest(data, lambda x: F(x,**p))[0];  print('DKS    :', DKS)
    DA = ksone.ppf(1-alpha/2, n);                      print('Dalpha :', DA)
    return(DKS<DA)
```

```
# Test, loc is the sequence average and scale is its standard deviation
x = [4,5,5,1,1,3,2,2,4,10,7,5,5,4,8,9,7,6]
alpha = 0.05
print('H0     : ',testKS(x, alpha, stats.norm.cdf, {'loc':4.88,'scale':2.60}))

#_______________________________   Output   _______________________________________
# DKS     : 0.14826048100509914
# Dalpha : 0.30936031179258944
# H0      : True
```

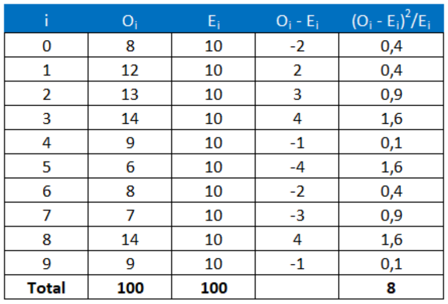
6.3.1.2 Chi-square test (χ^2)

This test is based on the same idea as the K-S test except that the statistic used is the expectation of the difference between the theoretical frequency and the observed frequency.
Consider a sample of n independent observations assumed to belong to some distribution. This observation sequence can be grouped into k intervals or classes. The number of observations within the i^{th} class is called the "observed frequency O_i". The number of theoretical observations within this class is called "theoretical frequency E_i" (Expected).

$$\chi_c^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$$

This value is called the calculated Chi-square statistic. For accuracy, the number of frequencies observed in each class must be greater than 5.

Example 2. *Consider a sequence of 100 random numbers between 0 and 9. The tested hypothesis are:*
H_0: the generated sequence follows the uniform distribution in the interval [0,9]
H_1: the generated sequence does not follow the uniform distribution in the interval [0,9]
The probability of occurrence of each integer is $1/10$. The expected theoretical frequency is $100 * 1/10$, equals 10. The following results are obtained:



i	O_i	E_i	$O_i - E_i$	$(O_i - E_i)^2/E_i$
0	8	10	-2	0,4
1	12	10	2	0,4
2	13	10	3	0,9
3	14	10	4	1,6
4	9	10	-1	0,1
5	6	10	-4	1,6
6	8	10	-2	0,4
7	7	10	-3	0,9
8	14	10	4	1,6
9	9	10	-1	0,1
Total	**100**	**100**		**8**

Since $\chi_c^2 = 8$ is lower than that of the table $\chi^2_{9,0.05} = 16.9$, we do not reject the hypothesis H_0 (that the generated sequence follows a uniform distribution in the interval [0,9]).
Pseudo-random number generators can also produce highly random sequences, empirical tests in this case give very low χ_c^2 values, or even close to 0. It is therefore recommended to perform a left test: H_0 is rejected if $\chi_c^2 < \chi^2_{(k-1,1-\alpha)}$. In this example, $\chi_c^2 = 8.0$ is not lower than that of the table $\chi^2_{(9,0.95)} = 3.32$, so we do not reject H_0.

Let's code!

```
#Code 603.py

from scipy.stats import chisquare
from scipy.stats import chi2

# chisquare: returns the calculeted Chi2 statistic
# chi2.ppf: returns the chi2 table value correspondent to alpha and df
def testChi2(data_obs, data_expected=None, alpha=0.05):
    df = len(data_obs) - 1
    Dc2 = chisquare(data_obs, data_expected) if data_expected else chisquare(data_obs)
    print('DChi2  :', Dc2[0])
    DA = chi2.ppf(1-alpha, df);                         print('Dalpha :', DA)
    return(Dc2[0]<DA)

# Test, loc is the sequence average and scale is its standard deviation
x = [8,12,13,14,9,6,8,7,14,9]
alpha = 0.05
print('H0     : ',testChi2(x))

#______________________________   Output   ______________________________________
# DChi2  : 8.0
# Dalpha : 16.918977604620448
# H0     :  True
```

6.3.2 Auto-correlation test

We consider a sequence $Y_1, Y_2, ...Y_n$ of random numbers. The serial auto-correlation coefficient (Pearson correlation coefficient) is defined by:

$$C = \frac{n(\sum_{i=1}^{n} Y_i Y_{i+1} + Y_n Y_1) - (\sum_{i=1}^{n} Y_i)^2}{n(\sum_{i=1}^{n} {Y_i}^2) - (\sum_{i=1}^{n} Y_i)^2}$$

It measures the dependence between two successive numbers. It always takes a value between -1 and $+1$. A value close to 0 means that the random numbers are independent, while a value close to -1 or $+1$ indicates a linear dependence between the numbers and therefore the absence of the randomness property.

Let's code!

```
# Code604.py

# Pearson correlation coefficient
def autocorrelation(data):
    n=len(data)
    nextData   = [data[i]*data[(i+1)%n] for i in range(n)]
    squareData = [data[i]**2 for i in range(n)]
    autocorrel = (n*sum(nextData) - sum(data)**2 ) / ( n*sum(squareData) - sum(data)**2)
    return autocorrel

data1 = [96, 929, 192, 833, 288, 737, 384, 641, 480, 545, 576, 449, 672, 353,
       768, 257, 864, 161, 960,65, 32, 993, 128, 897, 224, 801, 320, 705, 416,
       609, 512, 513, 608, 417, 704, 321, 800, 225,896, 129, 992, 33, 64, 961,
       160, 865, 256, 769, 352, 673, 448, 577, 544, 481, 640, 385, 736,289, 832,
       193, 928, 97, 0, 1, 96, 929, 192, 833, 288, 737, 384, 641, 480, 545, 576,
       449, 672,353, 768, 257, 864, 161, 960, 65, 32, 993, 128, 897, 224, 801,
       320, 705, 416, 609, 512, 513,608, 417, 704, 321, 800]
print('Data1 : ', autocorrelation(data1))

data2 = [100, 685, 232, 441, 652, 37, 592, 241, 308, 797, 56, 425, 92, 917, 672,
        993, 4, 397, 392, 921,44, 261, 240, 209, 212, 509, 216, 905, 508, 117,
        320, 961, 932, 109, 552, 377, 460, 485, 912,177, 116, 221, 376, 361,
        924, 341, 992, 929, 836, 845, 712, 857, 876, 709, 560, 145, 20, 957,
        536, 841, 316, 565, 640, 897, 740, 557, 872, 313, 268, 933, 208, 113,
        948, 669, 696, 297, 732,789, 288, 865, 644, 269, 8, 793, 684, 133, 880,
        81, 852, 381, 856, 777, 124, 1013, 960, 833,548, 1005, 168, 249, 76]
print('Data2 : ', autocorrelation(data2))

#______________________________   Output   ______________________________________
# Data1 : -0.6533458794659233
# Data2 : -0.023560372803231835
```

6.4 Random variables simulation

6.4.1 Simulating discrete r.v

6.4.1.1 Reverse transformation method

Suppose we want to generate the value of a discrete r.v X having the probability function:

$$P(X = x_j) = p_j \quad j = 0, 1, .. \text{ and } \sum_j p_j = 1$$

We need to generate a random number u uniformly distributed over $[0, 1]$ then we define:

$$X = \begin{cases} x_0 & \text{if } u < p_0 \\ x_1 & \text{if } p_0 \le u < p_0 + p_1 \\ ... & \\ x_j & \text{if } \sum\limits_{i=0}^{j-1} p_i \le u < \sum\limits_{i=0}^{j} p_i \\ ... & \end{cases}$$

And since for $0 < a < b < 1$, $p(a \le u < b) = b - a$, we have

$$P(X = x_j) = P(\sum_{i=0}^{j-1} p_i \le u < \sum_{i=0}^{j} p_i) = p_j$$

So, X has the required distribution.
Algorithmically this process results in:

Algorithm 2 Reverse transformation method for d.r.v

```
Input: $x_i, p_i \quad i \ge 0$
$u$ = RNG()
for $i = 0$ to $N$ do
    if $(u < \sum_{k=0}^{i} p_k)$ then
        return $x_k$
```

Where RNG() is the generator used to get a value from uniform distribution.
If the values of x_i, $i \ge 0$ are sorted in ascending order $x_0 < x_1 < x_2 <$..... and if F is the cumulative distribution function of X, then $F(x_i) = \sum\limits_{k=0}^{i} p_k$, and X will be equal to x_i if $F(x_{i-1}) \le u < F(x_i)$.

In other words, after generating a random number u, we determine the value of X by finding the interval $[F(x_{i-1}), F(x_i)]$ where u is situated (by finding the inverse of $F(u)$).
The required time to generate a discrete r.v using this method is proportional to the number of intervals that one must seek. For this reason, it is sometimes better to consider the possible values x_i of X in descending order of p_i

Example 3.
Simulate a d.r.v X such that: $p_1 = 0.20$, $p_2 = 0.15$, $p_3 = 0.25$, $p_4 = 0.40$
$p_j = P(X = j)$.
method 1.

1. *Generate u*
2. *If u<0.20 then, X=1.*
3. *If u<0.35 then, X=2.*
4. *If u<0.60 then, X=3.*
5. *otherwise X=4.*

A more efficient method is to consider the probabilities in descending order:

1. *Generate u*
2. *If u<0.40 then, X=4.*
3. *If u<0.65 then, X=3.*
4. *IF u<0.85 then, X=1.*
5. *Otherwise X=2.*

Let's code!

```
#Code 605.py

import random
from itertools import accumulate
from operator import itemgetter
#
def transformation_inverse(prob,indices):
    u = random.random()
    for i in range(len(prob)):
        if(u<prob[i]): return indices[i]
    return indices[-1]

# returns the frequencies in N generated rv
def generate(prob, N=100000):
    indices, prob = zip(*sorted(enumerate(prob), key=itemgetter(1),reverse=True))
    prob = list(accumulate(prob))
    freq = {k:0  for k in range(len(prob))}
    for _ in range(N):
        gen = transformation_inverse(prob,indices)
        freq[gen] += 1
    return {k:v/N for k,v in freq.items()}

# test
prob = [0.2,0.15,0.25,0.4]
print(generate(prob))

#______________________________   Output  ______________________________________
#{1: 0.19671, 2: 0.1498, 3: 0.25018, 4: 0.40331}
```

In case the r.v follows a discrete uniform distribution, it is not necessary to search for the appropriate interval where the random number is found. Indeed, if we want to generate the value of X which takes values of $1..n$:

$$P(X = j) = 1/n \qquad j = 1, ...n$$

Using the previous results, we can accomplish this by generating u and considering:

$$X = j \quad if \quad \frac{j-1}{n} \leq u < \frac{j}{n} \implies x = j \quad if \quad (j-1) \leq n \times u < j$$

In other words $X = \lfloor (n \times u) + 1 \rfloor$ such that $\lfloor x \rfloor$ is the integer part of x.

6.4.1.2 Acceptance-rejection method

Suppose we have an efficient method to simulate a r.v Y having the probability function $\{q_j, j \geq 0\}$, we can use it as a base to simulate a r.v X having the distribution $\{p_j, j \geq 0\}$ by simulating Y, then accept the obtained value with a probability proportional to $\frac{p_y}{q_y}$. This method is also called the *rejection* method.

Let c be a constant such that: $\frac{p_j}{q_j} \leq c, \forall j$. The rejection technique is as follows:

Algorithm 3 Reject method for d.r.v

```
Input: $c, p_j, q_j \quad \forall j$
while True do
    $y \leftarrow$ simulate the r.v $Y$ having the probability function q$_j$
    generate a random number $u$
    if $u < \frac{p_y}{cq_y}$ then
        return $y$
```

Example 4. *Consider the r.v X which takes values in $\{1, 2, 3, ..., 10\}$ with the respective probabilities: 0.11, 0.12, 0.09, 0.08, 0.12, 0.10, 0.09, 0.09,0.10, 0.10.*
It is preferable to use the rejection method with q the discrete uniform density; i.e: $q_j = 1/10$ $\forall j = 1, .., 10$. For this choice of q_j, we can take $C = Max\frac{p_j}{q_j} = 1.2$, and the simulation algorithm is as follows:

(1) generate a random number u_1, set, $y = Int(10u_1) + 1$
(2) generate a second random number u_2
(3) If $u_2 < \frac{p_y}{0.12}$ then $X = y$; otherwise go to (1).

The value 0.12 comes from the fact that $cq_j = 1.2/10$. This method takes on average of only 1.2 iterations to have the generated value of X.

Let's code!

```
#Code606.py

import random

# acceptance-rejection method
def acceptationRejet(pj, qj, genY, **p):
    c = max([pj[k]/qj[k] for k in pj.keys()])
    while (True):
        y = genY(qj, **p)
        u = random.random()
        if(u < pj[y]/(qj[y]*c)): return y
    return -1
#
def generate(pj,qj,N=100000):
    freq = {k:0  for k in pj.keys()}
    for _ in range(N):
        gen = acceptationRejet(pj, qj, lambda qi : int(random.random()*10)+1 , **{})
        freq[gen] += 1
    return {k:v/N for k,v in freq.items()}

# test
pj = {1:0.11,2:0.12,3:0.09,4:0.08,5:0.12,6:0.10,7:0.09,8:0.09,9:0.05,10:0.15}
qj = {1:0.1 ,2:0.1,3:0.1,4:0.1,5:0.1,6:0.1,7:0.1,8:0.1,9:0.1,10:0.1}
print(generate(pj,qj))

#______________________________   Output  ______________________________________
#{1: 0.10696, 2: 0.1202,  3: 0.09187, 4: 0.07928, 5: 0.11926, 6: 0.09927,
# 7: 0.09127, 8: 0.09055, 9: 0.0494, 10: 0.15194}
```

6.4.2 Simulating continuous r.v

6.4.2.1 Reverse transformation method

Consider a continuous r.v having the distribution function F. The inverse transformation method is based on the following proposition:

> **Proposition 3.**
> *Let U be a uniform r.v on $[0,1]$. For any continuous distribution function F, the r.v X defined as $X = F^{-1}(U)$ has distribution F. ($F^{-1}(U)$ is defined to be the value of x such that $F(x) = U$).*

Proof. Consider F_X the distribution function of $X = F^{-1}(U)$ (increasing and monotonic $a < b$ implies that $F(a) < F(b)$) so,

$$X \leq x \implies F^{-1}(U) \leq x \implies F(F^{-1}(U)) \leq F(x) \implies U \leq F(x)$$

then,

$$F_X(x) = P(X \leq x) = P(F^{-1}(U) \leq x) = P(U \leq F(x)) = F_U(F(x)) = F(x)$$

Since F is the distribution function then $F(x)$ is a function of x increasing and monotonous, thus $a \leq b$ is equivalent to $F(a) \leq F(b)$ so $F_X(x) = P(F(F^{-1}(U)) \leq F(x))$

$F_X(x) = P(U \leq F(x))$ because $F(F^{-1}(U)) = U$
$F_X(x) = F(x)$ because U is uniform on $[0,1]$

This proposition shows that we can generate a r.v X of a distribution function F by generating a random number u and set $X = F^{-1}(u)$

Algorithm 4 Reverse transformation method for c.r.v

```
Input: $F_X$
generate a random number $u \in [0, 1]$
return $F_X^{-1}(u)$
```

Example 5.
Use this method to generate a r.v with distribution function $F(x) = x^n$, $0 < x < 1$.

If we consider $x = F^{-1}(u)$ *then,* $u = F(x) = x^n$ *and equivalently,* $x = u^{1/n}$
We can therefore generate such r.v X *by simulating a uniform v.a* U *then set* $X = U^{1/n}$

Example 6. *If* X *is an exponential r.v of rate 1, then its distribution function is given by:* $F(x) = 1 - e^{-x}$
If $x = F^{-1}(u)$ *then* $u = F(x) = 1 - e^{-x}$
$1 - u = e^{-x} \implies x = -log(1 - u)$
We can therefore, simulate an exponential r.v of parameter 1 by simulating a uniform r.v U *and set:*

$$X = -log(1 - U)$$

We can replace $1 - U$ *by* U *because both are uniform r.v on [0,1].*
An exponential r.v X *of rate* λ *is simulated by simulating a uniform r.v* U *and set:*

$$X = -\frac{1}{\lambda} log(U)$$

Let's code!

```
#Code607.py

from sympy import symbols, solveset,S
from sympy.functions import exp
import random
import matplotlib.pyplot as plt
import numpy as np

#
def plot_fc(x,y):
    plt.plot(x,y)
    plt.show()

# to find the inverse of f we solve the equation y =f(x) using  solveset of sympy
def inverse(f):
    expr1 = solveset(f-y,x, domain=S.Reals)
    return(expr1.args[1].args[0])

#
def generate_inverse(f,l,N=1000):
    fc = inverse(f).subs(lamda,l)
    p = np.sort([random.random()  for _ in range(N)])
    r = [ fc.subs(y,u) for u in p]
    return {'x':r, 'y':p}

# symbols
x ,y ,lamda  = symbols('x ,y ,lamda')
if __name__== '__main__':
    f = 1-exp(-lamda*x)
    plot_fc(**generate_inverse(f,l))
```

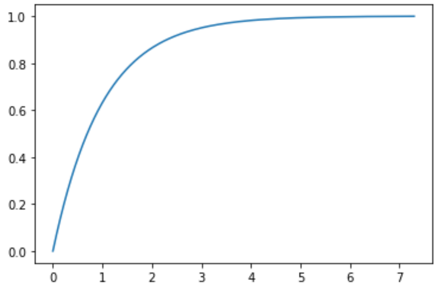


Figure 6.5: generated CDF

6.4.2.2 Acceptance-rejection method

The acceptance-rejection method is exactly the same as in the case of discrete r.v s, with the only difference that density functions replace mass functions.
Let X be a r.v of density f_X, Y another r.v of density f_Y and C a constant. We define $h = \frac{f_X}{f_Y}$. Consider the r.v U of uniform r.v on [0,1].

$$f_Y(x|U < \frac{h(Y)}{C}) = h(x)f_Y(x) = f_X(x)$$

Indeed:
$f_Y(x|U \le \frac{h(Y)}{C}) = \frac{P(U \le \frac{h(Y)}{C}|Y=x)f_Y(x)}{P(U \le \frac{h(Y)}{C})}$
$P(U \le \frac{h(Y)}{C}|Y = x) = P(U \le \frac{h(x)}{C}) = \frac{h(x)}{C}$
$P(U \le \frac{h(Y)}{C}) = \int_{\mathbb{R}} P(U \le \frac{h(Y)}{C}|Y = x)f_Y(x)dx = \int_{\mathbb{R}} \frac{h(x)}{C} f_Y(x)dx = \frac{1}{C}\int_{\mathbb{R}} f_X(x)dx = \frac{1}{C}$ So:

$$f_Y(x|U \le \frac{h(Y)}{C}) = f_X(x)$$

We set $C = max\{h(x)\}$ and we apply the following algorithm :

Algorithm 5 Rejection method for c.r.v

```
Input: $C, f_X, f_Y$
while True do
    generate $y$ of probability function $f_Y$
    generate a random number $u$
    if $u < \frac{f_X(y)}{Cf_Y(y)}$ then
        return $y$
```

The average number of iterations required for this method to simulate the r.v X equals to C. Since the stopping condition of the algorithm $u < \frac{f_X(y)}{Cf_Y(y)}$ depends on X and Y, its iterations number N is a r.v.
The algorithm is similar to a Bernoulli experiment such that a false condition is considered as a failure and a true condition as a success which ends the algorithm, N therefore follows the Geometric distribution of parameter $\frac{1}{C}$.
Indeed:

$$P(U \le \frac{f_X(Y)}{Cf_Y(Y)}) = \int_{-\infty}^{+\infty} P(U \le \frac{f_X(y)}{Cf_Y(y)})f_Y(y)dy = \int_{-\infty}^{+\infty} \frac{f_X(y)}{C}dy = \frac{1}{C}$$

So:

$$\mathbb{E}(N) = C$$

Example 7.
Let's use this method to generate a r.v having the density function:

$$f(x) = \frac{3}{4}x^2(2-x) \quad \textit{such that } 0 < x < 2$$

We consider the rejection method with:

$$g(x) = x/2, \quad 0 < x < 2$$

To determine the value of the constant C such that $\frac{f(x)}{g(x)} \leq C$, we must determine the maximum value of

$$h(x) = \frac{f(x)}{g(x)} = \frac{3}{2}x(2-x)$$

$$(\frac{f(x)}{g(x)})' = 3 - 3x = 0 \implies x = 1$$

$$\frac{f(x)}{g(x)} \leq h(1) = \frac{3}{2} = c$$

The average required number of iterations is $c = \frac{3}{2}$.

Let’s code!

```
#Code609.py

from sympy import diff, simplify
import random
import numpy as np
from sympy.solvers import solve
from Code607 import inverse, plot_fc, x, y

#
def reject(h, g, c):
    nb_iter = 0
    while True:
        v = inverse(g).subs(y,random.random())
        u = random.random()
        nb_iter += 1
        if u <= h.subs(x,v)/c: return v,nb_iter

#
def generate_reject(f,g,h,c,N=1000):
    res = [reject(h,g,c)  for _ in range(N)]
    freq = sum([v[1] for v in res])/N    ;print("average iterations number",freq)
    r = np.sort([v[0] for v in res])
    p = [ f.subs(x,u) for u in r]
    return {’x’:r, ’y’:p}

#symbols
f = (3/4)*x**2*(2-x)
g = (1/2)*x
h = f/g
hp = simplify(diff(h));  print("derivative:",hp)
sols = solve(hp); print("solutions:",sols)
c = h.subs(x,sols[0]); print("C:",c)

plot_fc(**generate_reject(f,g,h,c))

#______________________________   Output   ______________________________________
# derivative: 3.0 - 3.0*x
# solutions: [1.00000000000000]
# C: 1.50000000000000
# average iterations number 1.53
```

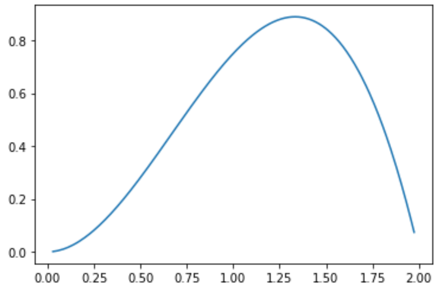


Figure 6.6: Generated PDF

6.5 Exercises

Exercise 1.
Give the first thirty numbers generated by the following RNGs (seed=1) :
1- $x_n = (5x_{n-1})[7]$
2- $x_n = (6x_{n-1} + 4)[11]$
3- $x_n = (x_{n-1} + x_{n-2})[9]$
4- $x_n = (2x_{n-1} + 3x_{n-2})[9]$

Exercise 2.
Consider the following sequence of random numbers:

$$92, 550, 238, 40, 110, 520, 615, 493, 369, 507$$

1.Using K-S test with a threshold $\alpha = 0.05$ *check if this sequence follows an exponential distribution with parameter* λ *equal to* $\frac{1}{e}$ *such that e is the empirical mean of the sample.*
2. By calculating the Pearson correlation coefficient, check if this sequence is random.

Exercise 3.
Consider the linear congruential generator of parameters a, m and seed $= x_0$*.*
1. Show that $x_n = a^n x_0[m]$*.*
2. What is the period of this generator? (use Euler's theorem: if m and a are positive integers prime to each other then the following relation holds: $a^{\phi(m)} = 1[m]$ *such that* ϕ *is Euler's function which gives the number of relatively prime numbers with m).*
3. What is the value of this period if m is prime? give its value for $m = 11$*.*
4. Show that the period of the RNG is $\frac{m}{4}$ *if* $m = 2^b$*,* $\forall\ b > 2$*.*

Exercise 4.
Consider the mixed congruential generator of parameters a,m and c.
1. Show that $x_{n+1} = (a^{n+1}x_0 + c\frac{a^n-1}{a-1})[m]$*.*
2. What is the period of this generator?

Exercise 5.
Two dice are rolled 150 times, the observed frequencies of the sum of the obtained values are given in the following table:

i	2	3	4	5	6	7	8	9	10	11	12
O_i	5	6	10	16	18	32	20	13	16	9	5

Test the following hypothesis by applying the χ^2 *test (*$\alpha = 0.05$*):*
H_0*: the two dice are fair.*
H_1*: the two dice are biased.*

Exercise 6.
Let X be a c.r.v having the density function $f_X(x) = 3x^2$ *for* $0 < x < 1$*. Using the inverse transformation method, write the algorithm that simulates this r.v.*

Exercise 7.
Consider the c.r.v. X of density $f(x) = 30x^2(1-x)^2$ *for* $0 \leq x \leq 1$
and Y of density $g(y) = 1$ *for* $0 \leq y \leq 1$*.*
Write the algorithm that simulates the r.v X by the rejection method.

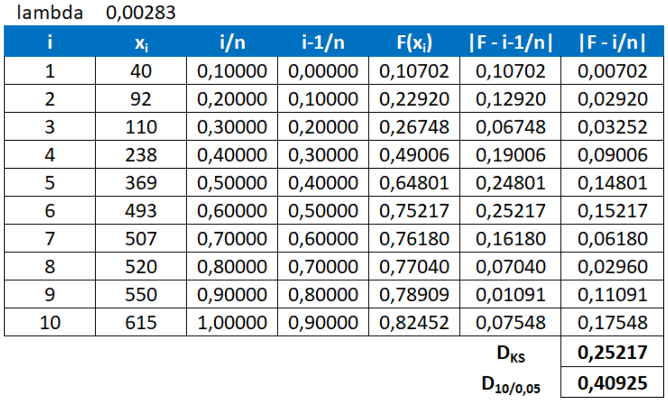
Exercise 8.
1. Simulate the generation of the values of a r.v X that follows the $\mathcal{B}in(5, 0.2)$ *distribution.*
2. Write the corresponding Python code.
3. Plot p_X*,* F_X *and* F_X^{-1}*.*

Exercise 9.
Consider the r.v X that follows the $\mathcal{G}eo(p)$ *distribution. Show that* $X = \lfloor \frac{ln(U)}{ln(1-p)} \rfloor + 1$

6.6 Solutions

Solution 1.
1. $[1, 5, 4, 6, 2, 3, 1, 5, 4, 6, 2, 3, 1, 5, 4, 6, 2, 3, 1, 5, 1, 5, 4, 6, 2, 3, 1, 5, 4, 6]$, *period=6.*
2. $[1, 10, 9, 3, 0, 4, 6, 7, 2, 5, 1, 10, 9, 3, 0, 4, 6, 7, 2, 5, 1, 10, 9, 3, 0, 4, 6, 7, 2, 5]$, *period=10.*
3. $[1, 1, 2, 3, 5, 8, 4, 3, 7, 1, 8, 0, 8, 8, 7, 6, 4, 1, 5, 6, 2, 8, 1, 0, 1, 1, 2, 3, 5, 8]$, *period= 24.*
3. $[1, 1, 5, 8, 7, 1, 8, 8, 4, 1, 2, 8, 1, 1, 5, 8, 7, 1, 8, 8, 4, 1, 2, 8, 1, 1, 5, 8, 7, 1]$, *period= 12.*

Solution 2.
$\lambda = 1/353.4$
$F(x) = 1 - e^{-\lambda x}$



lambda 0,00283

i	x_i	i/n	i-1/n	F(x_i)	|F - i-1/n|	|F - i/n|
1	40	0,10000	0,00000	0,10702	0,10702	0,00702
2	92	0,20000	0,10000	0,22920	0,12920	0,02920
3	110	0,30000	0,20000	0,26748	0,06748	0,03252
4	238	0,40000	0,30000	0,49006	0,19006	0,09006
5	369	0,50000	0,40000	0,64801	0,24801	0,14801
6	493	0,60000	0,50000	0,75217	0,25217	0,15217
7	507	0,70000	0,60000	0,76180	0,16180	0,06180
8	520	0,80000	0,70000	0,77040	0,07040	0,02960
9	550	0,90000	0,80000	0,78909	0,01091	0,11091
10	615	1,00000	0,90000	0,82452	0,07548	0,17548
					D_{KS}	**0,25217**
					D_{10/0,05}	**0,40925**

1- Since $D_{10,0.05} > D_{KS}$ *then the sequence follows the exponential distribution.*
2- The auto-correlation $C = 0.1014$*, so the sequence is random.*

Solution 3.
1. $x_{n+1} = ax_n[m] = a^2x_{n-1}[m] = a^3x_{n-2}[m] = ... = a^{n+1}x_0[m]$ *(by induction).*
2. Let p *be the period of this linear congruential generator.*
$x_{n+p} = a^{n+p}x_0[m] = x_n \implies a^p = 1[m]$ *so* $p = \phi(m)$ *according to Euler's theorem.*
3. Since m *is prime then,* $p = \phi(m) = m - 1$*,* $\phi(11) = 10$*. When* m *is prime, the period is maximal.*
4. $x_n = ax_{n-1}[2^b] = a^n x_0[2^b]$
We show by induction on b *that* $2^b/4$ *is the period:*
The induction hypothesis : $a^{\frac{2^b}{4}} = a^{2^{b-2}} = 1[2^b]$
We show that : $a^{\frac{2^{b+1}}{4}} = a^{2^{b-1}} = 1[2^{b+1}]$
$(a^{2^{b-2}})^2 = 1^2[2^b] = (k2^b + 1)^2 = k^2 2^{2b} + 2k2^b + 1$
$a^{2^{b-1}} = 2k2^b(k2^{b-1} + 1) + 1 = 1[2^{b+1}]$

Solution 4.
1.

$$\begin{aligned}
x_{n+1} &= (ax_n + c)[m] \\
&= (a^2x_{n-1} + ac + c)[m] \\
&= (a^3x_{n-2} + a^2c + ac + c)[m] \\
&= ... \\
&= (a^{n+1}x_0 + c\sum_{i=0}^{n} a^i)[m] \\
&= (a^{n+1}x_0 + c\frac{a^n - 1}{a - 1})[m]
\end{aligned}$$

2. Let p *be the period of* $x_{n+1} = a^{n+1}x_0[m]$
$x_{n+p} = (a^{n+p}x_0 + c\frac{a^{n+p-1}-1}{a-1})[m] = x_n$
p is also the period of the mixed congruential generator.

Solution 5.

i	2	3	4	5	6	7	8	9	10	11	12
O$_i$	5	6	10	16	18	32	20	13	16	9	5
p$_i$	$\frac{1}{36}$	$\frac{1}{18}$	$\frac{1}{12}$	$\frac{1}{9}$	$\frac{5}{36}$	$\frac{1}{6}$	$\frac{5}{36}$	$\frac{1}{9}$	$\frac{1}{12}$	$\frac{1}{18}$	$\frac{1}{36}$
E$_i$	4	8	12	16	20	24	20	16	12	8	4
$\frac{(O_i-E_i)^2}{E_i}$	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{3}$	0	$\frac{1}{5}$	$\frac{8}{3}$	0	$\frac{9}{16}$	$\frac{4}{3}$	$\frac{1}{8}$	$\frac{1}{4}$

$\chi^2_c = \sum_{i=1}^{n} \frac{(O_i-E_i)^2}{E_i} = 6.22083$
The value $\chi^2_{10,0.05} = 3.94 < \chi^2_c$*, so the hypothese* H_0 *is rejected.*

Solution 6.
$f_X(x) = 3x^2$*, so* $F_X(x) = x^3$ *for* $0 < x < 1$
$F_X^{-1}(x) = x^{1/3}$*, therefore, the simulation algorithm is as follows:*
1. generate a random number $u \in [0, 1]$.
2. return $F_X^{-1}(u)$.

Solution 7.
Let $h(x) = \frac{f(x)}{g(x)}$ *and* $C = max\{h(x)\}$.
$C = \frac{15}{8}$*. The simulation algorithm is as follows:*
1. generate y of function g.
2. generate a random number $u \in [0, 1]$.
3. if $u < \frac{h(u)}{c}$ *then return y else goto step 1.*

Solution 8.

$$p_X(k) = C_n^k p^k (1-p)^{n-k}$$

$$X = F_X^{-1}(U) = min\{x|F_X(x) < U\}$$

X	0	1	2	3	4	5
$F_X(x)$	0.32768	0.73728	0.94208	0.99328	0.99968	1.000

1. generate u
2. If u < 0.32768 then, X=0.
3. If u < 0.73728 then, X=1.
4. If u < 0.94208 then, X=2.
5. If u < 0.99328 then, X=3.
5. If u < 0.99968 then, X=4.
6. Otherwise X=5.

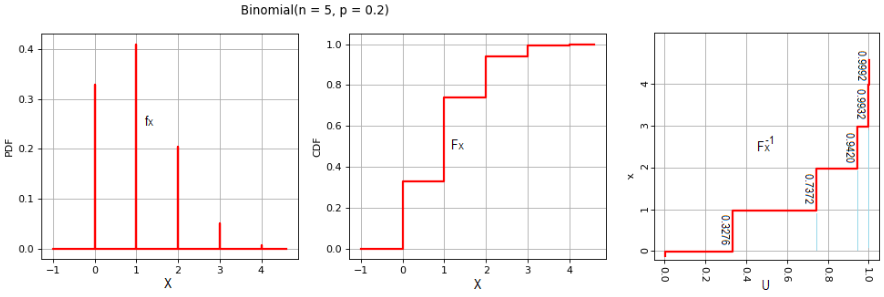


Figure 6.7: Binomial simulation

Solution 9.

$$\begin{array}{rcccl}
F_X(k-1) & < & U & \leq & F_X(k) \\
\sum_{n=0}^{k-1}(1-p)^{k-2}p & < & U & \leq & \sum_{n=0}^{k}(1-p)^{k-1}p \\
p\frac{1-(1-p)^{k-1}}{1-(1-p)} & < & U & \leq & p\frac{1-(1-p)^{k}}{1-(1-p)} \\
1-(1-p)^{k-1} & < & U & \leq & 1-(1-p)^{k} \\
(1-p)^{k} & \leq & 1-U & < & (1-p)^{k-1} \\
kln(1-p) & \leq & ln(1-U) & < & (k-1)ln(1-p) \\
k-1 & < & \frac{ln(1-U)}{ln(1-p)} & \leq & k
\end{array}$$

$So\ X = \lfloor \frac{ln(1-U)}{ln(1-p)} \rfloor + 1$

Chapter 7

Simulation in Python

7.1 Introduction

In *Python* different simulation packages exist, they allow to simulate predefined random experiments such as the coin toss or dice roll, the drawing of cards, balls from the urns as well as user-defined experiments. They also allow to define and simulate random variables with their distributions, random processes and the graphical representation of their outcomes. *Simpy* package for instance offers the possibility of simulating systems by modeling their components in the form of processes and shared resources. In what follows we present the APIs of these libraries as well as codes to show their use in simulation.

7.2 Simulation tools

7.2.1 Python

Python is an *interpreted* programming language that promotes structured imperative, functional and object-oriented programming (*multi-paradigm*) and it is *dynamic typing*, open source and multi-platform. It is appreciated for the simplicity of its syntax. In 1991, the first version of Python language was released by its creator *Guido van Rossum*. Currently it is maintained by Python Software Foundation under its own license (PSF License). Python community has adopted a code writing style convention that is standardized as *PEP8*. From 2009 and in order to eliminate the weaknesses of the 2.x versions of the language, version 3 was introduced without keeping compatibility with its previous one. The most recent *version 3.11.0a5* was released in February 2022. The most used IDEs for Python development are VS code, Pycharm, Jupyter, Spyder, PyDev and Eclipse.

In addition to the IDE, and in order to boost the productivity of developers, Python integrates two utilities that facilitate the development process which are Pip and venv (virtual environment). *Pip* is a package manager that allows you to manage the dependencies of software projects. As Python is used in many systems as a scripting language for administration tasks, when using it for development it is best to isolate it from the rest of the system, a virtual environment is created to ensure this isolation by creating a controlled environment (with dependencies specific to the current project) using *venv* module.

7.2.1.1 Python Standard Library

Python Standard Library (PSL: Python Standard Library) gathers a collection of packages and modules that implement the core of the language and the basic libraries (Integrated elements (*Functions, types, Exceptions*), *String, Math, Functional, IO, Parallelism, Internet Networks , XML JSON, I18, ..*). In what follows we will expose the modules that serve as support for statistics and simulation packages.

Random module

random generates the pseudo-random numbers by implementing the generators of different distributions and according to the desired types (integer, sequence or real). Python uses the

Mersenne Twister algorithm as a base generator whose period is $2^{19937}-1$ and 53 bits of precision for real numbers. *random()* function plays the role of a PRNG of the c.r.v X of uniform distribution $[0.0, 1.0[$, it is used by the other functions for the generation of other distributions.

Pyhthon 3.9/Lib/random.py:class Random(_random.Random)

1-Core	
init(x=None)	Random constructor, x is the seed.
seed(a=None, version=2)	Initializes the random number generator.
get/setState([value])	Returns an object capturing the internal state of the generator/modifying it.
random()	Returns a random floating point number in the range [0,1[.

2-Discrete generators	
randbytes(n)	Generate n random bytes.
randrange([b,] s[, p])	Returns a randomly selected element from range (b:start, s:stop, p:step).
randint(a,b)	Returns an integer N such that $a \leq N \leq b$. Alias for randrange (a,b+1).
getrandbits(k)	Returns a non-negative integer of k random bits
choice(seq)	Returns an element of sequence seq.
choices(pop, weights=Nn, *, cum_weights=Nn, k=1)	Returns a list of k items chosen from population p with discount.
sample(pop, k,*, counts=Nn)	Returns a list of k unique items in the population without replacement.
shuffle(x, random=Nn)	Shuffle the sequence x without creating a new instance.

3- Continuous generators	
distribution(parameters)	which are: betavariate(alpha, beta); expovariate(lambd); gammavariate(alpha, beta); gauss(mu, sigma); lognormvariate(mu, sigma); normalvariate(mu, sigma); paretovariate(alpha); triangular(low=0.0, high=1.0, mode=None); uniform(a,b); vonmisesvariate(mu, kappa)

Itertools Module

Itertools contains functions to create iterators for efficient loops. Some are widely used in the field of probability, statistics and combinatorial calculus:

Pyhthon3.9/Lib/itertools.py

1- Finite iterators	
accumulate(p [,func])	returns $p_0, p_0+p_1, p_0+p_1+p_2, \cdots$ (ϕ([1,2,3])=1 3 6).
chain(p, q,..)	returns $p_0, p_1, \cdots, p_l, q_0, q_1, \cdots$, (ϕ(ABC, DE)=ABCDE
compress(d,s)	d:data, s:selectors, return $(d_i \text{ if } s_i)\ _{i\geq 0}$, ($\phi$(ABCD,[1,0,1,0])=AC
dropwhile(pred, seq)	returns seq[n], seq[n+1], starting when pred fails, (ϕ(lambda x: x<5, [1,4,6,4,1])=6 4 1
filterfalse(pred, seq)	returns elements of seq for which pred(elem) is false, (ϕ(lambda x: x % 2, range(10))=0 2 4 6 8
groupby(iteraable [,key])	returns sub-iterators grouped by the value of key(v)
islice(seq, [b,] s [,p])	returns elements of seq [b:start, s:stop, p:], (ϕ(ABCDEF,2,Nn)=C D E F
startmap(func, seq)	returns func(*seq [0]), func(*seq[1]),...,(ϕ(pow,[(2,5),(3,2)])=32 9
takewhile(pre, seq)	returns seq[0], seq[1], until pred fails, (ϕ(lambda x: x <5, [1,4,6,4])=1 4
tee(it, n)	returns $it_1, it_2, \cdots, it_n$ separates an iterator into n
zip_longuest(p, q,..)	returns $(p_i,\ q_i)\ _{i\geq 0}$, $(\phi(ABCD, xy, fillvalue=-) = AxByC-D-$

2- Infinite iterators	
hline count(start [,step])	start, start + step, start + 2 * step, (phi (10) = 10, 11, 12, 13, 14 ...)
cycle(p)	$p_0, p_1, \cdots p_{last}, p_0, p_1 \cdots$,$(\phi('ABCD') = ABCDABCD...)$
repeat(e [,n])	$e, e, e, \cdots$ to ∞ or up to n times, $(\phi(8,3) = 888)$

3- Combinatorial iterators	
hline product(p, q,...[repeat=1])	Cartesian product,$(\phi(AB, repeat=2) = AA, AB, BA, BB$
permutations(p[, r])	Arrangements of n elements among r, without replacement. $\phi(AB,2) = AB, BA$
combinations(p,r)	tuples of length r, unordered, without repetition. $\phi(AB,2) = AB$
combinations_with _replacement(p, r)	n-tuples of length r, ordered, with repetition. $\phi(AB,2) = AA, AB, BB$

7.2.1.2 Python's Ecosystem

Numpy module

numpy is an open source library that allows to manipulate multidimensional structures (matrix, vector) based on the *ndarray* structure. Version 1.22.0 was released in december 2021.

Numpy 1.20/Lib/core.py

1- Methods

all/any([x, o, k, w])	Returns True if all/at least one element equals True.
argmax/min([x, o])	Returns the indices of the maximum/minimum values on the given axis.
astype(t [, s, cast, subok, cp])	Returns a copy of the array converted to the specified type.
copy([order])	Returns a copy of the array.
cumprod/sum([x, t, o])	Returns the cumulative product/sum of the elements of the given axis.
diagonal([offset, x_1, x_2])	Returns the specified diagonals.
fill(value)	Fill the array with a scalar.
flatten([order])	Returns a copy of the one-dimensional projected array.
max/min([x, o, k, init, w])	Returns the maximum/minimum of the given axis.
mean([x, t, o, k, w])	Returns the average of the array elements on the given axis.
nonzero ()	Returns the indices of nonzero elements.
prod/sum([x, t, o, k, init, w])	Returns the product/sum of the array elements on the given axis.
ravel([S])	Returns the flattened array.
repeat(repeats[, X])	Repeat the elements of an array.
reshape(shape [, S])	Resize an array.
resize(newshape [,refcheck])	Change the dimension and size of an array.
round([decimals, out])	Each item is rounded to the given number of decimals.
sort([X, kind, S])	Sort an array.
std/var([x, t, o, ddof, k, w])	Returns the standard deviation/variance of the elements of an array on the given axis.
tolist()	Turn an array into a nested list.
tostring([s])	A tobytes compatibility alias.
trace([offset, x_1, x_2, t, out])	Returns the sum of the diagonals of an array.
transpose(*axes)	Returns a view of the array with the axes transposed.

2- Attributes

T: ndarray	Transposed array.	data: buffer	Pointer to the start of the array.
dtype: dtype	Data type of the elements.	flags: dict	Information on memory occupation.
flat: flatiter	A 1-D iterator of the array.	imag: ndarray	Imaginary part of the array.
real: ndarray	Real part of the array.	size: int	Size of the array.
itemsize: int	Size of a single item in bytes.	nbytes: int	Size of the array in bytes.
ndim: int	Number of array dimensions.	shape: tuple	Tuple of the array dimensions.

Numpy 1.16, Numpy.random

1- Sampling

rand($\mathrm{d}_0, d_1, \cdots, \mathrm{d}_n$)	Random values of the given form.
randn($\mathrm{d}_0, d_1, \cdots, \mathrm{d}_n$)	Generate a sample(s) of the standard normal distribution.
randint(lw [,hi, sz, dtp])	Generate random integers between lw (inclusive) and hi (exclusive).
random_integers(lw [, hi, sz])	Random integers of type np.int between low and high, inclusive.
random_sample([sz])	Return random floats in [0.0, 1.0[.
random([sz])	Return random floats in [0.0, 1.0[.
ranf([sz])	Return random floats in [0.0, 1.0[.
sample([sz])	Return random floats in [0.0, 1.0[.
choice(a [, sz, replace, p])	Generate a random sample of the given 1D array.
bytes(length)	Returns random bytes.
shuffle(x)	Modify a sequence by shuffling its contents.
permutation(x)	Randomly permute a sequence.

3- Distributions

distribution(parameters)	beta(a,b), binomial(n,p), chisquare(df), dirichlet(alpha), exponential() sc, f(dfnum, dfden), gamma(sh) sc, geometric(p), gumbel() lc, sc, hypergeometric(ngood, nbad, nsample), laplace() lc sc, logistic() lc sc, lognormal([mean, sigma]), multinomial(n, pvals), normal() lc sc, pareto(a), fish([lam]) triangular(left, mode, right), uniform([low, high])

All distribution functions take as optional parameters *size*, *local* and *scale* (denoted respectively lc, sc, sz). This module also contains the functions of the generator (seed initialization ([s_0]) and management of its state RandomState([s_0]), get/set([state])).

Sympy module
sympy is a library designed for symbolic (formal) calculus. It is used in arithmetic, algebraic, differential calculus as well as other fields such as quantum mechanics.
sympy.stats.

1- Methods

class Probability(prob, condition=None, **kw)	The symbolic expression of the probability.
class Expectation(expr, condition = Nn, ** kw)	The symbolic expression of the expectation.
class Variance(arg, condition=Nn, **kw)	Symbolic expression of the variance.
class Covariance (arg1, arg2, condition=Nn, **kw)	Symbolic expression of the covariance.
JointRV(symbol, pdf, _set=Nn)	Creates a joint r.v with each continuous component.
marginal_distribution(rv, *indices)	Marginal distribution function of a joint r.v.
P(condition, given_condition=Nn, numsamples=Nn, evaluate=True, **kw)	Probability that a condition is true knowing another condition.
E(expr, condition=None, numsamples=Nn, evaluate=True, **kw)	The expectation of a random expression.
density (expr, condition=Nn, evaluate=True, numsamples=Nn, **kw)	The density of a random expression knowing a second condition.
given (expr, condition=Nn, **kw)	Conditions a random expression.
where (condition, given_condition=Nn, **kw)	Returns the domain where a condition is true.
variance(X, condition=Nn, **kw)	Variance of a random expression.
covariance(X, Y, condition=Nn, **kw)	Covariance expression of two r.vs.
median(X, evaluate=True, **kw)	Calculate the median of the probability distribution.
std(X, condition=Nn, **kw)	Standard deviation of a random expression.
sample(expr, condition=Nn, size=(), library ='scipy', numsamples=1, seed=Nn, **kw)	A realization of the random expression.
correlation(X, Y, condition=Nn, ** kw)	Correlation of two random expressions.

SciPy module
sciPy is a library for scientific computing based on numpy structures. Its modules are very diverse including linear algebra, optimization, image and signal processing, ... The most recent version is 1.8.0 released in February 2022. stats module contains a collection of statistical routines which can be classified according to its function into one of the following categories:

a- Generic functions :
  - *rv_ continuous([momtype, a, b, xtol, . . .])*: A generic c.r.v.
  - *rv_ discrete([a, b, name, badvalue, . . .])*: A generic d.r.v.
  - *rv_ histogram(histogram, *args, **kwargs)*: Generate a distribution.

b- Continuous/discrete distributions
It contains almost all the usual distributions. These functions take the same form of parameter definition, the collection of positional parameters and those of named parameters. (*args, **kwargs).
c- Multivariate distributions
d- Statistics and correlation functions.
e- Descriptive statistics.
f- Statistics tests

7.2.2 Symbulate

Symbulate is a framework for the simulation of probability models. It allows us to specify, run, analyze and view the simulation results. The most important functions serve to:

Symbulate

2- Distributions

discrete(parameters)	Bernouli(p), Binomial(n,p), Hypergeometric(n,N0,N1), Geometric(p), NegativeBinomial(), Pascal(r,p), Poisson(lam), DiscretUniform (a=0, b=1p)
continuous(parameters)	Uniform(a=0.0, b=1.0]), Normal(mean=0.0, sd=1.0, var=None), Exponential(rate=1.0, scale=Nn), Gamma(shape, rate=1.0 , scale=Nn), Beta(a,b), StudentT(df), chisquare(df), F(dfN,dfD), Lognormal([mean,sigma]), Pareto(b=1.0, scale=1.0), Triangular(left,mode,right),
mltivariate(parameters)	MultivariateNormal([mean,cov]), MultiNormal([n,p])

3- Processes

Gaussian	GaussianProcessProbabilitySpace(mean$_func, cov_func, index_set = Reals()); GaussianProcess(mean_func, cov_func, index_set = Reals())$
Brownian	BrownianProcessProbabilitySpace(drift=0, scale=1), BrownianProcess(drift=0, scale=1)
Poisson	ProcessProbabilitySpace(rate), PoissonProcess(rate)
Markov	MarckovChainProbabilitySpace(transiton_matrix, initial_dist, state_labels=None), MarckovChain(transiton_matrix, initial_dist, state_labels=None), ContinuousTimeMarckovChainProbabilitySpace(generator_matrix, initial_dist, state_labels = None), ContinuousTimeMarckovChain(generator_matrix, initial_dist, state_labels = None),

1. *Define probability spaces*: This is possible by using BoxModel() and draw() which allows to simulate a draw of the BoxModel or by writing functions specifying them. It is also possible to create independent probability spaces by multiplying two probability spaces.

2. *Define random variables* on already defined sample spaces using the *RV* class. To define a *g* function of a random variable *X* we can use *X.apply(g)* method. If *g* is a known mathematical function (log, exp, sin, ...), it can be called directly without using apply: *g(X)*. Other very useful functions exist like: *mean()* to calculate the expectation of the r.v simulations, *sd()* to calculate the standard deviation and *standardize()* for the normalization of the r.v (expectation = 0 and variance = 1).

3. *Use the most common probability models*: whether it is discrete, continuous, joint distributions or even some random processes, *Symbulate* offers a rich and easy-to-use set of methods. Methods such as: *DiscreteUniform()*, *Bernoulli()*, *Binomial()*, *Hypergeometric()*, *Poisson()*, *Geometric()* as well as other methods are available for discrete distributions; and it is always possible to define your own discrete distribution using the BoxModel.
   The same is possible for continuous distributions with the methods: *Uniform()*, *Normal()*, *LogNormal()*, *Exponential()*, *Gamma()*,...
   For both types of distributions, very useful methods can be used: *pdf(x)* returns the value *f(x) (density function)*, *cdf(x)* returns the cumulative function *F(x)*, *mean(), var(), sd(),median() and quantile()* returns respectively the values of the expectation, the variance, the standard deviation, the median and the quantiles.
   On the other hand, *MarkovChain(TransitionMatrix, InitialDistribution, states)* method allows to create a discrete MC and to effectively simulate it with *sim()*. The same is possible for continuous MCs using *ContinuousTimeMarkovChain(Q, InitialDistribution, states)*, and the Poisson process of a given transition rate with *PoissonProcess(rate = 2)* method.

   Conditional probabilities are possible with Symbulate by using the sign "|" (given that) and that can be applied to r.vs as well as events.
   Another *Symbulate* tool that should not be overlooked is the *plot()* graphical tool which allows us to visualize the simulation results in several possible forms.
   When the number of simulations is small, we can visualize the individual obtained values with the parameter *type = "rug"*. By default *plot()* displays the outcomes with their relative frequencies. We can also display the outcomes as a histogram or as a density (continuous), scatter plot and other custom types.

4. *Simulate*: the same simulation tools can simulate sample spaces, random variables and stochastic processes.

The already mentioned *draw()* method makes it possible to simulate a single outcome of a sample space; to simulate several ones, *sim()* method is used by specifying the desired number of simulations. *apply()* method is used to apply a given function to each outcome of the simulation. It is also possible to define a method written in Python and pass it as a parameter to *apply()* method.
tabulate() method presents in a tabular form the outcomes of the simulation by counting the number of appearances of each outcome. With the parameter *normalize* = *True*, the number of occurrences is transformed into probability. To get a subset of simulations equal to a particular outcome, the *filter_ eq ()* method can be used.

Let’s code!

```
#code701.py

from symbulate import BoxModel

# Die sampling
# this line could be replaced by :list(range(1, 6+1))
die = [1, 2, 3, 4, 5, 6]
roll = BoxModel(die)
print(roll.draw())

# draws 10 samples from [’A’, ’B’, ’C’, ’D’] with probabilities [0.22, 0.27, 0.41, 0.10]
print(’’.join(list(BoxModel([’A’, ’B’, ’C’, ’D’], probs=[0.22, 0.27, 0.41, 0.10], size=10).draw())
     ))

#______________________________   Output  ______________________________________
# 2
# CACCCCBAAB
```

In several basic situations, the probability space can be defined by a *BoxModel*. *draw()* method is used to simulate a BoxModel draw.
BoxModel options are:

box: a list of *tickets* used for the draw.
size: the number of tickets to be drawn from the box.
replace: *True* if the draw is made with replacement and False otherwise.
probs: ticket selection probabilities. By default they are equally likely.
order_ matters: *True* if the order is important and *False* otherwise.

Another way to define the probability model is to define a function allowing to describe how the draw should be done, then give this function as a parameter to the *ProbabilitySpace*() method as described in the following example:

Let’s code!

```
#code702.py

from symbulate import BoxModel, ProbabilitySpace, Poisson, Exponential

#1 probabaility space with user defined function
def gender_hobbies_sim():
    gender = BoxModel(["male", "female"], probs=[.2, .8]).draw()
    if gender == "male":
        hobbies = BoxModel(["science", "art", "sport"], probs=[.3, .3, .4]).draw()
    else:
        hobbies = BoxModel(["science", "art", "sport"], probs=[.2, .5, .3]).draw()
    return gender, hobbies
P = ProbabilitySpace(gender_hobbies_sim)
print(P.draw())

#2 independants probability spaces
die6, die4 = list(range(1,7,1)), list((1,5,1))
lancers = BoxModel(die6)*BoxModel(die4)
print(lancers.draw())

#3 independants probability spaces
three = BoxModel([’P’,’F’])*Poisson(lam=3)*Exponential(rate=4)
print(three.draw())

#______________________________   Output  ______________________________________
```

```
#('female', 'art')
#(5, 1)
#(F, 1, 0.029286540827854368)
```

7.2.2.1 Random variable Simulation

Let's code!

The following example shows the definition of a r.v from a BoxModel, its simulation with *sim()* method, specifying the number of simulations in parameters, as well as the tabular representation of the outcome using the method *tabulate()*.

```
#Code703.py

from symbulate import BoxModel, RV, Normal, exp

# 5 Bernoulli trials and successes number
P = BoxModel([0,1],size=5); X = RV(P,sum)
tab = (X>3).sim(10000).tabulate()
print("Tabulate without normalization: ",tab)
tab = (X>3).sim(10000).tabulate(normalize=True)
print("Tabulate with normalization: ",tab)
X.sim(10000).plot()

# function of rv
X = RV(Normal(mean=0, var=1)); Y = exp(X) # could be replaces with: X.apply(exp)
Y.sim(10000).plot()

#______________________________   Output  ______________________________________
# Tabulate without normalization:  {True: 1826, False: 8174}
# Tabulate awith normalization:  {True: 0.1842, False: 0.8158}
```

The r.v X in this example can be considered as the number of tails obtained in 5 successive tosses of a coin. Then 10,000 simulations are performed to simulate $X > 0$. We should note here that the simulation is repeated as many times as necessary to obtain 10,000 results such as $X > 3$ and not (as one might think) of 10,000 simulation we only keep those with $X > 3$. The use of *normalize* = *True* in the method allows to have the results in the form of frequencies. The graphic representation is also possible with the *plot()* method.

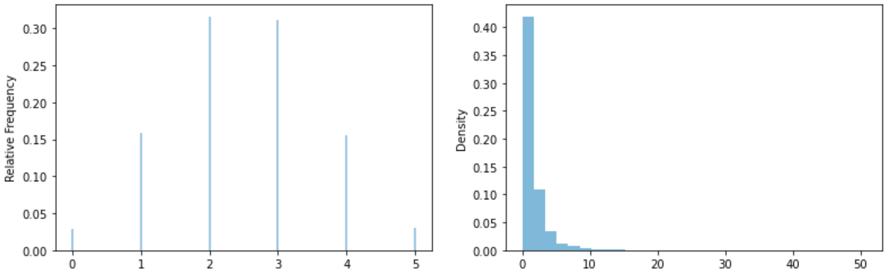


Figure 7.1: Simulation of Normal distribution (left) and Exponential distribution (right) (code703.py)

Let's code!

```
#Code704.py

from symbulate import RV, Normal, Exponential

X = RV(Exponential(rate=1/4))
simX = X.sim(10000)
print("Expected value:", simX.mean())
print("Variance:", simX.var())
print("Standard deviation:", simX.sd())

#Normalization
X = RV(Normal(mean=3,sd=2))
simX = X.sim(10000)
print("Before normalization. mean and sd:", round(simX.mean(),2), round(simX.sd(),2))
simX.plot()

z = simX.standardize()
```

```
print("After normalization. mean and sd:", round(z.mean(),2), round(z.sd(),2))
z.plot()

#______________________________   Output  ______________________________________
# Expected value: 3.9628993353781383
# Variance: 15.587130335771537
# Standard deviation: 3.9480539935228265
# Before normalization. mean and sd: 3.03 2.0
# After normalization. mean and sd: -0.0 1.0
```

In this example the mean, the variance and the standard deviation are calculated from the outcomes of the simulation and not theoretically.
Normalization consists of modifying the r.v so that its expectation is 0 and its standard deviation is 1. The modification consists in subtracting the expectation from each value and dividing it by the standard deviation. With *symbulate*, this can be done by the *standardize()* method which normalizes the values obtained by the simulation.

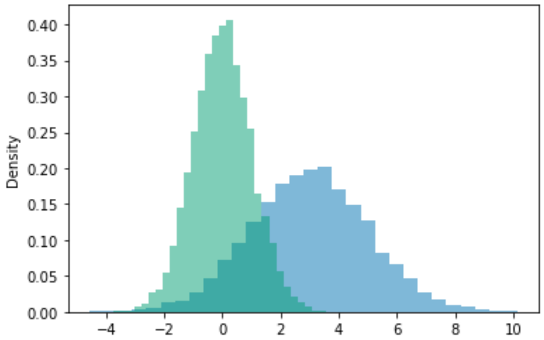


Figure 7.2: Normal distribution before vs after normalisation (code704.py)

7.2.3 Simpy

Simpy is a discrete event simulation Python framework based on processes. *processes* are created by *environments* using generator functions (returning an iterator) and can model active components like clients, vehicles or agents. Simpy also offers the concept of shared *resources* allowing to model the interactions between processes; they represent congestion points where processes must wait in a queue to use them. In Simpy, there is also the notion of *events* which is used to process different types of events in the simulator. Exceptions are used to synchronize the different processes.

7.2.3.1 Environments

Belonging to *simpy.core* package (the core of Simpy components), *BaseEnvironment* is the base class which mainly allows us to manage simulation time, schedule events and process them. Among the main methods:

simpy.core

1- API core	
C: BaseEnvironment	Base class for event processing environments.
C: BoundClass	Allows classes to behave like methods.
C: Environment<BaseEnvironment>	A runtime environment.
C: StopSimulation<Exception>	Stops the simulation.
a: now	The current time of the environment.
a: active_process	The currently active process in the environment.
m: schedule(ev, priority=NORMAL, delay=0)	Schedules an event with the given priority and delay.
m: step()	Processes the next event.
m: run(until=Nn)	Executes step() until the criterion is met.
m: peek()	Returns the time of the next scheduled event.

7.2.3.2 Events

The main events are managed by *simpy.events* module. *Event* is the base class for all event types:
Events

1- API events	
C: Event(env)	An event that can occur at a given point in time.
C: Timeout<Event>(env, delay, value=Nn)	An event that is triggered after an elapsed time.
C: Initialize<Event>(env, p)	Initialize a process p.
C: Interrupt<Event>(process, cause)	Immediately schedule an interrupt.
C: Process<Event>(env, generator)	Process a source generator of an event.
C: ConditionValue(object)	The result of a Condition.
C: Condition<Event>(env, evaluate, evts)	An event that is raised after the function of the condition returns True.
C: AllOf<Condition>(env,events)	The event that is launched if all the events in a list have been triggered.
C: AnyOf<Condition>(env,events)	The event that is fired if at least one of the events in the list has been fired.
C: Interrupt<Exception>	The exception thrown to the process when it is interrupted.
p: triggered()	True if the event has been triggered and its function is about to be invoked.
p: processed()	True if the event has been processed.
p: ok()	True if the event was started successfully.
p: defused([value])	True if the exception of a failed event has been "defused".
p: target()	The event the process is waiting for.
p: is_alive()	True until the generator quits.
p: value()	The value of the event if it is available.
m: trigger(event)	Trigger the event with the given state.
m: succeed(value=Nn)	Modify the value of the successful event.
m: fail(exception)	Modify the value of the failed event.
m: cause()	The cause of the interrupt.
m: interrupt(cause=Nn)	Interrupt this process by returning a cause.
sm: all_events(ev, count)	A function that returns "True" if all * events * have been started.
sm: any_events(evts, count)	A function that returns "True" if at least one of the * events * has been fired.

7.2.3.3 Shared resources

Three types of resources can be used to synchronize processes with Simpy in the *simpy.resources* module using the following three classes:
1- *resource*: for resources that allow priorities and preemption.
2- *container*: for sharing of homogeneous resources (discrete or continuous).
3- *store*: to store an unlimited number of items. Allows selective queries on specific objects.
Resources

1- API resources	
C: Put/Get<Event>(resource)	Generic event for a request to drop/pull an entity into/from the resource buffer.
m: cancel(self)	Cancel the put/get request.
C: BaseResource<object>(e, cap)	Abstract base class of the shared resource.
Put/GetQueue = list	Type of the put/get queue.
p: capacity(self)	The maximum capacity of the resource.
put = BoundClass(Put)	Request to put an object in the resource and return a Put event raised once the request is satisfied.
get = BoundClass(Get)	Request to remove an object from the resource and return a Get event triggered when the request is satisfied.
C: Preempted(by, usage_since, resource)	Cause a preemption, events.Interrupt contains information about the preemption.
C: Request<base.Put>(exc_type, value, traceback)	Request the use of the resource. The event is raised after access is granted.
C: Release<base.Get>(resource, request)	Release the use of the resource granted on request. This event is triggered immediately.
C: PriorityRequest<Request>(rsc, priority=0, preempt=True)	Request the use of the resource with a given priority.
C: SortedQueue(maxlen=None)	Queue to sort events by their PriorityRequest.key attribute.
m: append(self, item)	Sort the item in the queue. Raise a RuntimeError if the queue is full.
C: Resource<BaseResource> (env, capacity=1)	Resource with capacity that can be requested by processes.
p: def count(self)	Number of users currently using the resource. returns len (self.users).
request = BoundClass(Request)	Request a usage location.
release = BoundClass(Release)	Release a usage slot.
C: PriorityResource<Resource>(env, capacity=1)	A resource supporting priority requests.
PutQueue = SortedQueue	Type of the drop queue.
GetQueue = list	Type of the withdrawal queue.
request = BoundClass(PriorityRequest)	Request a location with the given priority.
release = BoundClass(Release)	Release a slot.
C: PreemptiveResource<PriorityResource>	A Resource with priority and preemptive.

Containers

1- Containers API	
C: ContainerPut/Get<base.Put/Get>(c, a)	Ask to put/get quantity a in container c.
C: Container<base.BaseResource>(env, capacity=float('inf'), init=0)	Limited capacity resource containing a continuous or discrete entity.
p: level()	The current quantity available.
a: put/get = BoundClass<ContainerPut>	Ask to put/get a quantity in the container.

Stores

1- Stores APIs	
C: StorePut<base.Put>(store, item)	Ask to put an 'i' object in store 's'.
C: StoreGet<base.Get>(resource, filter=lambda item:True)	Ask to remove an object from the "store".
C: FilterStoreGet<StoreGet>	Ask to remove an object corresponding to the filter in the "store".
C: Store<base.BaseResource>(env, capacity=float('inf'))	Resource of a given capacity to store objects.
C: PriorityItem(namedtuple('PriorityItem', 'priority item'))	Associate a priority to an object.
C: PriorityStore<Store>	Resource of a given capacity to store objects in order of priority.
C: FilterStore<Store>	Resource of a given capacity to store objects admitting requests with filter.
a: put/get = BoundClass<StorePut/Get>	Ask to put/remove an object in/from the "store".
a: get = BoundClass<FilterStoreGet>	Ask to remove an object from the "store" corresponding to the given filter.

7.2.3.4 Exceptions

The module *simpy.exceptions* contains the base class *SimPyException* for all specific exceptions. There is *Interrupt (cause)* which returns the exception thrown to the process if it is interrupted for the given cause and *StopProcess (value)* thrown to stop a Simpy process.

7.3 Simulation of stochastic processes

7.3.1 Simulation of Poisson process

The Poisson process $N(t)$ is considered as an arrival sequence whose inter-arrival time (IAT) is an exponential r.v of parameter λ (we know how to generate it). So to generate the number of arrivals up to time T, we proceed with the generation of IATs until their sum exceeds T. The number of generated IATs represents the value generated for $N(T)$.

Let's code!

```
# Code705.py

from scipy.stats import expon
import matplotlib.pyplot as plt

def generatePoissonSP(lamda=1, T=1):
    t, N, tim = expon.rvs(scale=1/lamda), 0, []
    while (t<T):
        tim.append(t)
        N, t = N+1, t+expon.rvs(scale=1/lamda)
    return N,tim

def plotPP(lamda):
    n,tim = generatePoissonSP(lamda,20)
    return plt.step(tim,range(n))

lamdas = [0.25,0.5,1,2]
plt.legend([plotPP(l)[0] for l in lamdas],["lambda="+str(l) for l in lamdas])
plt.xlabel('Time'); plt.ylabel('arrivals number')
plt.show()
```

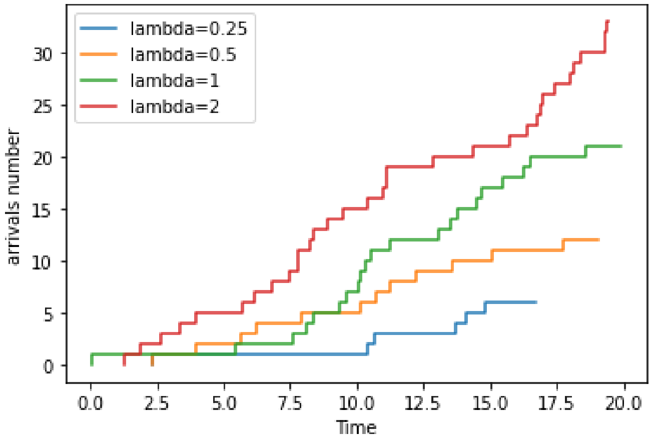


Figure 7.3: Simulation of Poisson process (code705.py)

7.3.2 Simulation of discrete time Markov chain

As shown in the following code, the simulation of a DMTD consists in, repeatedly, generating the following state out of the current state using the inverse transformation method having its distribution as a parameter.

Let's code!

```
# Code 706.py

from itertools import accumulate
from random import random
import sys;sys.path.append('../chap4')
from CMTD import CMTD

def simulateCM(P,S,steps,cs):
    CM = CMTD(P,S)
    ls = []
    indices = range(len(CM.S))
    for i in range(steps):
        ls.append(CM.S[cs])
        prob = list(accumulate(P[cs]))
        cs=transformation_inverse(prob,indices)
    return (CM, "".join(ls))

def transformation_inverse(prob,indices):
    u = random()
    for i in range(len(prob)):
        if(u<prob[i]): return indices[i]
    return indices[-1]

def compareToSteadyState(P,S,n,s0):
    res = simulateCM(P,S,n,s0)
    ana_p = res[0].steady_prob();            print("Analytical prob : ", ana_p)
    emp_p = [res[1].count(s)/n for s in S]; print("Empirical prob  : ", emp_p)

# Test
n = 10000
print('Ergodic')
P = [[1/3, 1/3, 1/3],
     [3/5, 0.0, 2/5],
     [3/4, 1/8, 1/8]]
S = ['R','N','S']

compareToSteadyState(P,S,n,1)
n = 50
print("CMTD1: ",simulateCM(P,S,n,1)[1])

print('Not ergodic')
P = [[1.0, 0.0, 0.0],
     [3/5, 0.0, 2/5],
     [3/4, 1/8, 1/8]]
print("CMTD2: ",simulateCM(P,S,n,1)[1])

#______________________________   Output   ______________________________________
```

```
# Ergodic
# Analytical prob :  [0.50769231 0.20512821 0.28717949]
# Empirical prob  :  [0.5065, 0.2072, 0.2863]
# CMTD1:  NRNRSRSRNSRRRNRSRNRNRSRRSNRRNRSRRSRNSNRSSRRNRSRNRR
# Not ergodic
# CMTD2:  NRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
```

Given the non-deterministic nature of the code, it is obvious that the obtained result can change from one execution to another.

7.3.3 Simulation of continuous time Markov chain

In this section, we will see how to simulate a CTMC. As input, we need the transition matrix P of the embedded MC and the value λ_i of each state of the CTMC. The simulation will run for a time T_{max}. The proposed Python code is as follows:

Let’s code!

```
# Code 707.py

from itertools import accumulate
from random import random
import math
import sys;sys.path.append(’../chap4’)
from CMTC import CMTC

expo = lambda lamda : (-1/lamda)*math.log(random())

def transformation_inverse(prob,indices):
    U = random()
    for i in range(len(prob)):
        if(U < prob[i]): return indices[i]
    return indices[-1]

def simulateCM(P,lamdas,S,T_max,cs):
    CM = CMTC(P,lamdas,S)
    indices = range(len(CM.S))
    ls, tc = [], 0
    while tc < T_max:
        tps = expo(lamdas[cs])
        ls.append((CM.S[cs], round(tps,2)))
        tc = tc + tps
        prob = list(accumulate(P[cs]))
        cs = transformation_inverse(prob,indices)
    return (CM , ls)

def compareToSteadyState(res,T):
    ana_p = [ round(p,2) for p in res[0].steady_prob()]
    print("Analytical prob : ", ana_p)
    emp_p = [ round(sum(map(lambda r: r[1], filter(lambda r: r[0]==s, res[1]))))/T,2) for s in S]
    print("Empirical prob  : ", emp_p)

S = [’R’,’N’,’S’]
P = [[0.0, 2/3, 1/3],
     [3/5, 0.0, 2/5],
     [3/4, 1/4, 0.0]]
lamdas = [4, 5, 6]
T_max = 500

res = simulateCM(P,lamdas, S,T_max,1)
compareToSteadyState(res, T_max)
print("CMTC : ",simulateCM(P,lamdas,S,5,1)[1])

#______________________________   Output  ______________________________________
# Analytical prob :  [0.47, 0.32, 0.21]
# Empirical prob  :  [0.46, 0.33, 0.22]
# CMTC : [(’N’, 0.05), (’R’, 0.0), (’S’, 0.03),..., (’S’, 0.09), (’R’, 0.19)]
```

7.3.4 Simulation of queuing systems

7.3.4.1 Simulation of station M/M/1

The simulation of a M/ M/1 station takes as input the average arrival rate λ (tau1), the average service rate μ (tau2) and the maximum simulation time (maxT); after this time, only departures of customers already received in the station are processed and no arrivals are possible. We use a *clock* variable to follow the evolution of time. We also need two variables: *t1* that represents the time of the next arrival in the system and *t2* the time of the next departure. Variable n calculates the number of customers in the station, it is incremented at each arrival event and decremented at each departure. The other variables are used for the calculation of simulation statistics, namely the average response time of customers in the station and the average response time.

Let’s code!

```
# Code708.py

import random
import math
import matplotlib.pyplot as plt
import sys;sys.path.append('../chap5')
from Code502 import getMM1

expo = lambda lmbd : (-1/lmbd)*math.log(random.random())

#
def simul(lamda,mu,maxT):
    clock, n, nb = 0, 0, {}             # n is customers number in the system
    t_arv, t_dep = expo(lamda), maxT    # next arrival/departure/max time
    arv, dep = [], []                   # Arrival/Departure times
    n_arv, n_dep = 0, 0                 # Arrivals/Departures number

    #
    def departure():
        nonlocal n_dep, dep, n, t_dep, mu, maxT, clock

        n_dep += 1;  dep.append(t_dep)
        if(n >= 0): nb[n] = (nb[n] if n in nb else 0) + t_dep - clock
        n, clock = n-1, t_dep
        t_dep = clock + expo(mu) if n>0 else maxT

    stats = []
    #
    while(clock < maxT):
        stats.append((clock,n))           # add event id and time
        if(t_arv < t_dep):                # arrival event

            n_arv += 1; arv.append(t_arv)
            nb[n] = (nb[n] if n in nb else 0) + t_arv - clock
            n, clock = n+1, t_arv
            t_arv = clock + expo(lamda)

            if(n == 1): t_dep = clock + expo(mu)
        else:                             # departure event
            departure()

    #
    while(n_arv > n_dep):
        stats.append((clock,n))
        departure()

    prob = {s:nb[s]/clock for s in nb}
    avg_nb_clients = sum(s*prob[s] for s in prob)
    wait = list(map(lambda x, y: x - y, dep, arv))
    avg_time = sum(wait)/len(wait)
    return(avg_nb_clients,avg_time, stats)

#
print('Analytical performances : ');   getMM1(2,3)
print('Empirical performances : ');     nb, tps, stats =simul(2,3,10)
print("Average number of clients in the station: "+str(round(nb,2)))
print("Average response time: " + str(round(tps,2)))

# Plot
plt.step([t[0] for t in stats], [t[1] for t in stats])
plt.xlabel('Time'); plt.ylabel('clients number')
plt.show()
#______________________________   Output   ______________________________________
# Performances ananlytiques :
# =================MM1 =======================
# p0* :0.33333
# p1* :0.22222
# p3* :0.09877
# L   :2.00
# Lq  :1.33
# W   :1.00
# Wq  :0.67
# Performances empiriques :
# Nombre moyen de clients dans la station:1.97
# Temps de reponse moyen:0.98
```

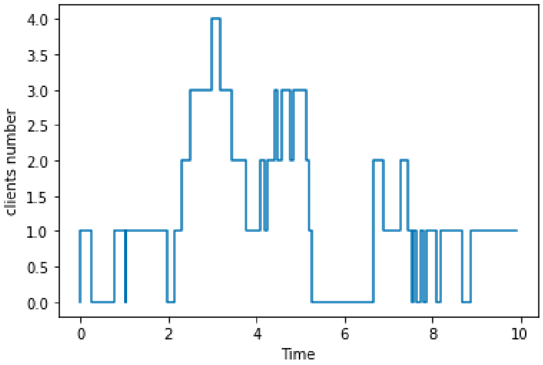


Figure 7.4: Simulation of M/M/1 (code708.py)

7.3.4.2 Simulation of queuing stations with simpy

To simulate queuing systems, servers are modeled as shared resources and clients as processes. Another process responsible for generating customer processes should be added. The simulation is started by the *run()* instruction. When a client process is created, it first requests a resource (a server), if no server is available, the process is suspended (put into the queue) until a server is released. At this point, the client process leaves the queue and hangs for the time of the service, then it ends its execution, which makes the resource available again. By default the simulation ends when no event is pending, but other stopping conditions can be used as the maximum simulation time for instance.

Let's code!

The following module is a utility that groups together the common functions and variables used by the codes in the following sections. Global variables are initialized by Simpy classes such as simpy.Environment and simpy.Resource, ... Among the functions we find the lambda expressions that define the laws of the random generators and those which collect and plot the statistics of the simulation as well as the Gant diagram used in the application of this chapter.

```
# Code709.py

import random
import simpy
from simpy.resources import resource

# Shared
e =  simpy.Environment()
P = e.process
R = e.run

# Resources
RSC = simpy.Resource
RSC_withCapacity = lambda cap: simpy.Resource(e, capacity=cap)
RSC_Prempt =  lambda cap: resource.PreemptiveResource( e , capacity = cap )
RSC_Prio =  lambda cap: resource.PriorityResource( e , capacity = cap )
RQT_prio = resource.PriorityRequest

TO = e.timeout
SETO =  simpy.events.Timeout

# load datastructures
ld  = { 'Ls':[],     # Load historique in the system
        'L':[0],     # current load
        'Q':[0],     # current queue
        'tl':[0],    # time when updating the load
        'tq':[0],    # time when updating the queue
        'Nfs':[1]    # Number of servers
        }
```

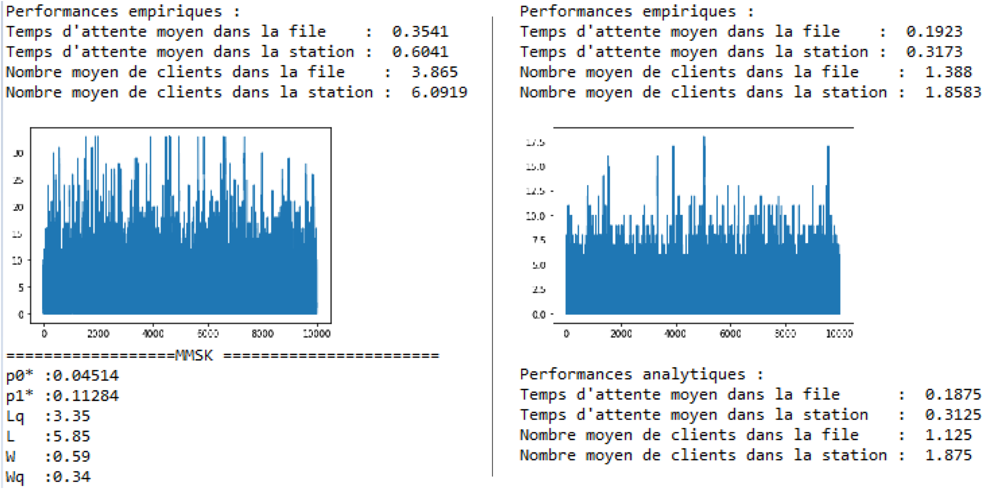
```
# updateLoad L/Q
def updateLoad(E, te, h,k, stationId=0):
    global ld
    ld[E][stationId] = ld[E][stationId] + h;     # number of client in Queue Q or System L
    ld[’Ls’].append((round(e.now - ld[te][stationId],4), ld[E], k)); ld[te][stationId] =e.now

#
rexpo = lambda mu : random.expovariate(mu)
rinverse = lambda probas : random.choices(range(len(probas)), probas)[0]
runif = lambda a,b : random.randint(a, b)
randColor = lambda:(random.random(),random.random(),random.random())

# iterator for the loop
def geti():
    i=0;
    while True: i=i+1; yield i
g = geti()

# on Event Arrival +  Sertvice  +  Departure
def onEvent(event, id, stats, stationId=0):
    typEvent = event[0]
    # update system load
    if typEvent  != ’S’: updateLoad(’L’ ,’tl’ ,(1 if typEvent=="A" else -1) ,0, stationId)

    # update system queue
    if ((typEvent  == ’A’ and ld[’Nfs’][stationId] ==  0) or
        (typEvent  == ’S’ and ld[’Q’][stationId] > 0)) :
        updateLoad(’Q’ ,’tq’ ,(1 if typEvent=="A" else -1) ,1, stationId)

    # update number of available servers
    if typEvent  != ’A’:
        ld[’Nfs’][stationId]= ld[’Nfs’][stationId] + (1 if typEvent=="D" else -1)

    #statistic array [’E’, time, ’ID’, servers , L ,Q ,Station]
    stats.append(
        [ typEvent,                       # Event type  Arrival, Service and Departure
          round(e.now,4),                 # time
          id,                             # client id ’ClientID’
          ld[’Nfs’][stationId],           # number of available servers
          ld[’L’][stationId],             # station load
          ld[’Q’][stationId],             # station queue
          stationId                       # station id
          ]
        )

# ===========================================================================
# wait_queue   +  wait_system  +
def process_output(psim):
    wait_queue, wait_system, N = 0, 0, len(psim)
    for evt in psim:
        wait_queue   += evt[2] - evt[1]
        wait_system  += evt[3] - evt[1]
    wait_queue   = round(wait_queue /N,4)
    wait_system  = round(wait_system/N,4)
    return (wait_queue, wait_system)

#
def plot_state(stats0):
    import matplotlib.pyplot as plt
    nbstations =  len(stats0)
    for i in range(nbstations):
        plt.step([v[0] for v in stats0[i].values()], [v[3] for v in stats0[i].values()])
    plt.xlabel(’Time’);  plt.ylabel(’clients number’)
    plt.show()

# Statistics
def collect_stats(psim, pld, show, nbstations=1):
    stats0 =[{} for _ in range(nbstations)]
    stats1 =[{} for _ in range(nbstations)]
    eprev, L, Lq = 0, 0, 0

    for currentStation in range(len(stats0)):
        stats = stats0[currentStation];  statsW = stats1[currentStation]
        psim0 = [e for e in psim[’staE’] if e[6] == currentStation]  # filter stats for the
     current station

        # collect stats [time, [(’E’, ’ID’),...], Server, Q, L, Station]
        for evt in psim0:
            # stats for the system based on time as key
            if not evt[1] in stats.keys():  # evt[1] time
                stats[evt[1]] = [evt[1], [],0, 0, 0,0]
                stats[evt[1]][2] = evt[3]   # Available servers
                stats[evt[1]][3] = evt[4]   # L
                stats[evt[1]][4] = evt[5]   # Q
            else:
                stats[evt[1]][2] = min(stats[evt[1]][2], evt[3])
                stats[evt[1]][3] = min(stats[evt[1]][3], evt[4])
                stats[evt[1]][4] = min(stats[evt[1]][4], evt[5])
            stats[evt[1]][1].append((evt[0], evt[2]))

            # stats for the system queue : W based on client_ID [ID, A, S, D]
            if evt[2] != ’None’:   # evt[2] Client ID
                if not evt[2] in statsW.keys(): statsW[evt[2]] = [evt[2], 0, 0, 0]
                if evt[0]==’A’: statsW[evt[2]][1] = evt[1]
                if evt[0]==’S’: statsW[evt[2]][2] = evt[1]
                if evt[0]==’D’: statsW[evt[2]][3] = evt[1]

            L += (evt[1] - eprev)*evt[4]
            Lq += (evt[1] - eprev)*evt[5]
            eprev = evt[1]

        wqs = process_output(statsW.values())
        lds = (round(L/psim[’T_sim’],4), round(Lq/psim[’T_sim’],4))
        return wqs, lds, stats0

# show_statistics
def show_statistics(psim, pld, show, nbstations=1):
    wqs, lds, sta = collect_stats(psim, pld, show, nbstations)#
    if show :
        for e in sta:
            for v in e.values() : print(v)

    print("Empirical performances :")
    print("Average time in the queue               : ", wqs[0])
    print("Average time in the system              : ", wqs[1])
    print("Average number of clients in the queue  : ", lds[1])
    print("Average number of clients in the system : ", lds[0])
    plot_state(sta)

# ===========================================================================
# method that plots the gant chart of the application
def gant(cpus, tasks,l, H=50, M=5 ):
    import matplotlib.pyplot as plt
    fig, axi = plt.subplots()
    axi.set_ylim(0, H);      axi.set_xlim(0, l)
    axi.set_xlabel(’Time’);  axi.set_ylabel(’Processor’)
    axi.grid(True)

    # Setting ticks on y-axis
    dx =  int((H - 2*M)/ len(cpus))
    axi.set_yticks([dx + i*dx  for i in range(len(cpus))])
    axi.set_yticklabels(cpus)

    # Declaring a bar in schedule
    for k,task in tasks.items():
        posH=(task[’cpu’]+1)*dx-M/2
        axi.broken_barh(task[’length’], (posH, M) , facecolors = task[’color’])
        for t in task[’length’]:
            axi.annotate(k, (1,1), xytext=(t[0]+t[1]/2-1, posH+M+1))
    plt.show()
```

7.3.4.3 Example of a car wash station

The following code allows us to simulate several configurations (M/M/1, M/M/S/K, ...) of a car wash system which has a limited number of automatic car wash machines and which receives cars to be washed.
Washing machines are modeled as resources, and washing as a process. When a car arrives, it asks for a washing machine, when it gets it, it starts the washing process and waits for it to finish, then it releases the machine and quits the system. If no machine is available, it remains on standby. Cars and washing processes are generated by a *setup* process.

Let’s code!

```
# Code710.py

from Code709 import e,P,R,TO, RSC, rexpo, g, show_statistics, ld, onEvent
import sys;sys.path.append(’../chap5’)

# =========================================================================
# station entity model
class Station(object):
    def __init__(self, psim):
        self.machines = RSC(e, psim[’S’]);
        self.mu = psim[’mu’]; self.sd = psim[’SD’]

    def wash(self, voit):
        yield TO(self.sd(self.mu))

# car entity model
def Car(name, station, psim):
    # event Arrival, Service, Departure
    onCarEvent = lambda E: onEvent(E, name, psim[’staE’])

    # core
    onCarEvent(’Arrival’)                                    # onArrival event
    with station.machines.request() as machine:
        yield machine;               onCarEvent(’Service’)   # onService event
        yield P(station.wash(name)); onCarEvent(’Departure’) # onDeparture event

# simulation setup
def setup(psim):
    station = Station(psim)
    ld[’Nfs’]= [psim[’S’]]
    psim[’staE’].append([’I’, 0.0, ’None’, psim[’S’] ,0 ,0 ,0])      # initilize statistics
    while True:
        yield TO(psim[’AD’](psim[’lambda’]))
        if psim[’K’]==-1 or len(station.machines.queue) < psim[’K’]: # Queue Capcity limit
            if(e.now > psim[’T_sim’]): break                         # Simulation time limit
            P(Car(’C%d’ % next(g), station, psim))                   # new arrival process

# =============================================================================

def run_sim(sim, show=False):
    P(setup(sim)); R()                                    # Simulation
    show_statistics(sim, ld, show)                        # Statistic
#
def sim_mmsk(show=False):
    from Code505 import getMMSK

    # simulation examples
    sim = {
    ’T’     : ’MMSK’,      # type of the model
    ’mu’    : 4,           # service rate
    ’lambda’: 10,          # arrival rate
    ’S’     : 3,           # number of servers
    ’K’     : 30,          # Queue capacity
    ’SD’    : rexpo ,      # service distribution
    ’AD’    : rexpo,       # arrival distribution
    ’T_sim’ : 1000,        # Simulation time
    ’staE’  : [] }         # Statistics array

    run_sim(sim, show)                                      # simulation
    getMMSK(sim[’mu’],sim[’lambda’],sim[’S’], sim[’K’])     # Theory

sim_mmsk()


# def sim_md1(show=False):
#     simulation de la station M/D/1
#     sim = {
#     ’T’:’MDS’,              # type of the model
#     ’mu’:8,                 # service rate
#     ’lambda’:6,             # arrival rate
#     ’S’:1,                  # number of servers
#     ’K’:-1,                 # Queue capacity
#     ’SD’: lambda x:1/x ,    # service distribution
#     ’AD’:rexpo,             # arrival distribution
#     ’T_sim’:10000,          # Simulation time
#     ’staE’:[] }             # Statistics array

#     run_sim(sim, show)                                          # simulation
#     print("Performances analytiques :")
#     D , rho = 1/sim[’mu’],  sim[’lambda’]/sim[’mu’]
#     Wq      = rho/((2/D)*(1-rho));
#     W       = D + Wq ;
#     Lq      = 1/2*rho**2/(1-rho);
#     L       = rho + Lq;
#     print("Temps d’attente moyen dans la file      : ", Wq)
#     print("Temps d’attente moyen dans la station   : ", W)
#     print("Nombre moyen de clients dans la file    : ", Lq)
#     print("Nombre moyen de clients dans la station : ", L)

# sim_md1()
```



```
Performances empiriques :
Temps d'attente moyen dans la file    :  0.3541
Temps d'attente moyen dans la station :  0.6041
Nombre moyen de clients dans la file     :  3.865
Nombre moyen de clients dans la station :  6.0919
```

```
==================MMSK =======================
p0* :0.04514
p1* :0.11284
Lq  :3.35
L   :5.85
W   :0.59
Wq  :0.34
```

```
Performances empiriques :
Temps d'attente moyen dans la file    :  0.1923
Temps d'attente moyen dans la station :  0.3173
Nombre moyen de clients dans la file     :  1.388
Nombre moyen de clients dans la station :  1.8583
```

```
Performances analytiques :
Temps d'attente moyen dans la file       :  0.1875
Temps d'attente moyen dans la station    :  0.3125
Nombre moyen de clients dans la file     :  1.125
Nombre moyen de clients dans la station :  1.875
```

Figure 7.5: Empirical vs analytical results (code710.py)

7.3.4.4 Simulation of queuing networks with simpy

In this section we simulate queuing networks (see appendix B.2). The network is described by the *topo* structure which contains a list of tuples, each one describes a station connected to the network as follows:
1. The number of servers (considered homogeneous) in the station and their speed (number of clients per unit of time).
2. The list of successor stations (where outgoing clients can go).
3. For each successor station, the probability that an outgoing customer will move on there. If outgoing customers can exit the system, an additional exit probability is added. If no successor exists, the value *None* is used.
The *setup()* method creates the network from the *topo* structure, then proceeds to the continuous generation (up to a maximum time T) of the incoming clients at the initial stations. Each created client is directed to his station to acquire a server (or join the queue if the server is not available), when he ends his service, he goes to the next station or leaves the system depending on the topology of the network.

Let’s code!

```
#Code711.py

import heapq  # heapqueue allows to allocate servers in round robin policy
from Code709 import e,P,R,TO, rexpo, g, rinverse, RSC_withCapacity, SETO

# ===============================================================================================
# show
def show(cz, args):
    if cz == 2 : print(’%7.2f \t %s \t %s \t %s \t %s’ % tuple([e.now] + args))
    else       : print(’%7.2f \t %s \t %s \t\t %s’ % tuple([e.now] + args))

# main method
def carwash(T, topology):
    class Station(object):
        def __init__(self, id, infos):
            self.id=id;
            self.S= infos[’S’]
            self.M = infos[’Mu’]
            self.nextStations= infos[’nextS’]
            self.probas=infos[’probas’]
            self.Indices = list(range(self.S));
            self.Servers = RSC_withCapacity(self.S);

        def pf_svc(self, c): X =rexpo(self.M);   yield TO(X)

        # onService
        def onService(self, id):
            if len(self.Indices) > 0:
                svr = heapq.heappop(self.Indices);    # pop in queue
                show(2, [id, ’S’, svr, self.id])
            return svr

        # onDeparture
        def onDeparture(self,id, svr):
            heapq.heappush(self.Indices, svr);        # push from queue
            show(1, [id, ’D’,self.id])

        # pf_getin : when access a station
        def pf_getin(self, id):
            show(1, [id, ’A’, self.id])                              # Arrival
            with self.Servers.request() as s:
                yield s;                   svr = self.onService(id)  # Service
                yield P(self.pf_svc(id));  self.onDeparture(id, svr) # Departure
                if self.nextStations != None:
                    i =  rinverse(self.probas)
                    if(i<len(self.nextStations)):
                        P(self.nextStations[i].pf_getin(id)) # go to next station

    # create_system : create the topology
    def create_system(topo):
        stations = [ Station(i,            #  the station’s ID
                    { ’S’: topo[i][0][0],  # number of servers
                      ’Mu’: topo[i][0][1], # service rate
                      ’nextS’:None ,       # list of next stations
                      ’probas’: topo[i][2] # probabilities to go to next stations
                     })
                     for i in range(len(topo)) if i!=0]
        # create next stations
        for station in stations:
            if(topo[station.id][1]!=None):
                station.nextStations = [stations[v-1] for v in topo[station.id][1]]

        # returns initial stations with their arrival rates
        return [stations[i-1] for i in topo[0][1]] , topo[0][2]

    # setup the simulation
    def setup():
        def firstS(lamdas):
            t=min(times)
            first=times.index(t)
            for k in range(len(times)): times[k]-=t
            times[first]=rexpo(lamdas[first])
            return first,t

        initStations, lamdas = create_system(topo)

        times=[0 for i in range(len(initStations))]
        for i in range(len(times)): times[i]= rexpo(lamdas[i])
        while e.now<T:
            indf, t=firstS(lamdas)
            car = yield SETO(e, delay=t, value='C%d' % next(g))
            P(initStations[indf].pf_getin(car))

    P(setup());  R()

# Test
topo = [
  #(Svr_Nb,mu)  [successors], [Probavility]
  ((None,None), [1,3], [6,2]),
  ((2   ,4   ), [2,3], [0.6,0.2,0.2]),
  ((1   ,3   ), [3]  , [0.5,0.5]),
  ((1   ,6   ), None , None)]

carwash(5,topo)
```

7.3.4.5 Simulating queues with priorities

In this section, we simulate a queue station whose incoming clients have priorities. Priorities may or may not be preemptive. We use *PreemptiveResource* and *PriorityResource* from *Simpy*. We consider an exponential service time (memoryless), so in the case of preemptive priorities, when a client is interrupted by another one with a higher priority, it is inserted into the server queue, and when he acquires the server again, it resumes its execution from the beginning (memoryless). The case of a service without memoryless property is dealt with in the application of this chapter.

Let's code!

```python
# Code712.py

from simpy import Interrupt
from Code709 import e,P,R,TO, rexpo, RSC_Prempt, RQT_prio, RSC_Prio

#
def station(nbClients,serveurs,lamdas,mu,stat, preemption=True):
    times = [ rexpo(l) for l in lamdas ]          # arrival times initialization
    for id in range(nbClients):
        t = min(times)                            # selects the earlier arrival
        priority = times.index(t)                 # gets its priority
        times = [ tc - t for tc in times ]        # updates the arrival times
        times[priority] = rexpo(lamdas[priority]) # generate the next arrival of this priority
        yield TO(t)
        stat[priority][id]=[0,0,0]
        P(client(id,priority,serveurs,mu,stat, e.now, preemption))

#
def client(id,priority,serveurs,mu,stat, beginTime, preemption=True):
    try:
        stat[priority][id][0] = beginTime
        with RQT_prio(serveurs,priority, preempt=preemption) as req:
            req.time=beginTime
            yield req
            stat[priority][id][1] = e.now
            yield TO(rexpo(mu))
            stat[priority][id][2] = e.now
    except Interrupt:   # in case of preemptive priority
        P(client(id,priority,serveurs,mu,stat, beginTime, preemption))

#
def statistiques(stat):
    rgs = range(len(stat))
    wait_queue, wait_system = [0 for _ in rgs], [0 for _ in rgs]
    for priority in rgs:
        for k in stat[priority].keys():
            wait_queue[priority] += stat[priority][k][1] - stat[priority][k][0]
            wait_system[priority]  += stat[priority][k][2]- stat[priority][k][0]
    print("Wq : Average waiting time in the queue")
    print("W  : Average waiting time in the station")

    print("priority \t  Wq \t\t  W ")
    for p in rgs:
        twq = round(wait_queue[p]/len(stat[p]),2)
        tw = round(wait_system[p]/len(stat[p]),2)
        print(p," \t\t\t ", twq, " \t ", tw)

#
nbClients, nbServeurs, mu, lamdas, preemption = 100, 1 ,10, [3,4,2], False
stat = [{} for p in range(len(lamdas))]

serveurs = (RSC_Prempt if preemption else RSC_Prio)(nbServeurs)
P(station(nbClients, serveurs, lamdas, mu, stat, preemption)); R()

#
print("Empirical performances:")
statistiques(stat)
```

7.4 Application: Scheduling

A scheduling problem consists of planning the execution of a set of tasks by allocating the required resources to them and setting their start dates. In the context of multiprocessor systems, tasks are the processes to run on parallel processors that represent the resources in that system. In the following application we consider a multiprocessor system having n processors with different characteristics (speed) and queues, and a sequence of tasks to be executed on this system. The scheduler is the component that assigns the submitted tasks to processors according to a given policy.
In what follows we consider that the tasks are independent of each other and they have different sizes (number of elementary instructions) and priorities. The adopted scheduling policy considers preemptive priorities (a higher priority task interrupts the execution of a lower priority task). Execution time is calculated based on the size of the task and the speed of the allocated processor. The Gant chart is the tool used to visualize the result of the scheduling and in particular the occupation of the processors.

Let’s code!

```
# method that plot the gant chart of the application
def gant(cpus, tasks,l, H=50, M=5 ):
    import matplotlib.pyplot as plt
    fig, axi = plt.subplots()
    axi.set_ylim(0, H);      axi.set_xlim(0, l)
    axi.set_xlabel(’Time’);  axi.set_ylabel(’Processor’)
    axi.grid(True)

    # Setting ticks on y-axis
    dx =  int((H - 2*M)/ len(cpus))
    axi.set_yticks([dx + i*dx  for i in range(len(cpus))])
    axi.set_yticklabels(cpus)

    # Declaring a bar in schedule
    for k,task in tasks.items():
        posH=(task[’cpu’]+1)*dx-M/2
        axi.broken_barh(task[’length’], (posH, M) , facecolors = task[’color’])
        for t in task[’length’]:
            axi.annotate(k, (1,1), xytext=(t[0]+t[1]/2-1, posH+M+1))
    plt.show()
```

The following code uses Simpy to implement the system in question. The *schedule* method fulfills the role of the scheduler by generating the tasks according to a transition rate *lamda* and by allocating the processors according to a policy given as a parameter. This policy is defined as a method which implements the strategy to be studied. In our case we have opted for the strategy which selects the processor with the least loaded queue *lessLoaded*. We implemented the processors as Simpy resources with preemptive priorities.

Let’s code!

```
#Code713.py

import simpy
from Code709 import (e,P,R,TO, rexpo, runif, g, RSC_Prempt, SETO,
                     RQT_prio, gant, randColor)

remain = lambda X,start,speed: X - (e.now - start)*speed # remaining time in the server
# show
def show(evt, s, task, X):
    print(round(e.now,3), " \t " + evt + " \t ",task[0], " \t ", X ,
          " \t ", s.id, " \t ", task[2], " \t ", end=’’)

class Station(object):
        def __init__(self, id, infos):
            self.id = id;
            self.Speed = infos[’Speed’]
            self.Servers = RSC_Prempt(1);

        def update(self, task, X, dicTask, evnt, start):
            X = int(remain(X,start,self.Speed))
            dicTask[task[0]][’length’].append((round(start,3),round(e.now-start,3)))
            show(evnt, self, task, X)
            return X

        def pf_svc(self, X): delay = X/self.Speed ;   yield TO(delay)

        def pf_run(self, task, stations, X, dicTask, preemption=True):
            if X>0:
                try:
                    with RQT_prio(self.Servers,task[2], preempt=preemption) as s:
                        print([len(s.Servers.queue) for s in stations])
                        yield s;
                        start = e.now
                        show(’S’, self, task, X) ; print()
                        yield P(self.pf_svc(X))
                        X = self.update(task, X, dicTask, ’D’, start)
                    print([len(s.Servers.queue) for s in stations])
                except simpy.Interrupt:
                    if remain(X,start,self.Speed)>0:
                        X = self.update(task, X, dicTask, ’I’, start)
                        P(self.pf_run(task,stations, X, dicTask, preemption))

# create the system: one station for each processor
def create_system(proc):
        stations = [ Station(i,{’Speed’: proc[i]}) for i in range(len(proc))]
        return stations

#
def schedule(T,lamda, procs, pmin, pmax, policy, dicTask):
    print(’Time \t Event \t Task \t Remain \t Server \t Priority \t Queues’)
    stations = create_system(procs)
    while e.now<T:
        t = rexpo(lamda) # next arrival
        # creates a new task (Id, length, priority)
        task = yield SETO(e, delay=t, value=(’T%d’ % next(g), runif(pmin,pmax)*10, runif(1,3)))
        proc = policy(stations) # calls the specified policy to selects a processor
        # intialize execution structure of the task
        dicTask[task[0]] = {’length’:[], ’cpu’:proc, ’color’:randColor()}
        show(’A’, stations[proc] , task, task[1])
        # creates the station process associated to the current task
        P(stations[proc].pf_run(task, stations, task[1],  dicTask ))

# lessLoaded policy
def lessLoaded(stations):
    lp = list( map(lambda s: s.Servers.count, stations))
    if(min(lp) == 0): return lp.index(min(lp))
    lq = list( map(lambda s: len(s.Servers.queue), stations))
    return lq.index(min(lq))

# processors speeds, tasks dictionary
processors, dicTask = [3,2,4], {}
# pmin, pmax: minimum and maximum length of tasks, T: simulation time
lamda, pmin, pmax, T = 5, 1, 9, 2
P(schedule(T,lamda,processors, pmin, pmax, lessLoaded, dicTask));  R()

for k,t in dicTask.items():
    print(k," cpu:",t[’cpu’],’  length:’,t[’length’])
lastD = max([d[0]+d[1] for t in dicTask.values() for d in t[’length’]])
gant(range(len(processors)),dicTask, int(lastD)+10)

#______________________________   Output  ______________________________________
# T1  cpu: 0   length: [(0.232, 1.872), (25.438, 21.333)]
# T2  cpu: 1   length: [(0.33, 0.647), (50.976, 24.0)]
# T3  cpu: 2   length: [(0.655, 22.5)]
# T4  cpu: 0   length: [(5.438, 20.0)]
# T5  cpu: 1   length: [(0.976, 20.0)]
# T6  cpu: 2   length: [(23.155, 12.5)]
# T7  cpu: 0   length: [(46.771, 6.667)]
# T8  cpu: 1   length: [(20.976, 30.0)]
# T9  cpu: 2   length: [(35.655, 15.0)]
# T10  cpu: 0   length: [(2.104, 3.333)]
```

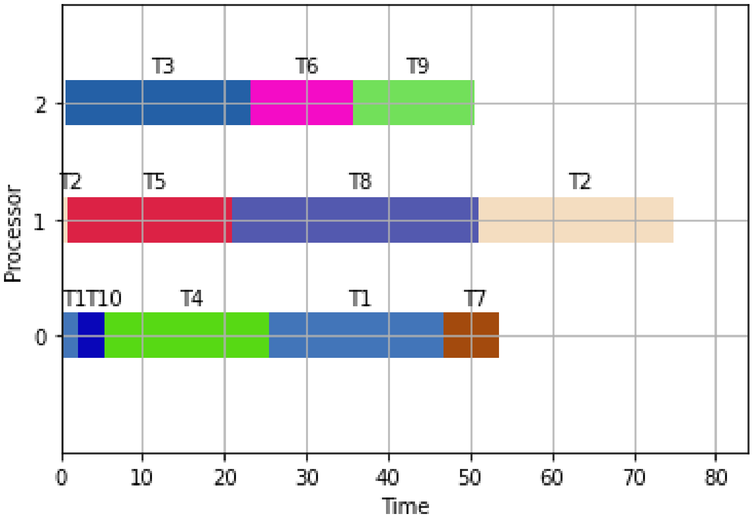


Figure 7.6: Gant diagram: processors occupency (code713.py)

7.5 Exercises

Exercise 1. *Write the code that simulates the birthday problem described in Chapter 1.*

Exercise 2. *Write the code that simulates exercise 23 of chapter 1 considering the number of k errors and n testers.*

Exercise 3. *Simulate example 25 from chapter 2.*

Exercise 4. *Empirically verify (using simulation) the Markov and Chebychev inequalities seen in chapter 2.*

Exercise 5.
1. Simulate exercise 9 in chapter 2.
2. Find an approximation of the number π using the result of the simulation.

Exercise 6. *Simulate the random walk (chapter 3) in a plane (2D) and in a 3D space.*

Exercise 7. *Simulate exercise 5 of chapter 3.*

Exercise 8. *Simulate example 2 from chapter 4.*

Exercise 9. *Simulate example 4 from chapter 4.*

Exercise 10. *Consider Polya's model which consists of a box containing w white balls and b black balls. At each stage, a ball is drawn at random and it is returned to the box, adding a ball of the same color. Simulate this model.*

Exercise 11. *Write the code that simulates and compares the two systems described in exercise 5 of chapter 5.*

Exercise 12. *Write the code that simulates the system described in exercise 7 of chapter 5 and compare the simulation results with the analytical results.*

Exercise 13. *Implement the code which calculates the analytical results of Jackson's networks in the appendix and compare them with those obtained by code711.*

Exercise 14. *In the application code:*
1. Implement the policy of assigning each priority to a different processor (e.g. processor1 to priority 1 tasks, processor2 to priority 2 tasks, and processor3 to priority 3 tasks).
2. Compare the total execution time (the time to complete the last executed task) and the load of the processors with the lessLoaded () policy.
3. Modify code 713 so that we increase the priority (in fact we lower its value to make it of greatest priority) of the task each time it is interrupted.
4. Compare the average wait time for the tasks of each priority with the previous strategies.

Appendices

Appendix A

Counting

The resolution of many probability problems require counting techniques to count the number of possible outcomes in a given random experiment.

A.1 Multiplication principle

Consider k random experiments $\{E_i\}_{1..k}$ such that S_i is the set of possible outcomes of the i^{th} experiment (E_i) and $|S_i| = n_i$. The random experiment E which consists in performing this sequence $\{E_i\}_{1..k}$ of experiments has

$$S = \prod_{i=1}^{k} S_i$$

whose number of possible outcomes is equal to

$$|S_E| = n = \prod_{i=1}^{k} n_i$$

Example 1. *We roll a coin and a dice, what is the number of possible outcomes of this experiment E?*
E is an experiment composed of the sequence of random experiments E_1: tossing a coin and E_2: rolling a dice.
- E_1 has the possible outcomes $S_1 = \{T, H\}$, $|S_1| = n_1 = 2$,
- E_2 has the possible outcomes $S_2 = \{1, 2, 3, 4, 5, 6\}$ $|S_2| = n_2 = 6$,
So,

$S_E = S_1 \times S_2 = \{(T, 1), (T, 2), (T, 3), (T, 4), (T, 5), (T, 6), (H, 1), (H, 2), (H, 3), (H, 4), (H, 5), (H, 6)\}$

$|S_E| = n_1 * n_2 = 2 * 6 = 12.$

Counting in the context of probability depends on the description of the random experiment and its characteristics described by the way in which the experiment is performed.

Sampling is the operation of drawing randomly from a set one or more of its elements. The set of possible outcomes of this experiment depends on how we perform the draw.

1. *With or without replacement* : in the case the draw is made with replacement, a given element may be drawn several times in the same sample because the drawn element is returned after each draw.

2. *With or without order* : if the draw is performed with order, the elements of the same sample are drawn one after another (sequence) otherwise they are said to be chosen simultaneously. In other words, the result $\{a_1, a_2, a_3, a_4\}$ is different from $\{a_3, a_2, a_4, a_1\}$ in the case of an ordered draw, but identical if the draw is without order.

Therefore, four types of draws are possible: ordered with replacement, ordered without replacement, unordered with replacement and unordered without replacement.

A.2 List (draw with order and replacement)

In this experiment, we have a set of n objects, and we want to draw k objects so that the order is important and that after each draw, the object is put back into the set. The result is called a *list*.
Here, the list to choose has k positions, and there are n options for each position and because the order is important, the total number of ways to choose k elements from a set of n elements is:

$$|S_E| = \underbrace{n \times n \times ... \times n}_{\text{k positions}} = n^k$$

Example 2. *Consider $A = \{1, 2, 3\}$. The number of lists of size 2 chosen from the elements of A is :*
$|S_E| = 3^2 = 9$ different possibilities :

$$S_E = \{(1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (2, 3), (3, 1), (3, 2), (3, 3)\}$$

A.3 Arrangement and permutation (draw with order and without replacement)

A list of k elements among n elements where replacement is not allowed is called *k-permutation or arrangement* of n elements and noted A_n^k. It is interpreted as the number of injective (one to one) mappings from a set of k elements to a set of n elements. Here, the list contains k positions, for the first one, we have n possibilities, in the 2^{nd} position we have $n - 1$ possibilities (because the first element drawn is not put back) , ..., and for the k^{th} position there are $n - k + 1$. So the total number of possible ways is:

$$A_n^k = n \times (n - 1) \times (n - 2) \times ... \times (n - k + 1) = \frac{n!}{(n - k)!}, \quad \text{for } 0 \leq k \leq n$$

When $k = n$, it is called *permutation:* $P_n = n!$. It can also be interpreted as the number of bijections from a set of n elements to itself.

Example 3. *Consider $A = \{1, 2, 3\}$. The number of permutations of size 2 chosen among the elements of A is $|S_E| = P_3^2 = \frac{3!}{(3-2)!} = \frac{3\times2}{1} = 6$ different possibilities :*

$$S_E = \{(1, 2), (1, 3), (2, 1), (2, 3), (3, 1), (3, 2)\}$$

A.4 Combination (draw without order and without replacement)

Here we have a set of n elements, and we derive a subset (the order is not important) of k elements. This experiment is called a *combination* of k elements among n and noted C_n^k. The number of subsets of 2^Ω of cardinality k is the number of combinations of k elements chosen from n elements :

$$C_n^k = \binom{n}{k} = \frac{n!}{(n - k)!k!}, \text{ for } 0 \leq k \leq n$$

C_n^k is also called the *binomial coefficient* which is the number of binary partitions (binomials of complementary subsets of cardinality k and $n-k$ elements). It can be interpreted as the number of possible ways to divide n objects into two groups of sizes k and $n - k$. It also represents the coefficients of the expanded terms of Newton's binomial.

$$(a + b)^n = \sum_0^n C_n^k a^{n-k} b^k$$

Example 4. *Consider $A = \{1, 2, 3\}$. The number of size 2 combinations chosen from the elements of A is $|S| = C_3^2 = 3 \times 2/2 = 3$ different possibilities :*

$$S = \{\{1, 2\}, \{1, 3\}, \{2, 3\}\}$$

Multinomial coefficient is a generalization of the binomial coefficient and which represents the number of possible ways to divide n objects into m groups (subsets) of sizes k_1, k_2,...,k_m (an ordered partition) and which is equal to:

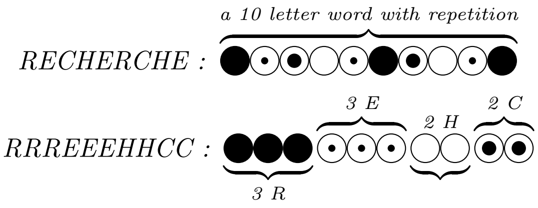
$$C_n^{k_1,k_2,...,k_m} = \binom{n}{k_1, k_2, ..., k_m} = \frac{n!}{k_1!k_2!...k_m!}$$

It can be considered as the number of permutations of n elements taken from the set $\{e_1, e_2, \cdots, e_m\}$ where each element e_i is repeated k_i times.
The nominator $n!$ is the number of total permutations if we consider that the n elements are discernible. To eliminate repeated arrangements, we divide by the number of possible permutations of each element). This is the number of m-partitions (m-tuples of parts of size $k_1, k_2, ..., k_m$ elements). It also represents the coefficients of the expanded terms of Newton's multinomial.

$$(a_1 + a_2 + \cdots + a_m)^n = \sum_{k_1+k_2+\cdots+k_m=n} C_n^{k_1,k_2,...,k_r} a_1^{k_1} a_2^{k_2} \cdots a_m^{k_m}$$

Example 5. *Consider the word 'RECHERCHER' what is the number of arrangements of the letters of this word (words that can be composed)?*
A positioning is made up of 3 E, 3 R, 2 C, 2 H. There are 4 partitions E,R,C,H of respective sizes 3, 3, 2, 2, therefore the number of these partitions is the number of permutations with repetition of the 10 characters of this word; then the corresponding multinomial coefficient is:

$$C_{10}^{3,3,2,2} = \frac{10!}{3!.3!.2!.2!} = 25200$$

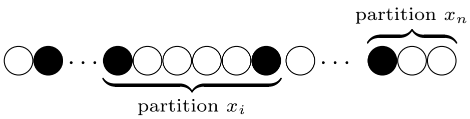


A.5 Combination with repetition (draw without order with replacement)

k-combination with replacement is a sampling of a set $A = \{a_1, a_2, \cdots, a_n\}$ k times so that replacement is allowed and the order is irrelevant. The number of results of this experiment is equal to the number of solutions of the equation : $x_1 + x_2 + \cdots + x_n = k$, such that $x_i \in \mathbb{N}$ is the number of times a_i appears. This number is equal to

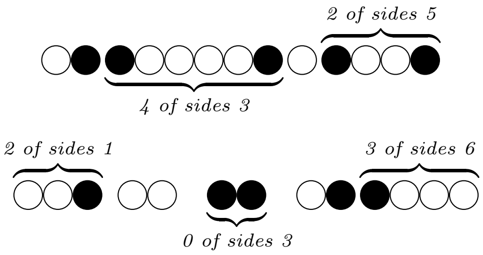
$$C_{n+k-1}^k = \frac{n+k-1}{k}$$

An intuitive method which justifies this result is to consider k white balls and $n-1$ black balls. each arrangement of these balls represents a combination with repetition where the number of white balls between the i^{th} and the $(i+1)^{th}$ black ball represents the repetition of element e_i. The non-repeating k-combination of the positions of the black balls among $\{1, 2, \cdots, n+k-1\}$ represents uniquely the k-combination with repeating n elements.



Example 6. *We have 10 million DA and we want to invest this amount in 5 projects, what is the number of possibilities of this investment?*
x_i is the amount invested in the project p_i. We have : $x_1+x_2+x_3+x_4+x_5 = 10$, such that $x_i \in \mathbb{N}$. So we look for the number of solutions of this equation which is equal to $C_{5+10-1}^{10} = 14!/10!.4! = 1001$

Example 7. *Consider the random experiment of rolling 8 dice, and the result as the tuple of the number of occurrences of each face, what is the cardinality of the sample space of this experiment?*
The tuple $(o_1, o_2, o_3, o_4, o_5, o_6)$ s.t o_i is the number of occurrences of side i. Since we have 8 rolled dice then $\sum_{i=1}^{6} o_i = 8$, a tuple represents 8-combination with repetition of 6 elements. The following figure shows two combinations: the first one represents the tuplet $(1, 0, 4, 1, 2, 0)$ and the second one $(2, 2, 0, 1, 0, 3)$.



The total number is :

$$C_{6+8-1}^{8} = \frac{13!}{8!5!} = \frac{13.12.11.10.9}{5.4.3.2} = 1287$$

Here is a summary table of the four previous cases:

Sampling	With replacement	Without replacement
With order	n^k	$P_n^k = \frac{n!}{(n-k)!}$ case $k = n, P_n = n!$
Without order	$C_{n+k-1}^k = \frac{(n+k-1)!}{k!(n-1)!}$	$C_n^k = \frac{k!(n-k)!}{n!}$

Table A.1: Combinatorial analysis

Appendix B

Queuing models with priority

In some systems, clients have different priorities, for example, in a task scheduling system, tasks are executed according to their importance. Servers serve the highest priority clients first, and if multiple clients in the queue have the same priority then FIFO discipline is applied.
We define N priority classes, the first is the highest priority and the last is the lowest priority. In the case of preemptive priority, the client who is being served is interrupted if a higher priority client arrives at the system, whereas in the case of non-preemptive priority, the service of the current client is completed before passing to the client with the highest priority.
With an exponential distribution service time (which has the memoryless property), we don't care about the remaining service time of an interrupted client.
The average waiting time of priority class k in a non-preemptive priority system is:

$$W_q^{(k)} = \frac{1}{UV_{k-1}V_k} \quad for \quad k = 1, ..., N.$$

So that:

$$U = s!\frac{s\mu - \lambda}{\rho^s}\sum_{i=0}^{s-1}\frac{\rho^i}{i!} + s\mu$$

$$V_k = 1 - \frac{\sum_{i=1}^{k}\lambda_i}{s\mu} \quad \text{and} \quad V_0 = 1$$

$$\lambda_i : \text{ arrival rate of priority } i \text{ clients .}$$

$$\lambda = \sum_{i=1}^{N}\lambda_i \quad (\lambda < s\mu \text{ at the steady state.})$$

L, L_q and $W^{(k)}$ can be obtained by the same relations seen previously.
In the case of a preemptive priority system, the average waiting time in a k class queue is:

$$W_q^{(k)} = \frac{1/\mu}{V_{k-1}V_k} - \frac{1}{\mu} \text{ for } k = 1, ..., N.$$

In some systems the average service times is different from a class to another, if there is only one server and the system is non-preemptive, then the average waiting time is given by:

$$W_q^{(k)} = \frac{U_k}{V_{k-1}V_k} \quad for \quad k = 1, ..., N.$$

So that:

$$U_k = \sum_{i=1}^{k}\frac{\lambda_i}{\mu_i^2}$$

$$V_k = 1 - \sum_{i=1}^{k}\frac{\lambda_i}{\mu_i} \quad \text{and} \quad V_0 = 1$$

$$\mu_i : \text{ arrival rate of priority } i \text{ clients.}$$

B.1 Queuing networks

A *queuing network* is a system where several individual queues are interconnected. The serial queuing system is the simplest model of this class. A queuing network is said *open* if the clients leave the system after receiving the desired services, otherwise it is said *closed* (clients flow between stations).
The equivalence property states that if the arrivals of customers at a station are Poissonian with rate λ, then the departure rate in the steady state is also Poissonian with rate λ.
The direct consequence of this property is that if, at the end of this station, customers must go directly through another station, then the latter will also have Poisson arrivals with the same rate.

B.1.1 Serial queuing systems

Consider a set of m stations of type $M/M/S$ linearly connected and each station satisfies $s_i \times \mu_i > \lambda$ such that s_i is the number of servers in station i, μ_i is the service rate of station i, λ is the arrival rate of customers. Each client visits the m stations in the same order of their connection. This system is a serie of infinite queues. Each station is studied as an independent $M/M/S$ system, and the complete set of queues is studied as follows:

The joint probability $\pi_{(n_1,n_2,...,n_m)}$ of having n_1 customers in station1, n_2 customers in station2,...,n_m customers in station m at the steady state is the product of the individual probabilities:

$$\pi_{(n_1,n_2,...,n_m)} = \prod_{i=1}^{m} \pi^1_{n_1}\pi^2_{n_2}...\pi^m_{n_m} = \prod_{i=1}^{m}(1-\rho_i)\rho_i^{n_i}$$

The average number of L customers in the system is the sum of the L_i in all stations; and the average waiting time W in the system is the average of W_i for $i = 1, ..., m$.
If stations have queues of limited size, they are not considered independent, so the previous results do not apply.

B.1.2 Jackson networks

A Jackson network is an open queuing network described by a set of independent M/M/S stations, and the steady state probability is calculated by the product of the individual probabilities. The main difference with the serial queuing case is that in a Jackson network customers may visit stations in a different order and may not visit all stations. So each station can receive customers coming from any other station or from outside.
The arrival rate of customers at station i is:

$$\lambda_i = e_i + \sum_{j=1}^{m} \lambda_j p_{ji}$$

so that:
e_i is the rate of customers arriving directly from outside station i.
p_{ji} is the probability that a customer leaving station j goes to station i.
$q_j = 1 - \sum_{i=1}^{m} p_{ji}$ is the probability that the customer leaves the system when leaving station j. This formula comes from the fact that the process of arrival at station i is the sum of several Poisson processes, it is therefore a Poisson process whose parameter is the sum of the parameters of the processes that compose it (exterior + other stations), and since the departure rate of each station j is the same as its arrival rate then the arrival rate at station i from station j is $\lambda_j p_{ji}$.
For the stationary state, the condition $\lambda_i < s_i\mu_i$ must be satisfied.
s_i is the number of servers of station i, and μ_i is its service rate.
The values of λ_i for $1 \leq i \leq m$ are obtained by solving the system of linear equations: $\Lambda = \mathbf{e} + \Lambda P$. So that:

$$\Lambda = (\lambda_1, \lambda_2, ..., \lambda_m).$$
$$\mathbf{e} = (e_1, e_2, ..., e_m)$$
$$\mathbf{P} = [p_{ij}]_{1 \leq i,j \leq m}.$$

The solution of the system is: $\Lambda = \mathbf{e}(\mathbf{I} - \mathbf{P})^{-1}$
The matrix $\mathbf{I} - \mathbf{P}$ is invertible because the Jackson networks are open systems. The system's performances are calculated in the same way as described for serial stations.

Example 8. *Consider two queuing stations connected by a network. The service time is exponential of rate 8 for the server of the first station and rate 10 for the second. Customers arrive from outside following a Poisson process of rate 4 towards the first station and rate 5 towards the second. Customers leaving station 1 go to the second station with probability 0.5, and those leaving the second go to the first with probability 0.25.*
Determine the steady-state probabilities of the network, the average number of customers in the system, and the average waiting time.

Solution
$\mathbf{e} = (4, 5), \quad \boldsymbol{\mu} = (8, 10)$

$$\mathbf{P} = \begin{bmatrix} 0 & 0.5 \\ 0.25 & 0 \end{bmatrix}$$

$$\Lambda = \mathbf{e} + \mathbf{P}\Lambda = (6, 8) \quad \textit{and} \quad \lambda = \Lambda\mathbf{1} = \lambda_1 + \lambda_2 = 14$$

$\rho = (\frac{3}{4}, \frac{4}{5})$
$\pi(n_1, n_2) = \frac{1}{20}(\frac{3}{4})^{n_1}(\frac{4}{5})^{n_2}$
$\mathbf{L} = (\frac{\lambda_1}{\mu_1 - \lambda_1}, \frac{\lambda_2}{\mu_2 - \lambda_2}) = (3, 4), \quad L = L_1 + L_2 = 7 \quad \textit{and} \quad W = \frac{L}{\lambda} = \frac{1}{2}$

B.2 Models with non exponential distributions

In many systems, exponential distribution cannot be used to describe inter-arrival times and service times. This is for example the case when arrivals are previously scheduled (planned) and the service is similar for all customers. However, the analytical method applied to this type of models is more difficult than those based on Poisson processes.

B.2.1 General distribution service

In such systems, customers' service times are independent and have the same probability distribution. This distribution can be of any law, but only the mean μ and the variance σ^2 of the distribution must be known.
For the case of the $M/G/1$ model, the arrivals are Poissonnian with rate λ, the service is of general distribution and there is a single server and a queue of infinite size. The steady state is reached if $\rho = \frac{\lambda}{\mu} < 1$, and the performance of this system is:

$$\pi_0 = 1 - \rho$$
$$W_q = \lambda \frac{\sigma^2 + 1/\mu^2}{2(1 - \rho)}$$

Note that the M/M/1 model is a special case of the M/G/1 where $\sigma^2 = \frac{1}{\mu^2}$.
For the case of $M/G/s$, the performances are still unknown and remain an open research problem.

B.2.2 Degenerative distribution (deterministic) service

This model is used in the case of systems characterized by a fixed (deterministic) and identical service time for all customers.
In the $M/D/1$ model, arrivals are Poissonnian with rate λ and service time is deterministic with

value D with a single server and a queue of infinite size. The service rate $\mu = \frac{1}{D}$.
The performances of this system are:

$$W_q = \frac{\rho}{2\mu(1-\rho)}$$

B.2.3 Erlang distribution service

Erlang distribution is defined by the following density function:

$$f(x) = \frac{(\mu k)^k}{(k-1)!} x^{k-1} e^{-k\mu x} \qquad \text{for } x \geq 0.$$

k is an integer, called the shape parameter. It makes it possible to specify the degree of variability of the service times compared to their average. μ is a real called the intensity parameter. When $k = 1$, this law reduces to the Exponential distribution; and when k takes real values greater than 1, it generalizes to the *Gamma* distribution. Its expectation is $\frac{1}{\mu}$ and its standard deviation $\sigma = \frac{1}{\sqrt{k}\mu}$.
Note that $0 < \sigma < \frac{1}{\mu}$ such that 0 is the standard deviation of the degenerate (deterministic) distribution and $\frac{1}{\mu}$ is that of the exponential distribution, which allows the *Erlang* distribution to represent all cases of intermediate service times between the two.
The sum of k v.a *Exponential* with mean $\frac{1}{k\mu}$ is an *Erlang* r.v with parameters μ and k.
The *Erlang* distribution can be used in the case where the service of each client requires the execution of the same task (of exponential time) k times.
Another major advantage of the *Erlang* distribution is that it can be used as an approximation to empirical distribution service times. Indeed, after estimating the values of the expectation and the variance of the empirical distribution, these are used to choose the value of k which best corresponds to this estimate.

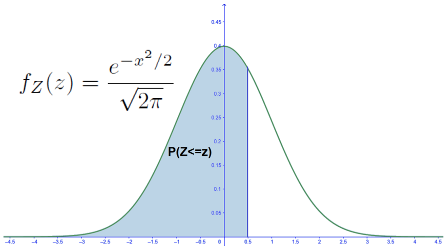
For the $M/E_k/1$ system, the performances are as follows:

$$W_q = \frac{\lambda(1+k)}{2k\mu(\mu-\lambda)}$$

No general solution for the performance of the system $M/E_k/S$ (S>1) exists so far.
For the other types of systems where the arrivals are non-Poissonian, there are very few established results, which led us to omit them from this section. The reader can refer to [12] for more details.

Appendix C

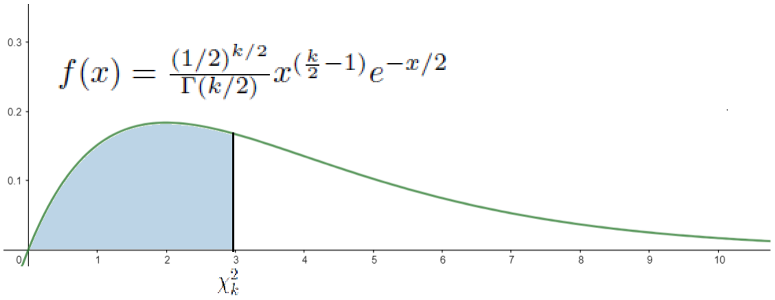
Statistics tables



$$f_Z(z) = \frac{e^{-x^2/2}}{\sqrt{2\pi}}$$

	0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.1	0.1	0.1
0.0	0.5000	0.5040	0.5080	0.5120	0.5160	0.5199	0.5239	0.5279	0.5319	0.5359
0.1	0.5398	0.5438	0.5478	0.5517	0.5557	0.5596	0.5636	0.5675	0.5714	0.5753
0.2	0.5793	0.5832	0.5871	0.5910	0.5948	0.5987	0.6026	0.6064	0.6103	0.6141
0.3	0.6179	0.6217	0.6255	0.6293	0.6331	0.6368	0.6406	0.6443	0.6480	0.6517
0.4	0.6554	0.6591	0.6628	0.6664	0.6700	0.6736	0.6772	0.6808	0.6844	0.6879
0.5	0.6915	0.6950	0.6985	0.7019	0.7054	0.7088	0.7123	0.7157	0.7190	0.7224
0.6	0.7257	0.7291	0.7324	0.7357	0.7389	0.7422	0.7454	0.7486	0.7517	0.7549
0.7	0.7580	0.7611	0.7642	0.7673	0.7704	0.7734	0.7764	0.7794	0.7823	0.7852
0.8	0.7881	0.7910	0.7939	0.7967	0.7995	0.8023	0.8051	0.8078	0.8106	0.8133
0.9	0.8159	0.8186	0.8212	0.8238	0.8264	0.8289	0.8315	0.8340	0.8365	0.8389
1.0	0.8413	0.8438	0.8461	0.8485	0.8508	0.8531	0.8554	0.8577	0.8599	0.8621
1.1	0.8643	0.8665	0.8686	0.8708	0.8729	0.8749	0.8770	0.8790	0.8810	0.8830
1.2	0.8849	0.8869	0.8888	0.8907	0.8925	0.8944	0.8962	0.8980	0.8997	0.9015
1.3	0.9032	0.9049	0.9066	0.9082	0.9099	0.9115	0.9131	0.9147	0.9162	0.9177
1.4	0.9192	0.9207	0.9222	0.9236	0.9251	0.9265	0.9279	0.9292	0.9306	0.9319
1.5	0.9332	0.9345	0.9357	0.9370	0.9382	0.9394	0.9406	0.9418	0.9429	0.9441
1.6	0.9452	0.9463	0.9474	0.9484	0.9495	0.9505	0.9515	0.9525	0.9535	0.9545
1.7	0.9554	0.9564	0.9573	0.9582	0.9591	0.9599	0.9608	0.9616	0.9625	0.9633
1.8	0.9641	0.9649	0.9656	0.9664	0.9671	0.9678	0.9686	0.9693	0.9699	0.9706
1.9	0.9713	0.9719	0.9726	0.9732	0.9738	0.9744	0.9750	0.9756	0.9761	0.9767
2.0	0.9772	0.9778	0.9783	0.9788	0.9793	0.9798	0.9803	0.9808	0.9812	0.9817
2.1	0.9821	0.9826	0.9830	0.9834	0.9838	0.9842	0.9846	0.9850	0.9854	0.9857
2.2	0.9861	0.9864	0.9868	0.9871	0.9875	0.9878	0.9881	0.9884	0.9887	0.9890
2.3	0.9893	0.9896	0.9898	0.9901	0.9904	0.9906	0.9909	0.9911	0.9913	0.9916
2.4	0.9918	0.9920	0.9922	0.9925	0.9927	0.9929	0.9931	0.9932	0.9934	0.9936
2.5	0.9938	0.9940	0.9941	0.9943	0.9945	0.9946	0.9948	0.9949	0.9951	0.9952
2.6	0.9953	0.9955	0.9956	0.9957	0.9959	0.9960	0.9961	0.9962	0.9963	0.9964
2.7	0.9965	0.9966	0.9967	0.9968	0.9969	0.9970	0.9971	0.9972	0.9973	0.9974
2.8	0.9974	0.9975	0.9976	0.9977	0.9977	0.9978	0.9979	0.9979	0.9980	0.9981
2.9	0.9981	0.9982	0.9982	0.9983	0.9984	0.9984	0.9985	0.9985	0.9986	0.9986
3.0	0.9987	0.9987	0.9987	0.9988	0.9988	0.9989	0.9989	0.9989	0.9990	0.9990
3.1	0.9990	0.9991	0.9991	0.9991	0.9992	0.9992	0.9992	0.9992	0.9993	0.9993
3.2	0.9993	0.9993	0.9994	0.9994	0.9994	0.9994	0.9994	0.9995	0.9995	0.9995
3.3	0.9995	0.9995	0.9995	0.9996	0.9996	0.9996	0.9996	0.9996	0.9996	0.9997
3.4	0.9997	0.9997	0.9997	0.9997	0.9997	0.9997	0.9997	0.9997	0.9997	0.9998
3.5	0.9998	0.9998	0.9998	0.9998	0.9998	0.9998	0.9998	0.9998	0.9998	0.9998
3.6	0.9998	0.9998	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999
3.7	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999

Table 1 Normal Distribution



dl	$\chi^2_{0.005}$	$\chi^2_{0.01}$	$\chi^2_{0.025}$	$\chi^2_{0.05}$	$\chi^2_{0.1}$	$\chi^2_{0.9}$	$\chi^2_{0.95}$	$\chi^2_{0.975}$	$\chi^2_{0.99}$	$\chi^2_{0.995}$
1	.0000	.0002	.0010	.0039	.0158	2.706	3.841	5.024	6.635	7.879
2	.0100	.02010	.0506	.1026	.2107	4.605	5.991	7.378	9.210	10.60
3	.0717	.1148	.2158	.3518	.5844	6.251	7.815	9.348	11.34	12.84
4	.207	.2971	.4844	.7107	1.064	7.779	9.488	11.14	13.28	14.86
5	.4117	.5543	.8312	1.145	1.610	9.236	11.07	12.83	15.09	16.75
6	.6757	.8721	1.237	1.635	2.204	10.64	12.59	14.45	16.81	18.55
7	.9893	1.239	1.690	2.167	2.833	12.02	14.07	16.01	18.48	20.28
8	1.344	1.646	2.180	2.733	3.490	13.36	15.51	17.53	20.09	21.95
9	1.735	2.088	2.700	3.325	4.168	14.68	16.92	19.02	21.67	23.59
10	2.156	2.558	3.247	3.940	4.865	15.99	18.31	20.48	23.21	25.19
11	2.603	3.053	3.816	4.575	5.578	17.28	19.68	21.92	24.72	26.76
12	3.074	3.571	4.404	5.226	6.304	18.55	21.03	23.34	26.22	28.30
13	3.565	4.107	5.009	5.892	7.042	19.81	22.36	24.74	27.69	29.82
14	4.075	4.660	5.629	6.571	7.790	21.06	23.68	26.12	29.14	31.32
15	4.601	5.229	6.262	7.261	8.547	22.31	25.00	27.49	30.58	32.80
16	5.142	5.812	6.908	7.962	9.312	23.54	26.30	28.85	32.00	34.27
17	5.697	6.408	7.564	8.672	10.09	24.77	27.59	30.19	33.41	35.72
18	6.265	7.015	8.231	9.390	10.86	25.99	28.87	31.53	34.81	37.16
19	6.844	7.633	8.907	10.12	11.65	27.2	30.14	32.85	36.19	38.58
20	7.434	8.260	9.591	10.85	12.44	28.41	31.41	34.17	37.57	40.00
22	8.643	9.542	10.98	12.34	14.04	30.81	33.92	36.78	40.29	42.80
24	9.886	10.86	12.40	13.85	15.66	33.20	36.42	39.36	42.98	45.56
26	11.16	12.20	13.84	15.38	17.29	35.56	38.89	41.92	45.64	48.29
28	12.46	13.56	15.31	16.93	18.94	37.92	41.34	44.46	48.28	50.99
30	13.79	14.95	16.79	18.49	20.60	40.26	43.77	46.98	50.89	53.67
35	17.19	18.51	20.57	22.47	24.80	46.06	49.80	53.20	57.34	60.27
40	20.71	22.16	24.43	26.51	29.05	51.81	55.76	59.34	63.69	66.77
45	24.31	25.90	28.37	30.61	33.35	57.51	61.66	65.41	69.96	73.17
50	27.99	29.71	32.36	34.76	37.69	63.17	67.50	71.42	76.15	79.49
60	35.53	37.48	40.48	43.19	46.46	74.40	79.08	83.30	88.38	91.95
70	43.28	45.44	48.76	51.74	55.33	85.53	90.53	95.02	100.4	104.2
80	51.17	53.54	57.15	60.39	64.28	96.58	101.9	106.6	112.3	116.3
90	59.20	61.75	65.65	69.13	73.29	107.6	113.1	118.1	124.1	128.3
100	67.33	70.06	74.22	77.93	82.36	118.5	124.3	129.6	135.8	140.2
120	83.85	86.92	91.57	95.70	100.6	140.2	146.6	152.2	159.0	163.6

Table 2 Chi-square Distribution

	Significance level, α			
Trials number, n	0.1	0.05	0.02	0.01
1	0.95000	0.97500	0.99000	0.99500
2	0.77639	0.84189	0.90000	0.92929
3	0.63604	0.70760	0.78456	0.82900
4	0.56522	0.62394	0.68887	0.73424
5	0.50945	0.56328	0.62718	0.66853
6	0.46799	0.51926	0.57741	0.61661
7	0.43607	0.48342	0.53844	0.57581
8	0.40962	0.45427	0.50654	0.54179
9	0.38746	0.43001	0.4796	0.51332
10	0.36866	0.40925	0.45662	0.48893
11	0.35242	0.39122	0.43670	0.46770
12	0.33815	0.37543	0.41918	0.44905
13	0.32549	0.36143	0.40362	0.43247
14	0.31417	0.34890	0.38970	0.41762
15	0.30397	0.33760	0.37713	0.40420
16	0.29472	0.32733	0.36571	0.39201
17	0.28627	0.31796	0.35528	0.38086
18	0.27851	0.30936	0.34569	0.37062
19	0.27136	0.30143	0.33685	0.36117
20	0.26473	0.29408	0.32866	0.35241
21	0.25858	0.28724	0.32104	0.34427
22	0.25283	0.28087	0.31394	0.33666
23	0.24746	0.27490	0.30728	0.32954
24	0.24242	0.26931	0.30104	0.32286
25	0.23768	0.26404	0.29516	0.31657
26	0.23320	0.25907	0.28962	0.31064
27	0.22898	0.25438	0.28438	0.30502
28	0.22497	0.24993	0.27942	0.29971
29	0.22117	0.24571	0.27471	0.29466
30	0.21756	0.24170	0.27023	0.28987
31	0.21412	0.23788	0.26596	0.28530
32	0.21085	0.23424	0.26189	0.28094
33	0.20771	0.23076	0.25801	0.27677
34	0.20472	0.22743	0.25429	0.27279
35	0.20185	0.22425	0.26073	0.26897
36	0.19910	0.22119	0.24732	0.26532
37	0.19646	0.21826	0.24404	0.26180
38	0.19392	0.21544	0.24089	0.25843
39	0.19148	0.21273	0.23786	0.25518
40	0.18913	0.21012	0.23494	0.25205

Table 3 Kolmogorov-Smirnov Distribution

Bibliography

[1] Rational sequence increasing to real number. https://proofwiki.org/wiki/Rational_Sequence_Increasing_to_Real_Number.

[2] A.Aissani. *Modelisation et simulation*. OPU, Algerie, 2007.

[3] W.D.Kelton A.M.Law. *Simulation, Modeling and analysis, 2nd edition*. McGraw-Hill, Inc., 1991.

[4] I. D. Hill B.A. Wichmann. Algorithm as183 an efficient and portable pseudo-random number generator. *Journal of the Royal Statistical Society*, 31(2):188–190, 1982.

[5] D.Knuth. *The Art of Computer Programming volume 2: Seminumerical algorithms, Generating Uniform Random Numbers*. Addison-Wesley, 1981.

[6] G.J.Lieberman F.S.Hillier. *Introduction to operations research, 6th edition*. McGraw-Hill, Inc., 1995.

[7] S.Lafortune G.C.Cassandras. *Introduction to discrete event systems, 2nd edition*. Springer, 2008.

[8] G.Marsaglia. Random numbers fall mainly in the planes. *PNAS*, 61(1):25–28, Sept 1, 1968.

[9] H.Pishro-Nik. Introduction to probability, statistics and random processes. https://www.probabilitycourse.com, 2014.

[10] H.Tijms. *Probability: a lively introduction*. Cambridge University Press, 2017.

[11] M. j. Neely. On probabilty axioms and sigma algebras. https://ee.usc.edu/stochastic-nets/docs/probability-axioms-sigma-algebras.pdf, 2019.

[12] D.Gross C.M.Harris J.F.Shortle, J.M.Thompson. *Fundamentals of queueing theory, 5th edition*. Wiley, 2017.

[13] J.Unpingco. *Python for Probability, Statistics ans Machicne Learning*. Springer, 2019.

[14] K.Siegrist. Random services. https://www.randomservices.org.

[15] P. L'Ecuyer. *Uniform Random Number Generation*. Springer, Annals of Operations Research 53(1):77-120, 1994.

[16] T. Nishimura M. Matsumot. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 1(1):3–30, Jan 1, 1998.

[17] M.Harchol-Balter. *Performance Modeling and Design of Computer Systems. Queueing theory in Action*. Cambridge University Press, 2013.

[18] T. Oliphant. Numpy project. https://www.numpy.org.

[19] G. Van Rossum. Python api. http://www.python.org/doc/api/api.html.

[20] S.M.Ross. *A first course in probability, 5th edition*. Prentice Hall, Inc., 1998.

[21] S.M.Ross. *Introduction to probability models, 10th edition.* Academic Press. Elsevier, 2009.

[22] A. R. Dobell T. E. Hull. Random number generators. *SIAM Review*, 4(3):230–254, 1962.

In the age of artificial intelligence, robotics, intelligent systems and machine learning, which are increasingly taking up space in practically all areas; and since the emergence of intelligent applications on which most of the major IT players rely; in this great technological whirlwind, we ask the computer scientist to know, create and innovate more and more. However, the development of the necessary skills to master these recent technologies cannot be achieved without having the essential bases in probability, which is one of their fundamental pillars. Mastering the foundations of this pillar has thus become essential.

It is with this in mind that we have proposed this book on probability, stochastic processes and simulation, and the challenge of which is to simplify the basic concepts by the example and by the practice.

This book is intended primarily for computer scientists, but can obviously be used by anyone wishing to learn the elementary concepts of probability with their practical side in order to subsequently be able to tackle other more advanced subjects such as machine learning and artificial intelligence.