Introduction to discrete probabilities with Scilab

Michaël Baudin

January 2010

Abstract

In this article, we present an introduction to discrete probabilities with Scilab. Numerical experiments are based on Scilab. The first section present discrete random variables and conditionnal probabilities. In the second section, we present combinations problems, tree diagrams and Bernouilli trials. In the third section, we present simulation of random processes with Scilab.

Contents

1 Discrete random variables **2**
- 1.1 Sets . . . 2
- 1.2 Distribution function and probability . . . 4
- 1.3 Properties of discrete probabilities . . . 5
- 1.4 Uniform distribution . . . 8
- 1.5 Conditional probability . . . 9

2 Combinatorics **12**
- 2.1 Tree diagrams . . . 12
- 2.2 Permutations . . . 13
- 2.3 The gamma function . . . 16
- 2.4 Overview of functions in Scilab . . . 19
- 2.5 The gamma function in Scilab . . . 19
- 2.6 The factorial and log-factorial functions . . . 20
- 2.7 Computing factorial and log-factorial with Scilab . . . 21
- 2.8 Computing permutations and log-permutations with Scilab . . . 23
- 2.9 Combinations . . . 26
- 2.10 Computing combinations with Scilab . . . 27
- 2.11 The poker game . . . 29
- 2.12 Bernoulli trials . . . 31

3 Simulation of random processes with Scilab **33**
- 3.1 Overview . . . 33
- 3.2 Generating uniform random numbers . . . 34
- 3.3 Simulating random discrete events . . . 35
- 3.4 Simulation of a coin . . . 37

4 Acknowledgments 38

5 References and notes 38

Bibliography 39

Index 39

1 Discrete random variables

In this section, we present discrete random variables. The first section presents general definition for sets, including union, intersection. Then we present the definition of the discrete distribution function and the probability of an event. In the third section, we give properties of probabilities, such as, for example, the probability of the union of two disjoints events. The fourth section is devoted to the very common discrete uniform distribution function. Then we present the definition of conditional probability.

1.1 Sets

A *set* is a collection of elements. In this document, we consider sets of elements in a fixed non empty set Ω, to be called a *space*.

Assume that A is a set of elements. If x is a point in that set, we note $x \in A$. If there is no point in A, we write $A = \emptyset$. If the number of elements in A is finite, let us denote by $\#(A)$ the number of elements in the set A. If the number of elements in A is infinite, the cardinality cannot be computed (for example $A = \mathbb{N}$).

The set A^c is the set of all points in Ω which are not in A:

$$A^c = \{x \in \Omega \,/\, x \notin A\}\,. \tag{1}$$

The set A^c is called the *complementary* set of A.

The set B is a *subset* of A if any point in B is also in A and we can write $A \subset B$. The two sets A and B are equal if $A \subset B$ and $B \subset A$. The difference set $A - B$ is the set of all points of A which are not in B:

$$A - B = \{x \in A \,/\, x \notin B\}\,. \tag{2}$$

The *intersection* $A \cap B$ of two sets A and B is the set of points common to A and B:

$$A \cap B = \{x \in A \text{ and } x \in B\}\,. \tag{3}$$

The *union* $A \cup B$ of two sets A and B is the set of points which belong to at least one of the sets A or B:

$$A \cup B = \{x \in A \text{ or } x \in B\}\,. \tag{4}$$

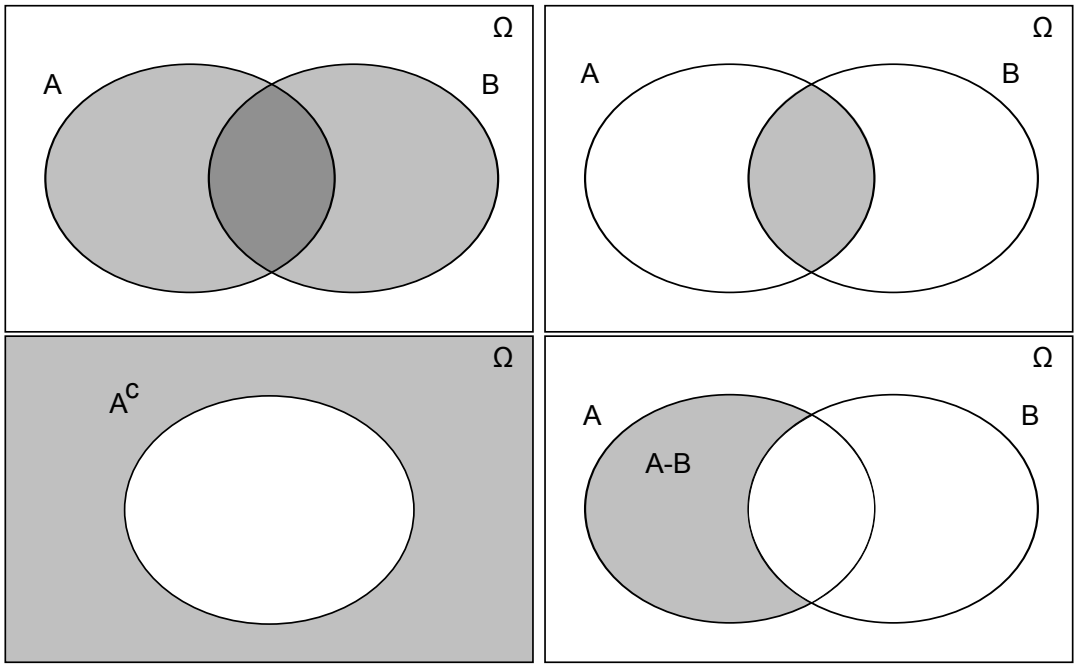

Figure 1: Operations on sets – Upper left, union: $A \cup B$, Upper right, intersection: $A \cap B$, Lower left, complement: A^c, Lower right, difference: $A - B$

The operations that we defined are presented in figure 1. These figures are often referred to as Venn's diagrams.

Two sets A and B are *disjoints*, or *mutually exclusive* if their intersection is empty, i.e. $A \cap B = \emptyset$.

In the following, we will use the fact that we can always decompose the union of two sets as the union of three disjoints subsets. Indeed, assume that $A, B \subset \Omega$. We have

$$A \cup B = (A - B) \cup (A \cap B) \cup (B - A), \tag{5}$$

where the sets $A - B$, $A \cap B$ and $B - A$ are disjoints. This decomposition will be used several times in this chapter.

The cross product of two sets is the set

$$A \times B = \{(x, y) \ / x \in A, y \in B\}. \tag{6}$$

Assume that n is a positive integer. The power set A^n is the set

$$A^n = \{(x_1, \ldots, x_n) \ / x_1, ..., x_n \in A\}. \tag{7}$$

Example 1.1 *(Die with 6 faces)* Assume that a 6-face die is rolled once. The sample space for this experiment is

$$\Omega = \{1, 2, 3, 4, 5, 6\}. \tag{8}$$

The set of even numbers is $A = \{2, 4, 6\}$ and the set of odd numbers is $B = \{1, 3, 5\}$. Their intersection is empty, i.e. $A \cap B = \emptyset$ which proves that A and B are disjoints. Since their union is the whole sample space, i.e. $A \cup B = \Omega$, these two sets are mutually complement, i.e. $A^c = B$ and $B^c = A$.

1.2 Distribution function and probability

A random event is an event which has a chance of happening, and the probability is a numerical measure of that chance. What exactly is random is quite difficult to define. In this section, we define the *probability* associated with a distribution function, a concept that can be defined very precisely.

Assume that Ω is a set, called the *sample space*. In this document, we will consider the case where the sample space is finite, i.e. the number of elements in Ω is finite. Assume that we are performing *random* trials, so that each trial is associated to one *outcome* $x \in \Omega$. Each subset A of the sample space Ω is called an *event*. We say that event $A \cap B$ occurs if both the events A and B occur. We say that event $A \cup B$ occurs if the event A or the event B occurs.

We will first define a *distribution function* and then derive the *probability* from there. The example of a 6-faces die will serve as an example for these definitions.

Definition 1.1. (Distribution) *A distribution function is a function* $f : \Omega \to [0, 1]$ *which satisfies*

$$0 \leq f(x) \leq 1, \tag{9}$$

for all $x \in \Omega$ *and*

$$\sum_{x \in \Omega} f(x) = 1. \tag{10}$$

Example 1.2 *(Die with 6 faces)* Assume that a 6-face die is rolled once. The sample space for this experiment is

$$\Omega = \{1, 2, 3, 4, 5, 6\}. \tag{11}$$

Assume that the die is fair. This means that the probability of each of the six outcomes is the same, i.e. the distribution function is $f(x) = 1/6$ for $x \in \Omega$, which satisfies the conditions of the definition 1.1.

Definition 1.2. (Probability) *Assume that* f *is a distribution function on the sample space* Ω*. For any event* $A \subset \Omega$*, the probability* P *of* A *is*

$$P(A) = \sum_{x \in A} f(x). \tag{12}$$

Example 1.3 *(Die with 6 faces)* Assume that a 6-face die is rolled once so that the sample space for this experiment is $\Omega = \{1, 2, 3, 4, 5, 6\}$. Assume that the distribution function is $f(x) = 1/6$ for $x \in \Omega$. The event

$$A = \{2, 4, 6\} \tag{13}$$

corresponds to the statement that the result of the roll is an even number. From the definition 1.2, the probability of the event A is

$$\begin{aligned} P(A) &= f(2) + f(4) + f(6) && (14) \\ &= \frac{1}{6} + \frac{1}{6} + \frac{1}{6} = \frac{1}{2}. && (15) \end{aligned}$$

1.3 Properties of discrete probabilities

In this section, we present the properties that the probability $P(A)$ satisfies. We also derive some results for the probabilities of other events, such as unions of disjoints events.

The following theorem gives some elementary properties satisfied by a probability P.

Proposition 1.3. (Probability) *Assume that Ω is a sample space and that f is a distribution function on Ω. Assume that P is the probability associated with f. The probability of the event Ω is one, i.e.*

$$P(\Omega) = 1. \tag{16}$$

The probability of the empty set is zero, i.e.

$$P(\emptyset) = 0. \tag{17}$$

Assume that A and B are two subsets of Ω. If $A \subset B$, then

$$P(A) \leq P(B). \tag{18}$$

For any event $A \subset \Omega$, we have

$$0 \leq P(A) \leq 1. \tag{19}$$

Proof. The equality 16 derives directly from the definition 10 of a distribution function, i.e. $P(\Omega) = \sum_{x \in \Omega} f(x) = 1$. The equality 17 derives directly from 10.

Assume that A and B are two subsets of Ω so that $A \subset B$. Since a probability is the sum of positive terms, we have

$$P(A) = \sum_{x \in A} f(x) \leq \sum_{x \in B} f(x) = P(B) \tag{20}$$

which proves the inequality 18.

The inequalities 19 derive directly from the definition of a probability 12. First, the probability P is positive since 9 states that f is positive. Second, the probability P of an event A is lower than 1, since $P(A) = \sum_{x \in A} f(x) \leq \sum_{x \in \Omega} f(x) = P(\Omega) = 1$, which concludes the proof. □

Proposition 1.4. (Probability of two disjoint subsets) *Assume that Ω is a sample space and that f is a distribution function on Ω. Assume that P is the probability associated with f.*

Let A and B be two disjoints subsets of Ω, then

$$P(A \cup B) = P(A) + P(B). \tag{21}$$

The figure 2 presents the situation of two disjoints sets A and B. Since the two sets have no intersection, it suffices to add the probabilities associated with each event.

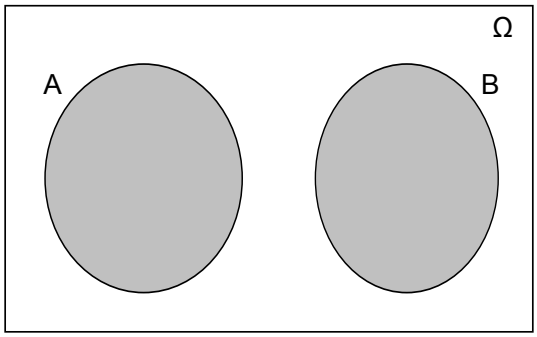

Figure 2: Two disjoint sets.

Proof. Assume that A and B be two disjoints subsets of Ω. We can decompose $A \cup B$ as $A \cup B = (A - B) \cup (A \cap B) \cup (B - A)$, so that

$$P(A \cup B) = \sum_{x \in A \cup B} f(x) \tag{22}$$

$$= \sum_{x \in A - B} f(x) + \sum_{x \in A \cap B} f(x) + \sum_{x \in B - A} f(x). \tag{23}$$

But A and B are disjoints, so that $A - B = A$, $A \cap B = \emptyset$ and $B - A = B$. Therefore,

$$P(A \cup B) = \sum_{x \in A} f(x) + \sum_{x \in B} f(x) \tag{24}$$

$$= P(A) + P(B), \tag{25}$$

which concludes the proof. □

Notice that the equality 21 can be generalized immediately to a sequence of disjoints events.

Proposition 1.5. (Probability of disjoints subsets) *Assume that Ω is a sample space and that f is a distribution function on Ω. Assume that P is the probability associated with f.*

For any disjoints events $A_1, A_2, \ldots, A_k \subset \Omega$ with $k \geq 0$, we have

$$P(A_1 \cup A_2 \cup \ldots \cup A_k) = P(A_1) + P(A_2) + \ldots + P(A_k). \tag{26}$$

Proof. For example, we can use the proposition 1.4 to state the proof by induction on the number of events. □

Example 1.4 *(Die with 6 faces)* Assume that a 6-face die is rolled once so that the sample space for this experiment is $\Omega = \{1, 2, 3, 4, 5, 6\}$. Assume that the distribution function is $f(x) = 1/6$ for $x \in \Omega$. The event $A = \{1, 2, 3\}$ corresponds to the numbers lower or equal to 3. The probability of this event is $P(A) = \frac{1}{2}$. The event $B = \{5, 6\}$ corresponds to the numbers greater than 5. The probability of this event is $P(B) = \frac{1}{3}$. The two events are disjoints, so that the proposition 1.5 can be applied which proves that $P(A \cup B) = \frac{5}{6}$.

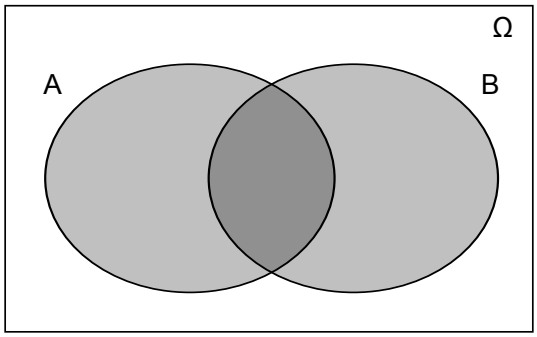

Figure 3: Two sets with a non empty intersection.

Proposition 1.6. (Probability of the complementary event) *Assume that Ω is a sample space and that f is a distribution function on Ω. Assume that P is the probability associated with f.*

For all subset A of Ω,

$$P(A) + P(A^c) = 1. \tag{27}$$

Proof. We have $\Omega = A \cup A^c$, where the sets A and A^c are disjoints. Therefore, from proposition 1.4, we have

$$P(\Omega) = P(A) + P(A^c), \tag{28}$$

where $P(\Omega) = 1$, which concludes the proof. □

Example 1.5 *(Die with 6 faces)* Assume that a 6-face die is rolled once so that the sample space for this experiment is $\Omega = \{1, 2, 3, 4, 5, 6\}$. Assume that the distribution function is $f(x) = 1/6$ for $x \in \Omega$. The event $A = \{2, 4, 6\}$ corresponds to the statement that the result of the roll is an even number. The probability of this event is $P(A) = \frac{1}{2}$. The complementary event is the event of an odd number, i.e. $A^c = \{1, 3, 5\}$. By proposition 1.6, the probability of an odd number is $P(A^c) = 1 - P(A) = 1 - \frac{1}{2} = \frac{1}{2}$.

The following equality gives the relationship between the probability of the union of two events in terms of the individual probabilities and the probability of the intersection.

Proposition 1.7. (Probability of the union) *Assume that Ω is a sample space and that f is a distribution function on Ω. Assume that A and B are two subsets of Ω, not necessarily disjoints. We have:*

$$P(A \cup B) = P(A) + P(B) - P(A \cap B). \tag{29}$$

The figure 3 presents the situation where two sets A and B have a non empty intersection. When we add the probabilities of the two events A and B, the intersection is added twice. This is why it must be removed by subtraction.

Proof. Assume that A and B are two subsets of Ω. The proof is based on the analysis of Venn's diagram presented in figure 1. The idea of the proof is to compute the probability $P(A \cup B)$, by making disjoints sets on which the equality 21 can be applied. We can decompose the union of the two set A and B as the union of disjoints sets:

$$A \cup B = (A - B) \cup (A \cap B) \cup (B - A). \tag{30}$$

The equality 21 leads to

$$P(A \cup B) = P(A - B) + P(A \cap B) + P(B - A). \tag{31}$$

The next part of the proof is based on the computation of $P(A - B)$ and $P(B - A)$. We can decompose the set A as the union of disjoints sets

$$A = (A - B) \cup (A \cap B), \tag{32}$$

which leads to $P(A) = P(A - B) + P(A \cap B)$, which can be written as

$$P(A - B) = P(A) - P(A \cap B). \tag{33}$$

Similarly, we can prove that

$$P(B - A) = P(B) - P(B \cap A). \tag{34}$$

By plugging the two equalities 33 and 34 into 31, we find

$$P(A \cup B) = P(A) - P(A \cap B) + P(A \cap B) + P(B) - P(B \cap A), \tag{35}$$

which simplifies into

$$P(A \cup B) = P(A) + P(B) - P(B \cap A), \tag{36}$$

and concludes the proof. □

Example 1.6 *(Disease)* Assume that the probability of infections can be bacterial (B), viral (V) or both $(B \cap V)$. This implies that $B \cup V = \Omega$ but the two events are not disjoints, i.e. $B \cap V \neq \emptyset$. Assume that $P(B) = 0.7$ and $P(V) = 0.4$. What is the probability of having both types of infections ? The probability of having both infections is $P(B \cap V)$. From proposition 1.7, we have $P(B \cup V) = P(B) + P(V) - P(B \cap V)$, which leads to $P(B \cap V) = P(B) + P(V) - P(B \cup V)$. We finally get $P(B \cap V) = 0.1$. This example is presented in [8].

1.4 Uniform distribution

In this section, we describe the particular situation where the distribution function is *uniform*.

Definition 1.8. (Uniform distribution) *Assume that Ω is a finite sample space. The uniform distribution function is*

$$f(x) = \frac{1}{\#(\Omega)}, \tag{37}$$

for all $x \in \Omega$.

Proposition 1.9. (Probability with uniform distribution) *Assume that Ω is a finite sample space and that f is a uniform distribution function. Then the probability of the event $A \subset \Omega$ is*

$$P(A) = \frac{\#(A)}{\#(\Omega)}. \tag{38}$$

Proof. When the distribution function is uniform, the definition 1.2 implies that

$$\begin{aligned} P(A) &= \sum_{x \in A} f(x) = \sum_{x \in A} \frac{1}{\#(\Omega)} && (39) \\ &= \frac{\#(A)}{\#(\Omega)}, && (40) \end{aligned}$$

which concludes the proof. □

Example 1.7 *(Die with 6 faces)* Assume that a 6-face die is rolled once so that the sample space for this experiment is $\Omega = \{1, 2, 3, 4, 5, 6\}$. In the previous analysis of this example, we have assumed that the distribution function is $f(x) = 1/6$ for $x \in \Omega$. This is consistent with definition 1.8, since $\#(\Omega) = 6$. Such a die is a *fair* die, meaning that all faces have the same probability. The event $A = \{2, 4, 6\}$ corresponds to the statement that the result of the roll is an even number. The number of outcomes in this event is $\#(A) = 3$. From proposition 1.9, the probability of this event is $P(A) = \frac{1}{2}$.

1.5 Conditional probability

In this section, we define the conditional distribution function and the conditional probability. We analyze this definition in the particular situation of the uniform distribution.

In some situations, we want to consider the probability of an event A given that an event B has occurred. In this case, we consider the set B as a new sample space, and update the definition of the distribution function accordingly.

Definition 1.10. (Conditional distribution function) *Assume that Ω is a sample space and that f is a distribution function on Ω. Assume that A is a subset of Ω with $P(A) = \sum_{x \in A} f(x) > 0$. The function $f(x|A)$ defined by*

$$f(x|A) = \begin{cases} \frac{f(x)}{\sum_{x \in A} f(x)}, & \text{if } x \in A, \\ 0, & \text{if } x \notin A, \end{cases} \tag{41}$$

is the conditional distribution function of x given A.

The figure 4 presents the situation where an event A is considered for a conditionnal distribution. The distribution function $f(x)$ is with respect to the sample space Ω while the conditionnal distribution function $f(x|A)$ is with respect to the set A.

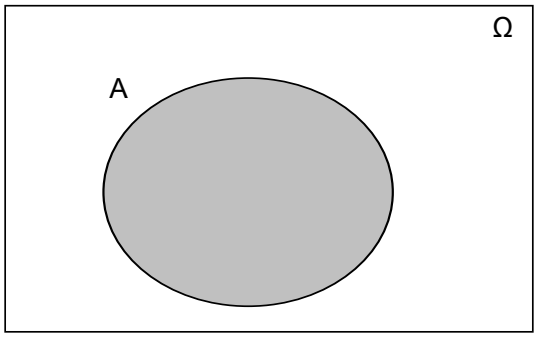

Figure 4: A set A, subset of the sample space Ω.

Proof. What is to be proved in this proposition is that the function $f(x|A)$ is a distribution function. Let us prove that the function $f(x|A)$ satisfies the equality

$$\sum_{x\in\Omega} f(x|A) = 1. \tag{42}$$

Indeed, we have

$$\begin{aligned}
\sum_{x\in\Omega} f(x|A) &= \sum_{x\in A} f(x|A) + \sum_{x\notin A} f(x|A) && (43)\\
&= \sum_{x\in A} \frac{f(x)}{\sum_{x\in A} f(x)} && (44)\\
&= 1, && (45)\\
& && (46)
\end{aligned}$$

which concludes the proof. □

This leads us to the following definition of the conditional probability of an event A given an event B.

Proposition 1.11. (Conditional probability) *Assume that Ω is a finite sample space and A and B are two subsets of Ω. Assume that $P(B) > 0$. The conditional probability of the event A given the event B is*

$$P(A|B) = \frac{P(A\cap B)}{P(B)}. \tag{47}$$

The figure 5 presents the situation where we consider the event $A|B$. The probability $P(A)$ is with respect to Ω while the probability $P(A|B)$ is with respect to B.

Proof. Assume that A and B are subsets of the sample space Ω. The conditional distribution function $f(x|B)$ allows us to compute the probability of the event A given the event B. Indeed, we have

$$\begin{aligned}
P(A|B) &= \sum_{x\in A} f(x|B) && (48)\\
&= \sum_{x\in A\cap B} f(x|B) && (49)
\end{aligned}$$

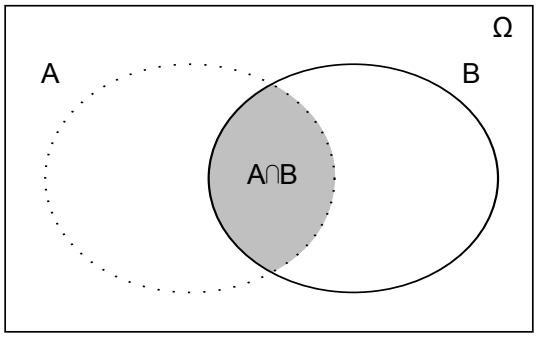

Figure 5: The conditionnal probability $P(A|B)$ measure the probability of the set $A \cap B$ with respect to the set B.

since $f(x|B) = 0$ if $x \notin B$. Hence,

$$P(A|B) = \sum_{x \in A \cap B} \frac{f(x)}{\sum_{x \in B} f(x)} \tag{50}$$

$$= \frac{\sum_{x \in A \cap B} f(x)}{\sum_{x \in B} f(x)} \tag{51}$$

$$= \frac{P(A \cap B)}{P(B)}. \tag{52}$$

The previous equality is well defined since $P(B) > 0$. □

This definition can be analyzed in the particular case where the distribution function is uniform. Assume that $\#(\Omega)$ is the size of the sample space and $\#(A)$ (resp. $\#(B)$ and $\#(A \cap B)$) is the number of elements of A (resp. of B and $A \cap B$). The conditional probability $P(A|B)$ is

$$P(A|B) = \frac{\#(A \cap B)}{\#(B)}. \tag{53}$$

We notice that

$$\frac{\#(B)}{\#(\Omega)} \frac{\#(A \cap B)}{\#(B)} = \frac{\#(A \cap B)}{\#(\Omega)}, \tag{54}$$

for all $A, B \subset \Omega$. This leads to the equality

$$P(B)P(A|B) = P(A \cap B), \tag{55}$$

for all $A, B \subset \Omega$. The previous equation could have been directly found based on the equation 47.

The following example is given in [3], in section 4.1, "Discrete conditional Probability".

Example 1.8 Grinstead and Snell [3] present a table which presents the number of survivors at single years of age. This table gathers data compiled in the USA in

1990. The first line counts 100,000 born alive persons, with decreasing values when the age is increasing. This table allows to see that 89.8 % in a population of 100,000 females can expect to live to age 60, while 57.0 % can expect to live to age 80. Given that a women is 60, what is the probability that she lives to age 80 ?

Let us denote by $A = \{a \geq 60\}$ the event that a woman lives to age 60, and let us denote by $B = \{a \geq 80\}$ the event that a woman lives to age 80. We want to compute the conditionnal probability $P(\{a \geq 80\}|\{a \geq 60\})$. By the proposition 1.11, we have

$$\begin{aligned} P(\{a \geq 80\}|\{a \geq 60\}) &= \frac{P(\{a \geq 60\} \cap \{a \geq 80\})}{P(\{a \geq 60\})} && (56) \\ &= \frac{P(\{a \geq 80\})}{P(\{a \geq 60\})} && (57) \\ &= \frac{0.570}{0.898} && (58) \\ &= 0.635, && (59) \end{aligned}$$

with 3 significant digits. In other words, a women who is already 60, has 63.5 % of chance to live to 80.

2 Combinatorics

In this section, we present several tools which allow to compute probabilities of discrete events. One powerful analysis tool is the tree diagram, which is presented in the first part of this section. Then, we detail permutations and combinations numbers, which allow to solve many probability problems.

2.1 Tree diagrams

In this section, we present the general method which allows to count the total number of ways that a task can be performed. We illustrate that method with tree diagrams.

Assume that a task is carried out in a sequence of n steps. The first step can be performed by making one choice among m_1 possible choices. Similarly, there are m_2 possible ways to perform the second step, and so forth. The total number of ways to perform the complete sequence can be performed in $n = m_1 m_2 \dots m_n$ different ways.

To illustrate the sequence of steps, the associated tree can be drawn. An example of such a tree diagram is given in the figure 6. Each node in the tree corresponds to one step in the sequence. The number of children of a parent node is equal to the number of possible choices for the step. At the bottom of the tree, there are N leafs, where each path, i.e. each sequence of nodes from the root to the leaf, corresponds to a particular sequence of choices.

We can think of the tree as representing a random experiment, where the final state is the outcome of the experiment. In this context, each choice is performed at random, depending on the probability associated with each branch. We will review tree diagrams throughout this section and especially in the section devoted to Bernoulli trials.

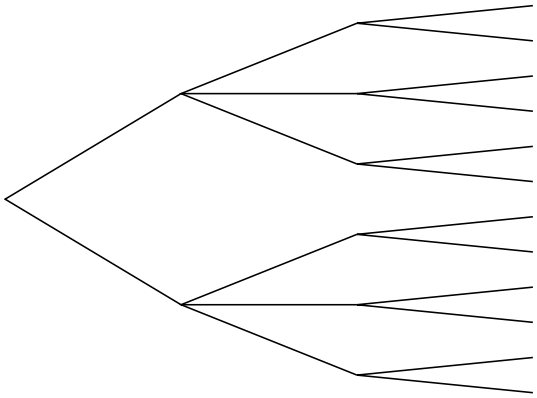

Figure 6: Tree diagram - The task is made with 3 steps. There are 2 choices for the step #1, 3 choices for step #2 and 2 choices for step #3. The total number of ways to perform the full sequence of steps is $n = 2 \cdot 3 \cdot 2 = 12$.

2.2 Permutations

In this section, we present permutations, which are ordered subsets of a given set.

Definition 2.1. *(*Permutation*) Assume that A is a finite set. A permutation of A is a one-to-one mapping of A onto itself.*

Without loss of generality, we can assume that the finite set A can be ordered and numbered from 1 to $n = \#(A)$, so that we can write $A = \{1, 2, \ldots, n\}$. To define a particular permutation, one can write a matrix with 2 rows and n columns which represents the mapping. One example of a permutation on the set $A = \{a_1, a_2, a_3, a_4\}$ is

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \end{pmatrix}, \tag{60}$$

which signifies that the mapping is:

- $a_1 \rightarrow a_2$,
- $a_2 \rightarrow a_1$,
- $a_3 \rightarrow a_4$,
- $a_3 \rightarrow a_4$.

Since the first row is always the same, there is no additional information provided by this row. This is why the permutation can be written by uniquely defining the second row. This way, the previous mapping can be written as

$$\sigma = \begin{pmatrix} 2 & 1 & 4 & 3 \end{pmatrix}. \tag{61}$$

We can try to count the number of possible permutations of a given set A with n elements.

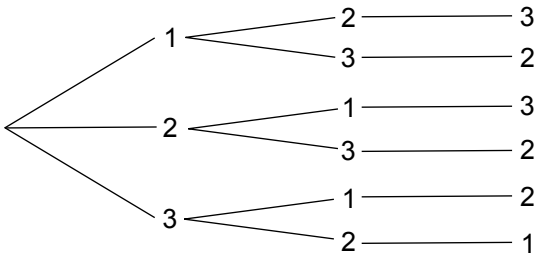

Figure 7: Tree diagram for the computation of permutations of the set $A = \{1, 2, 3\}$.

The tree diagram associated with the computation of the number of permutations for $n = 3$ is presented in figure 7. In the first step, we decide which number to place at index 1. For this index, we have 3 possibilities, that is, the numbers 1, 2 and 3. In the second step, we decide which number to place at index 2. At this index, we have 2 possibilities left, where the exact numbers depend on the branch. In the third step, we decide which number to place at index 3. At this last index, we only have one number left.

This leads to the following proposition, which defines the factorial function.

Proposition 2.2. *(*Factorial*) The number of permutations of a set A of n elements is the factorial of n defined by*

$$n! = n \cdot (n-1) \dots 2 \cdot 1. \tag{62}$$

Proof. #1 Let us pick an element to place at index 1. There are n elements in the set, leading to n possible choices. For the element at index 2, there are $n - 1$ elements left in the set. For the element at index n, there is only 1 element left. The total number of permutations is therefore $n \cdot (n-1) \dots 2 \cdot 1$, which concludes the proof. □

Proof. #2 The element at index 1 can be located at indexes $1, 2, \dots, n$ so that there are n ways to set the element #1. Once the element at index 1 is placed, there are $n - 1$ ways to set the element at index 2. The last element at index n can only be set at the remaining index. The total number of permutations is therefore $n \cdot (n-1) \dots 2 \cdot 1$, which concludes the proof. □

When $n = 0$, it seems that we cannot define the number 0!. For reasons which will be clearer when we will introduce the gamma function, it is convenient to define 0! as equal to one:

$$0! = 1. \tag{63}$$

Example 2.1 Let us compute the number of permutations of the set $A = \{1, 2, 3\}$. By the equation 62, we have $6! = 3 \cdot 2 \cdot 1 = 6$ permutations of the set A. These

permutations are:

$$\begin{array}{ccc} (1 & 2 & 3) \\ (1 & 3 & 2) \\ (2 & 1 & 3) \\ (2 & 3 & 1) \\ (3 & 1 & 2) \\ (3 & 2 & 1) \end{array} \tag{64}$$

The previous permutations can also be directly read from the tree diagram 7, from the root of the tree to each of the 6 leafs.

In some situations, all the elements in the set A are not involved in the permutation. Assume that j is a positive integer, so that $0 \leq j \leq n$. A j-permutation is a permutation of a subset of j elements in A. The general counting method used for the previous proposition allows to count the total number of j-permutations of a given set A.

Proposition 2.3. *(j-permutations) Assume that j is a positive integer. The number of j-permutations of a set A of n elements is*

$$(n)_j = n \cdot (n-1) \dots (n-j+1). \tag{65}$$

Proof. The element at index 1 can be located at indexes $1, 2, \dots, n$ so that there are n ways to set the element at index 1. Once element at index 1 is placed, there are $n-1$ ways to set the element at index 2. The element at index j can only be set at the remaining $n-j+1$ indexes. The total number of j-permutations is therefore $n \cdot (n-1) \dots (n-j+1)$, which concludes the proof. □

Notice that the number of j-permutations of n elements and the factorial of n are equal when $j = n$. Indeed, we have

$$(n)_n = n \cdot (n-1) \dots (n-n+1) = n!. \tag{66}$$

On the other hand, the number of 0-permutations of n elements can be defined to be equal to 1:

$$(n)_0 = 1. \tag{67}$$

Example 2.2 Let us compute the number of 2-permutations of the set $A = \{1, 2, 3, 4\}$. By the equation 65, we have $(4)_2 = 4 \cdot 3 = 12$ permutations of the set A. These permutations are:

$$\begin{array}{cccc} (1 & 2) & (2 & 1) & (3 & 1) & (4 & 1) \\ (1 & 3) & (2 & 3) & (3 & 2) & (4 & 2) \\ (1 & 4) & (2 & 4) & (3 & 4) & (4 & 3) \end{array} \tag{68}$$

We can check that the number of 2-permutations in a set of 4 elements is $(4)_2 = 12$ which is stricly lower that the number of permutations $4! = 24$.

2.3 The gamma function

In this section, we present the gamma function which is closely related to the factorial function. The gamma function was first introduced by the Swiss mathematician Leonard Euler in his goal to generalize the factorial to non integer values[10]. Efficient implementations of the factorial function are based on the gamma function and this is why this functions will be analyzed in detail. The practical computation of the factorial function will be analyzed in the next section.

Definition 2.4. *(*Gamma function*) Let x be a real with $x > 0$. The gamma function is defined by*

$$\Gamma(x) = \int_0^1 (-\log(t))^{x-1} dt. \tag{69}$$

The previous definition is not the usual form of the gamma function, but the following proposition allows to get it.

Proposition 2.5. *(*Gamma function*) Let x be a real with $x > 0$. The gamma function satisfies*

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt. \tag{70}$$

Proof. Let us consider the change of variable $u = -\log(t)$. Therefore, $t = e^{-u}$, which leads, by differenciation, to $dt = -e^{-u}du$. We get $(-\log(t))^{x-1}dt = -u^{x-1}e^{-u}du$. Moreover, if $t = 0$, then $u = \infty$ and if $t = 1$, then $u = 0$. This leads to

$$\Gamma(x) = -\int_\infty^0 u^{x-1} e^{-u} du. \tag{71}$$

For any continuously differentiable function f and any real numbers a and b.

$$\int_a^b f(x)dx = -\int_b^a f(x)dx. \tag{72}$$

We reverse the bounds of the integral in the equality 71 and get the result. □

The gamma function satisfies

$$\Gamma(1) = \int_0^\infty e^{-t} dt = \left[-e^{-t}\right]_0^\infty = (0 + e^0) = 1. \tag{73}$$

The following proposition makes the link between the gamma and the factorial functions.

Proposition 2.6. *(*Gamma and factorial*) Let x be a real with $x > 0$. The gamma function satisfies*

$$\Gamma(x + 1) = x\Gamma(x) \tag{74}$$

and

$$\Gamma(n + 1) = n! \tag{75}$$

for any integer $n \geq 0$.

Proof. Let us prove the equality 74. We want to compute

$$\Gamma(x+1) = \int_0^\infty t^x e^{-t} dt. \tag{76}$$

The proof is based on the integration by parts formula. For any continuously differentiable functions f and g and any real numbers a and b, we have

$$\int_a^b f(t)g'(t)dt = [f(t)g(t)]_a^b - \int_a^b f(t)'g(t)dt. \tag{77}$$

Let us define $f(t) = t^x$ and $g'(t) = e^{-t}$. We have $f'(t) = xt^{x-1}$ and $g(t) = -e^{-t}$. By the integration by parts formula 77, the equation 76 becomes

$$\Gamma(x+1) = \left[-t^x e^{-t}\right]_0^\infty + \int_0^\infty xt^{x-1}e^{-t}dt. \tag{78}$$

Let us introduce the function $h(t) = -t^x e^{-t}$. We have $h(0) = 0$ and $\lim_{t\to\infty} h(t) = 0$, for any $x > 0$. Hence,

$$\Gamma(x+1) = \int_0^\infty xt^{x-1}e^{-t}dt, \tag{79}$$

which proves the equality 74.

The equality 75 can be proved by induction on n. First, we already noticed that $\Gamma(1) = 1$. If we define $0! = 1$, we have $\Gamma(1) = 0!$, which proves the equality 75 for $n = 0$. Then, assume that the equality holds for n and let us prove that $\Gamma(n+2) = (n+1)!$. By the equality 74, we have $\Gamma(n+2) = (n+1)\Gamma(n+1) = (n+1)n! = (n+1)!$, which concludes the proof. □

The gamma function is not the only function f which satisfies $f(n) = n!$. But the Bohr-Mollerup theorem prooves that the gamma function is the unique function f which satisfies the equalities $f(1) = 1$ and $f(x+1) = xf(x)$, and such that $log(f(x))$ is convex [2].

It is possible to extend this function to negative values by inverting the equation 74, which implies

$$\Gamma(x) = \frac{\Gamma(x+1)}{x}, \tag{80}$$

for $x \in]-1, 0[$. This allows to compute, for example $\Gamma(-1/2) = -2\Gamma(1/2)$. By induction, we can also compute the value of the gamma function for $x \in]-2, -1[$. Indeed, the equation 80 implies

$$\Gamma(x+1) = \frac{\Gamma(x+2)}{x+1}, \tag{81}$$

which leads to

$$\Gamma(x) = \frac{\Gamma(x+2)}{x(x+1)}. \tag{82}$$

By induction of the intervals $]-n-1, -n[$ with n a positive integer, this formula allows to compute values of the gamma function for all $x \leq 0$, except the negative integers $0, -1, -2, \ldots$. This leads to the following proposition.

Proposition 2.7. *(*Gamma function for negative arguments*) For any non zero integer n and any real x such that $x + n > 0$,*

$$\Gamma(x) = \frac{\Gamma(x+n)}{x(x+1)\dots(x+n-1)}. \tag{83}$$

Proof. The proof is by induction on n. The equation 80 prooves that the equality is true for $n = 1$. Assume that the equality 83 is true for n et let us proove that it also holds for $n + 1$. By the equation 80 applied to $x + n$, we have

$$\Gamma(x+n) = \frac{\Gamma(x+n+1)}{x+n}. \tag{84}$$

Therefore, we have

$$\Gamma(x) = \frac{\Gamma(x+n+1)}{x(x+1)\dots(x+n-1)(x+n)} \tag{85}$$

which proves that the statement holds for $n + 1$ and concludes the proof. □

The gamma function is singular for negative integers values of its argument, as stated in the following proposition.

Proposition 2.8. *(*Gamma function for integer negative arguments*) For any non negative integer n,*

$$\Gamma(-n+h) \sim \frac{(-1)^n}{n!h}, \tag{86}$$

when h is small.

Proof. Consider the equation 83 with $x = -n + h$. We have

$$\Gamma(-n+h) = \frac{\Gamma(h)}{(h-n)(h-n+1))\dots(h+1)}. \tag{87}$$

But $\Gamma(h) = \frac{\Gamma(h+1)}{h}$, which leads to

$$\Gamma(-n+h) = \frac{\Gamma(h+1)}{(h-n)(h-n+1))\dots(h+1)h}. \tag{88}$$

When h is small, the expression $\Gamma(h+1)$ converges to $\Gamma(1) = 1$. On the other hand, the expression $(h-n)(h-n+1))\dots(h+1)h$ converges to $(-n)(-n+1)\dots(1)h$, which leads to the the term $(-1)^n$ and concludes the proof. □

We have reviewed the main properties of the gamma function. In practical situations, we use the gamma function in order to compute the factorial number, as we are going to see in the next sections. The main advantage of the gamma function over the factorial is that it avoids to form the product $n! = n \cdot (n-1)\dots 1$, which allows to save a significant amount of CPU time and computer memory.

factorial	returns $n!$
gamma	returns $\Gamma(x)$
gammaln	returns $\ln(\Gamma(x))$

Figure 8: Scilab commands for permutations.

2.4 Overview of functions in Scilab

The figure 8 presents the functions provided by Scilab to compute permutations.

Notice that there is no function to compute the number of permutations $(n)_j = n \cdot (n-1) \dots (n-j+1)$. This is why, in the next sections, we provide a Scilab function to compute $(n)_j$.

In the next sections, we analyze each function in Scilab. We especially consider their numerical behavior and provide accurate and efficient Scilab functions to manage permutations. We emphasize the need for accuracy and robustness. For this purpose, we use the logarithmic scale to provide intermediate results which stays in the limited bounds of double precision floating point arithmetic.

2.5 The gamma function in Scilab

The `gamma` function allows to compute $\Gamma(x)$ for real input argument. The mathematical function $\Gamma(x)$ can be extended to complex arguments, but this has not be implemented in Scilab.

The following script allows to plot the `gamma` function for $x \in [-4, 4]$.

```
x = linspace ( -4 , 4 , 1001 );
y = gamma ( x );
plot ( x , y );
h = gcf();
h.children.data_bounds = [
  - 4.  -6
    4.   6
];
```

The previous script produces the figure 9.

The following session presents various values of the `gamma` function.

```
-->x = [-2 -1 -0 +0 1 2 3 4 5 6]';
-->[x gamma(x)]
 ans  =
  - 2.   Nan
  - 1.   Nan
    0.  -Inf
    0.   Inf
    1.    1.
    2.    1.
    3.    2.
    4.    6.
    5.    24.
    6.    120.
```

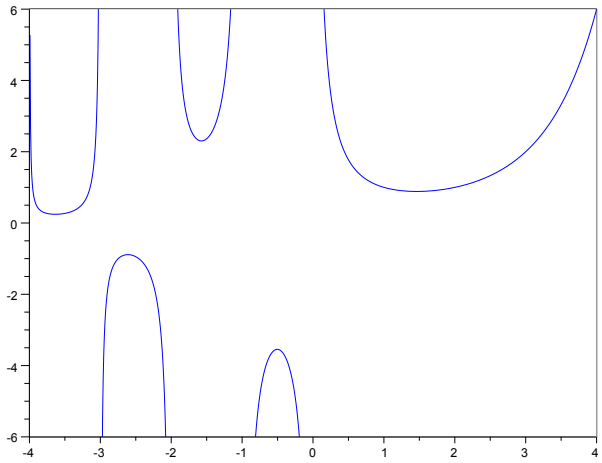

Figure 9: The gamma function.

Notice that the two floating point signed zeros `+0` and `-0` are associated with the function values $-\infty$ and $+\infty$. This is consistent with the value of the limit of the function from either sides of the singular point. This contrasts with the value of the gamma function on negative integer points, where the function value is `%nan`. This is consistent with the fact that, on this singular points, the function is equal to $-\infty$ on one side and $+\infty$ on the other side. Therefore, since the argument x has one single floating point representation when it is a negative nonzero integer, the only solution consistent with the IEEE754 standard is to set the result to `%nan`.

Notice that we used 1001 points to plot the gamma function. This allows to get points exactly located at the singular points. These values are ignored by the `plot` function and makes a nice plot. Indeed, if 1000 points are used instead, vertical lines corresponding to the y-value immediately at the left and the right of the singularity would be displayed.

2.6 The factorial and log-factorial functions

In the following script, we plot the `factorial` function for values of n from 1 to 10.

```
f = factorial(1:10)
plot ( 1:10 , f , "b-o" )
```

The result is presented in figure 10. We see that the growth rate of the factorial function is large.

The largest values of n so that $n!$ is representable as a double precision floating point number is $n = 170$. In the following session, we check that 171! is not representable as a Scilab double.

```
-->factorial(170)
 ans  =
    7.257+306
```

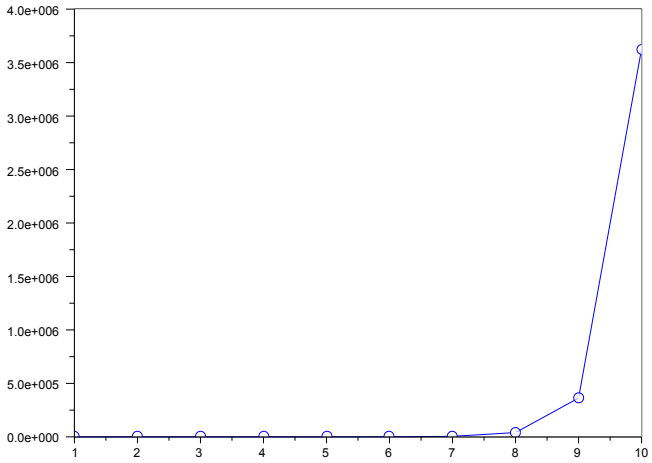

Figure 10: The factorial function.

```
-->factorial(171)
 ans  =
   Inf
```

The factorial function is implied in many probability computations, sometimes as an intermediate result. Since it grows so fast, we might be interested in computing its order of magnitude instead of its value. Let us introduce the function f_{ln} as the logarithm of the factorial number $n!$:

$$f_{ln}(n) = \ln(n!). \tag{89}$$

Notice that we used the base-e logarithm function ln, that is, the reciprocal of the exponential function.

The factorial number $n!$ grows exponentially, but its logarithm grows much more slowly. In the figure 11, we plot the logarithm of $n!$ in the interval $[0, 170]$. We see that the y coordinate varies only from 0 up to 800. Hence, there are a large number of integers n for which $n!$ may be not representable as a double but $f_{ln}(n)$ is still representable as a double.

2.7 Computing factorial and log-factorial with Scilab

In this section, we present how to compute the factorial function in Scilab. We focus in this section on accuracy and efficiency.

The `factorial` function returns the factorial number associated with the given n. It has the following syntax:

```
f = factorial ( n )
```

In the following session, we compute $n!$ for several values of n, from 1 to 7.

```
-->n = (0:7)';
```

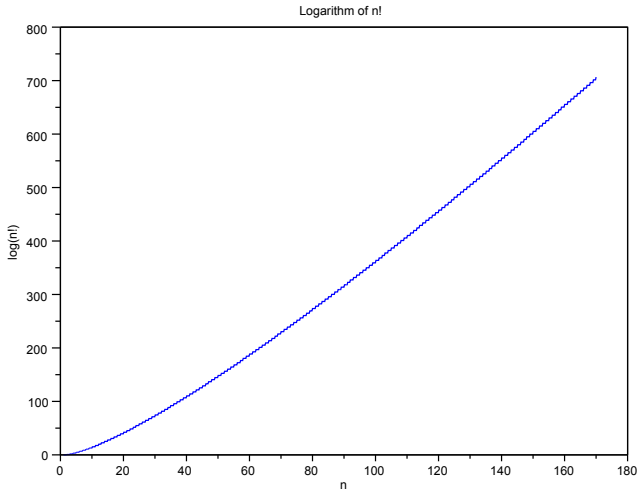

Figure 11: Logarithm of the factorial number.

```
-->[n factorial(n)]
 ans  =
    0.    1.
    1.    1.
    2.    2.
    3.    6.
    4.    24.
    5.    120.
    6.    720.
    7.    5040.
```

The implementation of the `factorial` function in Scilab allows to take both matrix and hypermatrices input arguments. In order to be fast, it uses vectorization. The following `factorialScilab` function represents the computationnal core of the actual implementation of the `factorial` function in Scilab.

```
function f = factorialScilab ( n )
  n(n==0)=1
  t = cumprod(1:max(n))
  v = t(n(:))
  f = matrix(v,size(n))
endfunction
```

The statement `n(n==0)=1` allows to set all zeros of the matrix `n` to one, so that the next statements do not have to manage the special case $0! = 1$. Then, we use the `cumprod` function in order to compute a column vector containing cumulated products, up to the maximum entry of `n`. The use of `cumprod` allows to get all the results in one call, but also produces unnecessary values of the factorial function. In order to get just what is needed, the statement `v = t(n(:))` allows to extract the required values. Finally, the statement `f = matrix(v,size(n))` reshapes the matrix of values so that the shape of the output argument is the same as the shape of the input argument.

The following function allows to compute $n!$ based on the `prod` function, which computes the product of its input argument.

```
function f = factorial_naive ( n )
  f = prod(1:n)
endfunction
```

The `factorial_naive` function has two drawbacks. The first one is that it cannot manage matrix input arguments. Furthermore, it requires more memory than necessary.

In practice, the factorial function can be computed based on the gamma function. The following implementation of the factorial function is based on the equality 75.

```
function f = myfactorial ( n )
  if ( ( or (n(:) <  0) ) | ( n(:) <> round (n(:) ) ) ) then
    error ("myfactorial: n must all be nonnegative integers");
  end
  f = gamma ( n + 1 )
endfunction
```

The `myfactorial` function also checks that the input argument `n` is positive. It also checks that `n` is an integer by using the condition `( n(:) <> round (n(:) )`. Indeed, if the value of `n` is different from the value of `round(n)`, this means that the input argument `n` is not an integer.

The main drawback of the `factorialScilab` function is that it uses more memory than necessary. It may fail to produce a result when it is given a large input argument. In the following session, we use the `factorial` function with a very large input integer. In this particular case, it is obvious that the correct result is `Inf`. Anyway, this should have been the result of the function, which should not have generated an error. On the other side, the `myfactorial` function works perfectly.

```
-->factorial(1.e10)
 !--error 17
stack size exceeded!
-->myfactorial(1.e10)
 ans  =
   Inf
```

We now consider the computation of the log-factorial function f_{ln}. We can use the `gammaln` function which directly provides the correct result.

```
function flog = factoriallog ( n )
  flog = gammaln(n+1);
endfunction
```

The advantage of this method is that matrix input arguments can be manage by the `factoriallog` function.

2.8 Computing permutations and log-permutations with Scilab

There is no Scilab function to compute the number of permutations $(n)_j = n.(n-1)\dots(n-j+1)$.

We might be interested in simplifying the expression for the permutation number We have

$$
\begin{aligned}
(n)_j &= n.(n-1)\dots(n-j+1) && (90)\\
&= \frac{n.(n-1).\dots.1}{(n-j).(n-j-1)\dots 1}, && (91)\\
&= \frac{n!}{(n-j)!}. && (92)
\end{aligned}
$$

This leads to the following function `permutations_verynaive`.

```
function p = permutations_verynaive ( n , j )
  p = factorial(n)./factorial(n-j)
endfunction
```

In the following session, we see that the previous function works for small values of n and j.

```
-->n = [5 5 5 5 5 5]';
-->j = [0 1 2 3 4 5]';
-->[n j permutations_verynaive(n,j)]
 ans  =
    5.    0.    1.
    5.    1.    5.
    5.    2.    20.
    5.    3.    60.
    5.    4.    120.
    5.    5.    120.
```

In the following session, we compute the permutations number $(171)_{171} = 171!$.

```
-->permutations_verynaive ( 171 , 171 )
 ans  =
   Inf
```

This is caused by an overflow during the computation of the factorial function. There is unfortunately no way to fix this problem, since the result is, indeed, not representable as a double precision floating point number.

On the other hand, the `permutations_verynaive` function performs poorly in cases where n is large, whatever the value of j, as presented in the following session.

```
-->permutations_verynaive ( 171 , 0 )
 ans  =
   Nan
```

There is certainly something to do about this problem, since, when $j = 0$, we have $(n)_0 = 1$.

The following `permutations_naive` function allows to compute $(n)_j$ for positive integer values of n, j. It is based on the `prod` function, which computes the product of the given vector.

```
function p = permutations_naive ( n , j )
  p = prod ( n-j+1 : n )
endfunction
```

In the following session, we check the values of the function $(n)_j$ for $n = 5$ and $j = 1, 2, \dots, 5$.

```
-->n = 5;
-->for j = 0 : 5
-->  p = permutations_naive ( n , j );
-->  disp([n j p]);
-->end
    5.    0.    1.
    5.    1.    5.
    5.    2.    20.
    5.    3.    60.
    5.    4.    120.
    5.    5.    120.
```

The following session shows that `permutations_naive` performs more nicely than the previous function for small values of j.

```
-->permutations_naive ( 171 , 0 )
 ans  =
    1.
```

The `permutations_naive` function has still several drawbacks. First, it requires more memory than necessary. For example, it may fail to compute $(n)_n = 1$ for values of n larger than 10^5.

```
-->permutations_naive ( 1.e7 , 1.e7 )
 !--error 17
stack size exceeded!
```

Furthermore, the function `permutations_naive` does not manage matrix input arguments.

In order to accurately compute the permutation number, we may compute its logarithm first. By the equation 92, we have

$$\begin{aligned} \ln((n)_j) &= \ln\left(\frac{n!}{(n-j)!}\right) && (93)\\ &= \ln(n!) - \ln((n-j)!) && (94)\\ &= \ln(\Gamma(n+1)) - \ln(\Gamma(n-j+1)). && (95) \end{aligned}$$

The previous equation leads to the definition of the log-permutation function, as defined in the following function.

```
function plog = permutationslog ( n , j )
  plog = gammaln(n+1)-gammaln(n-j+1);
endfunction
```

In order to compute the permutation number, we compute the exponential of the expression. This leads to the following function `permutations`, where we round the result in order to get integer results.

```
function p = permutations ( n , j )
  p = exp(gammaln(n+1)-gammaln(n-j+1));
  if ( and(round(n)==n) & and(round(j)==j) ) then
    p = round ( p )
  end
endfunction
```

The `permutations` function takes matrix input arguments, as presented in the following session.

```
-->n = [5 5 5 5 5 5]';
-->j = [0 1 2 3 4 5]';
-->[n j permutations(n,j)]
 ans  =
    5.    0.    1.
    5.    1.    5.
    5.    2.    20.
    5.    3.    60.
    5.    4.    120.
    5.    5.    120.
```

Finally, the `permutations` functions requires the minimum amount of memory and performs correctly, even for large values of n.

```
-->permutations ( 1.e7 , 1.e7 )
 ans  =
   Inf
-->permutations ( 1.e7 , 0 )
 ans  =
    1.
```

2.9 Combinations

In this section, we present combinations, which are unordered subsets of a given set.

The number of distinct subsets with j elements which can be chosen from a set A with n elements is the binomial coefficient and is denoted by $\binom{n}{j}$. The following proposition gives an explicit formula for the binomial number.

Proposition 2.9. *(*Binomial*) The number of distinct subsets with j elements which can be chosen from a set A with n elements is the binomial coefficient and is defined by*

$$\binom{n}{j} = \frac{n.(n-1)\dots(n-j+1)}{1.2\dots j}. \tag{96}$$

The following proof is based on the fact that subsets are unordered, while permutations are based on the order.

Proof. Assume that the set A has n elements and consider subsets with $j > 0$ elements. By proposition 2.3, the number of j-permutations of the set A is $(n)_j = n.(n-1)\dots(n-j+1)$. Notice that the order does not matter in creating the subsets, so that the number of subsets is lower than the number of permutations. This is why each subset is associated with one or more permutations. By proposition 2.2, there are $j!$ ways to order a set with j elements. Therefore, the number of subsets with j elements is given by $\binom{n}{j} = \frac{n.(n-1).....(n-j+1)}{1.2...j}$, which concludes the proof. □

The expression for the binomial coefficient can be simplified if we use the number of j-permutations and the factorial number, which leads to

$$\binom{n}{j} = \frac{(n)_j}{j!}. \tag{97}$$

The equality $(n)_j = \frac{n!}{(n-j)!}$ leads to

$$\binom{n}{j} = \frac{n!}{(n-j)!j!}. \tag{98}$$

This immediately leads to

$$\binom{n}{j} = \binom{n}{n-j}. \tag{99}$$

The following proposition shows a recurrence relation for binomial coefficients.

Proposition 2.10. *For integers $n > 0$ and $0 < j < n$, the binomial coefficients satisfy*

$$\binom{n}{j} = \binom{n-1}{j} + \binom{n-1}{j-1}. \tag{100}$$

The proof recurrence relation of the proposition 2.10 is given as an exercise.

2.10 Computing combinations with Scilab

In this section, we show how to compute combinations with Scilab.

There is no Scilab function to compute the binomial number $\binom{n}{j}$. In order to compute the required combinations, we will use the gamma function. By the equation 98, we have

$$\begin{aligned} \ln\left(\binom{n}{j}\right) &= \ln(n!) - \ln((n-j)!) - \ln(j!) && (101)\\ &= \ln(\Gamma(n+1)) - \ln(\Gamma(n-j+1)) - \ln(\Gamma(j+1)). && (102) \end{aligned}$$

The following Scilab function performs the computation of the binomial number for positive values of n and j.

```
function c = combinations ( n , j )
  c = exp(gammaln(n+1)-gammaln(j+1)-gammaln(n-j+1));
  if ( and(round(n)==n) & and(round(j)==j) ) then
    b = round ( b )
  end
endfunction
```

In the following session, we compute the value of the binomial coefficients for $n = 1, 2, \ldots, 5$. The values in this table are known as Pascal's triangle.

```
-->for n=0:5
-->  for j=0:n
-->    c = combinations ( n , j );
-->    mprintf("%2d   ",c);
-->  end
-->  mprintf("\n");
-->end
 1
```

```
 1.    1.
 1.    2.    1.
 1.    3.    3.    1.
 1.    4.    6.    4.    1.
 1.    5.   10.   10.    5.    1.
```

We now explain why we choose to use the `exp` and the `gammaln` to perform our computation for the `combinations` function. Indeed, we could have used a more naive method, based on the `prod` function, as in the following example :

```
function c = combinations_naive ( n , j )
  c = prod ( n : -1 : n-j+1 )/prod(1:j)
endfunction
```

For small integer values of n, the two previous functions produce the same result. Unfortunately, even for moderate values of n, the naive method fails. In the following session, we compute the value of $\binom{n}{j}$ with $n = 10000$ and $j = 134$.

```
-->combinations ( 10000 , 134 )
 ans  =
    2.050+307
-->combinations_naive ( 10000 , 134 )
 ans  =
   Inf
```

The reason why the naive computation fails is because the products involved in the intermediate variables for the naive method are generating an overflow. This means that the values are too large for being stored in a double precision floating point variable. This is a pity, since the result can be stored in a double precision floating point variable. The `combinations` function, on the other hand, computes first the logarithm of the factorial number. This logarithm cannot overflow because if `x` is a double precision floating point number, then `log(x)` can always be represented since its exponent is always smaller than the exponent of `x`. In the end, the combination of the `exp` and `gammaln` functions allows to accurately compute the result in the sense that, if the result is representable as a double precision floating point number, then `combinations` will produce a result as accurate as possible.

Notice that we use the `round` function in our implementation of the `combinations` function. This is because the `combinations` function manages in fact real double precision floating point input arguments. Consider the example where $n = 4$ and $j = 1$ and let us compute the associated number of combinations $\binom{n}{j}$. In the following Scilab session, we use the `format` so that we display at least 15 significant digits.

```
-->format(20);
-->n = 4;
-->j = 1;
-->c = exp(gammaln(n+1)-gammaln(j+1)-gammaln(n-j+1))
 c  =
    3.99999999999999822
```

We see that there are 15 significant digits, which is the best that can be expected from the `exp` and `gammaln` functions. But the result is not an integer anymore, i.e.

1□	2□	3□	4□	5□	6□	7□	8□	9□	10□	J□	Q□	K□
1♡	2♡	3♡	4♡	5♡	6♡	7♡	8♡	9♡	10♡	J♡	Q♡	K♡
1♣	2♣	3♣	4♣	5♣	6♣	7♣	8♣	9♣	10♣	J♣	Q♣	K♣
1♠	2♠	3♠	4♠	5♠	6♠	7♠	8♠	9♠	10♠	J♠	Q♠	K♠

Figure 12: Cards of a 52 cards deck - "J" stands for Jack, "Q" stands for Queen and "K" stands for King

it is very close to the integer 4, but not exactly equal to it. This is why in the `combinations` function, if n and j are both integers, we round the number c to the nearest integer with a call to the `round` function.

Finally, notice that our implementation of the `combinations` function uses the function `and`. This allows to use arrays of integers as input variables. In the following session, we compute $\binom{5}{j}$, for $j = 0, 1, \ldots, 5$ in one single call. This is a consequence of the fact that the `exp` and `gammaln` both accept matrices input arguments.

```
-->n = 5 * ones(6,1);
-->j = (0:5)';
-->c = combinations ( n , j );
-->[n j c]
 ans  =
    5.    0.    1.
    5.    1.    5.
    5.    2.    10.
    5.    3.    10.
    5.    4.    5.
    5.    5.    1.
```

2.11 The poker game

In the following example, we use Scilab to compute the probabilities of poker hands.

The poker game is based on a 52 cards deck, which is presented in figure 12. Each card can have one of the 13 available *ranks* from 1 to K, and have on the 4 available *suits* □, ♡, ♣ and ♠. Each player receives 5 cards randomly chosen in the deck. Each player tries to combine the cards to form a well-known combination of cards as presented in the figure 13. Depending on the combination, the player can beat, or be defeated by another player. The winning combination is the rarest; that is, the one which has the lowest probability. In figure 13, the combinations are presented in decreasing order of probability.

Even if winning at this game requires some understanding of human psychology, understanding probabilities can help. Why does the *four of a kind* beats the *full house* ?

To answer this question, we will compute the probability of each event. Since the order of the cards can be changed by the player, we are interested in combinations (and not in permutations). We make the assumption that the process of choosing the cards is really random, so that all combinations of 5 cards have the same probabilities, i.e. the distribution function is uniform. Since the order of the cards does

Name	Description	Example
no pair	none of the below combinations	7□ 3♣ 6♣ 3♡ 1♠
pair	two cards of the same rank	Q♣ Q♣ 2♡ 3♣ 1□
double pair	2× two cards of the same rank	2□ 2♠ Q♠ Q♣
three of a kind	three cards of the same rank	2□ 2♠ 2♡ 3♣ 1□
straight	five cards in a sequence, not all the same suit	3♡ 4♠ 5□ 6♠ 7□
flush	five cards in a single suit	2□ 3□ 7□ J□ K□
full house	one pair and one triple, each of the same rank	2□ 2♠ 2♡ Q♣ Q♡
four of a kind	four cards of the same rank	5□ 5♠ 5♣ 5♡ 2♡
straight flush	five in a sequence in a single suit	2□ 3□ 4□ 5□ 6□
royal flush	10, J, Q, K, 1 in a single suit	10♠ J♠ Q♠ K♠ 1♠

Figure 13: Winning combinations at the Poker

not matter, the sample space Ω is the set of all combinations of 5 cards chosen from 52 cards. Therefore, the size of Ω is

$$\#\Omega = \binom{52}{5} = 2598960. \tag{103}$$

The probability of a *four of a kind* is computed as follows. In a 52-cards deck, there are 13 different *four of a kind* combinations. Since the 5-th card is chosen at random from the 48 remaining cards, there are 13.48 different *four of a kind*. The probability of a *four of a kind* is therefore

$$P(\text{four of a kind}) = \frac{13.48}{\binom{52}{5}.} = \frac{624}{2598960} \approx 0.0002401 \tag{104}$$

The probability of a *full house* is computed as follows. There are 13 different ranks in the deck, and, once a rank is chosen, there are $\binom{4}{2}$ different pairs for one rank Therefore the total number of pairs is 13. $\binom{4}{2}$. Once the pair is set, there are 12 different ranks to choose for the triple and there are $\binom{4}{3}$ different triples for one rank. The total number of *full house* is therefore 13. $\binom{4}{2}$.12. $\binom{4}{3}$. Notice that the triple can be chosen first, and the pair second, but this would lead exactly to the same result. Therefore, the probability of a *full house* is

$$P(\text{full house}) = \frac{13.12.\binom{4}{2}.\binom{4}{3}}{\binom{52}{5}.} = \frac{3744}{2598960} \approx 0.0014406 \tag{105}$$

The computation of all the probabilities of the winning combinations is given as exercise.

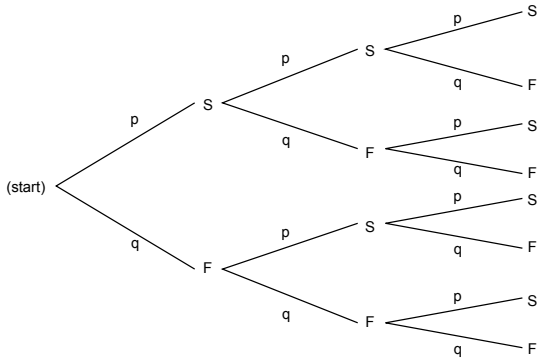

Figure 14: A Bernoulli process with 3 trials. – The letter "S" indicates "success" and the letter "F" indicates "failure".

2.12 Bernoulli trials

In this section, we present Bernoulli trials and the binomial discrete distribution function. We give the example of a coin tossed several times as an example of such a process.

Definition 2.11. *A Bernoulli trials process is a sequence of $n > 0$ experiments with the following rules.*

1. *Each experiment has two possible outcomes, which we may call success and failure.*
2. *The probability $p \in [0, 1]$ of success of each experiment is the same for each experiment.*

In a Bernoulli process, the probability p of success is not changed by any knowledge of previous outcomes For each experiment, the probability q of failure is $q = 1 - p$.

It is possible to represent a Bernoulli process with a tree diagram, as the one in figure 14.

A complete experiment is a sequence of success and failures, which can be represented by a sequence of S's and F's. Therefore the size of the sample space is $\#(\Omega) = 2^n$, which is equal to $2^3 = 8$ in our particular case of 3 trials.

By definition, the result of each trial is independent from the result of the previous trials. Therefore, the probability of an event is the product of the probabilities of each outcome.

Consider the outcome $x = "SFS"$ for example. The value of the distribution function f for this outcome is

$$f(x = "SFS") = pqp = p^2q. \tag{106}$$

The table 15 presents the value of the distribution function for each outcome $x \in \Omega$.

x	$f(x)$
SSS	p^3
SSF	p^2q
SFS	p^2q
SFF	pq^2
FSS	p^2q
FSF	pq^2
FFS	pq^2
FFF	q^3

Figure 15: Probabilities of a Bernoulli process with 3 trials

We can check that the sum of probabilities of all events is equal to 1. Indeed,

$$\begin{aligned}
\sum_{i=1,8} f(x_i) &= p^3 + p^2q + p^2q + pq^2 + p^2q + pq^2 + pq^2 + q^3 && (107)\\
&= p^3 + 3p^2q + 3pq^2 + q^3 && (108)\\
&= (p+q)^3 && (109)\\
&= 1 && (110)
\end{aligned}$$

We denote by $b(n, p, j)$ the probability that, in n Bernoulli trials with success probability p, there are exactly j successes. In the particular case where there are $n = 3$ trials, the figures 14 and 15 gives the following results:

$$\begin{aligned}
b(3, p, 3) &= p^3 && (111)\\
b(3, p, 2) &= 3p^2p && (112)\\
b(3, p, 1) &= 3pq^2 && (113)\\
b(3, p, 0) &= q^3 && (114)
\end{aligned}$$

The following proposition extends the previous analysis to the general case.

Proposition 2.12. *(*Binomial probability*) In a Bernoulli process with $n > 0$ trials with success probability $p \in [0, 1]$, the probability of exactly j successes is*

$$b(n, p, j) = \binom{n}{j} p^j q^{n-j}, \tag{115}$$

where $0 \leq j \leq n$ and $q = 1 - p$.

Proof. We denote by $A \subset \Omega$ the event that one process is associated with exactly j successes. By definition, the probability of the event A is

$$b(n, p, j) = P(A) = \sum_{x \in A} f(x). \tag{116}$$

Assume that an outcome $x \in \Omega$ is associated with exactly j successes. Since there are n trials, the number of failures is $n - j$. That means that the value of the

distribution function of this outcome $x \in A$ is $f(x) = p^j q^{n-j}$. Since all the outcomes x in the set A have the same distribution function value, we have

$$b(n, p, j) = \#(A) p^j q^{n-j}. \tag{117}$$

The size of the set A is the number of subsets of j elements in a set of size n. Indeed, the order does not matter since we only require that, during the whole process, the total number of successes is exactly j, no matter of the order of the successes and failures. The number of outcomes with exactly j successes is therefore $\#(A) = \binom{n}{j}$, which, combined with equation 117, concludes the proof. □

Example 2.3 A fair coin is tossed six times. What is the probability that exactly 3 heads turn up ? This process is a Bernoulli process with $n = 6$ trials. Since the coin is fair, the probability of success at each trial is $p = 1/2$. We can apply proposition 2.12 with $j = 3$ and get

$$b(6, 1/2, 3) = \binom{6}{3} (1/2)^3 (1/2)^3 \approx 0.3125, \tag{118}$$

so that the probability of having exactly 3 heads is 0.3125.

3 Simulation of random processes with Scilab

In this section, we present how to simulate random events with Scilab. The problem of generating random numbers is more complex and will not be detailed in this chapter. We begin by a brief overview of random number generation and detail the random number generator used in the `rand` function. Then we analyze how to generate random numbers in the interval $[0, 1]$ with the `rand` function. We present how to generate random integers in a given interval $[0, m-1]$ or $[m_1, m_2]$. In the final part, we present a practical simulation of a game based on tossing a coin.

3.1 Overview

In this section, we present a special class of random number generators so that we can have a general representation of what exactly this means.

The goal of a uniform random number generator is to generate a sequence of real values $u_n \in [0, 1]$ for $n \geq 0$. Most uniform random number generators are based on the fraction

$$u_n = \frac{x_n}{m} \tag{119}$$

where m is a large integer and x_n is a positive integer so that $0 < x_n < m$. In many random number generators, the integer x_{n+1} is computed from the previous element in the sequence x_n.

The linear congruential generators [5] are based on the sequence

$$x_{n+1} = (a x_n + c) \pmod{m}, \tag{120}$$

where

- m is the modulus satisfying $m > 0$,
- a is the multiplier satisfying $0 \leq a < m$,
- c is the increment satisfying $0 \leq c < m$,
- x_0 is the starting value satisfying $0 \leq x_0 < m$.

The parameters m, a, c and x_0 should satisfy several conditions so that the sequence x_n has good statistical properties. Indeed, naive approaches leads to poor results in this matter. For example, consider the example where $x_0 = 0$, $a = c = 7$ and $m = 10$. The following sequence of number is produced :

$$0.6 \quad 0.9 \quad 0. \quad 0.7 \quad 0.6 \quad 0.9 \quad 0. \quad 0.7 \ldots \tag{121}$$

Specific rules allow to design the parameters of a uniform random number generator.

As a practical example, consider the Urand generator [7] which is used by Scilab in the `rand` function. Its parameters are

- $m = 2^{31}$,
- $a = 843314861$,
- $c = 453816693$,
- x_0 arbitrary.

The first 8 elements of the sequence are

$$0.2113249 \quad 0.7560439 \quad 0.0002211 \quad 0.3303271 \tag{122}$$

$$0.6653811 \quad 0.6283918 \quad 0.8497452 \quad 0.6857310 \ldots \tag{123}$$

3.2 Generating uniform random numbers

In this section, we present some Scilab features which allow to simulate a discrete random process.

We assume here that a good source of random numbers is provided.

Scilab provides two functions which allow to generate uniform real numbers in the interval $[0, 1]$. These functions are `rand` and `grand`. Globally, the `grand` function provides much more features than `rand`. The random number generators which are used are also of higher quality. For our purpose of presenting the simulation of random discrete events, the `rand` will be sufficient and have the additional advantage of having a simpler syntax.

The simplest syntax of the `rand` function is

```
rand()
```

Each call to the `rand` function produces a new random number in the interval $[0, 1]$, as presented in the following session.

```
-->rand()
 ans  =
    0.2113249
-->rand()
 ans  =
    0.7560439
-->rand()
 ans  =
    0.0002211
```

A random number generator is based on a sequence of integers, where the first element in the sequence is the *seed*. The seed for the generator used in the `rand` is hard-coded in the library as being equal to 0, and this is why the function always returns the same sequence of Scilab. This allows to reproduce the behavior of a script which uses the `rand` function more easily.

The seed can be queried with the function `rand("seed")`, while the function `rand("seed",s)` sets the seed to the value `s`. The use of the "seed" input argument is presented in the following session.

```
-->rand("seed")
 ans  =
    0.
-->rand("seed",1)
-->rand()
 ans  =
    0.6040239
-->rand()
 ans  =
    0.0079647
```

In most random processes, several random numbers are to use at the same time. Fortunately, the `rand` function allows to generate a matrix of random numbers, instead of a single value. The user must then provide the number of rows and columns of the matrix to generate, as in the following syntax.

```
rand(nr,nc)
```

The use of this feature is presented in the following session, where a 2×3 matrix of random numbers is generated.

```
-->rand(2,3)
 ans  =
    0.6643966    0.5321420    0.5036204
    0.9832111    0.4138784    0.6850569
```

3.3 Simulating random discrete events

In this section, we present how to use a uniform random number generator to generate integers in a given interval. Assume that, given a positive integer m, we want to generate random integers in the interval $[0, m-1]$. To do this, we can use the `rand` function, and multiply the generated numbers by m. We must additionally use the `floor` function, which returns the largest integer smaller than the given number. The following function returns a matrix with size `nr`×`nc`, where entries are random integers in the set $\{0, 1, \ldots, m-1\}$.

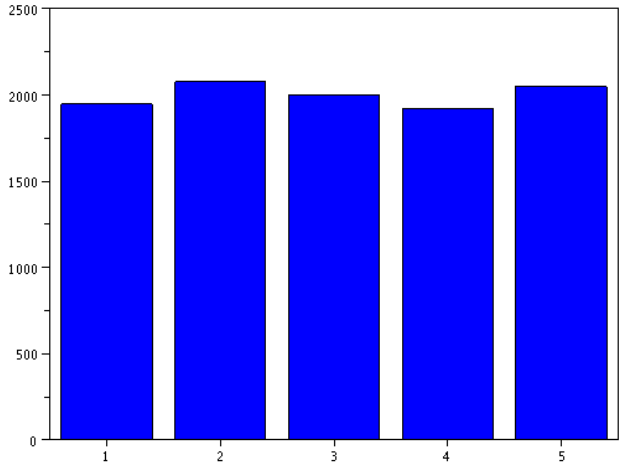

Figure 16: Distribution of random integers from 0 to 4.

```
function ri = generateInRange0M1 ( m , nbrows , nbcols )
  ri = floor(rand( nbrows , nbcols ) * m)
endfunction
```

In the following session, we generate random integers in the set $\{0, 1, \ldots, 4\}$.

```
-->r = generateInRange0M1 ( 5 , 4 , 4 )
 r  =
    2.    0.    3.    0.
    1.    0.    2.    2.
    0.    1.    1.    4.
    4.    2.    4.    0.
```

To check that the generated integers are uniform in the interval, we compute the distribution of the integers for 10000 integers in the set $\{0, 1, \ldots, 4\}$. We use the `bar` to plot the result, which is presented in the figure 16. We check that the probability of each integer is close to $\frac{1}{5} = 0.2$.

```
-->r = generateInRange0M1 ( 5 , 100 , 100 );

-->counter = zeros(1,5);
-->for i = 1:100
-->for j = 1:100
-->  k = r(i,j);
-->  counter(k+1) = counter(k+1) + 1;
-->end
-->end
-->counter = counter / 10000;
-->counter
 counter  =
    0.2023    0.2013    0.1983    0.1976    0.2005
-->bar(counter)
```

We emphasize that the previous verifications allow to check that the empirical distribution function is the expected one, but that does not guarantee that the uniform random number generator is of good quality. Indeed, consider the sequence $x_n = n \pmod 5$. This sequence produces uniform integers in the set $\{0, 1, \ldots, 4\}$, but, obviously, is far from being truly random. Testing uniform random number generators is a much more complicated problem and will not be presented here.

3.4 Simulation of a coin

Many practical experiments are very difficult to analyze by theory and, most of the time, very easy to experiment with a computer. In this section, we give an example of a coin experiment which is simulated with Scilab. This experiment is simple, so that we can check that our simulation matches the result predicted by theory. In practice, when no theory is able to predict a probability, it is much more difficult to assess the result of simulation.

The following Scilab function generates a random number with the `rand` function and use the `floor` in order to get a random integer, either 1, associated with "Head", or 0, associated with "Tail". It prints out the result and returns the value.

```
// tossacoin --
//   Prints "Head" or "Tail" depending on the simulation.
//   Returns 1 for "Head", 0 for "Tail"
function face = tossacoin ( )
  face = floor ( 2 * rand() );
  if ( face == 1 ) then
    mprintf ( "Head\n" )
  else
    mprintf ( "Tail\n" )
  end
endfunction
```

With such a function, it is easy to simulate the toss of a coin. In the following session, we toss a coin 4 times. The "seed" argument of the `rand` is used so that the seed of the uniform random number generator is initialized to 0. This allows to get consistent results across simulations.

```
rand("seed",0)
face = tossacoin ();
face = tossacoin ();
face = tossacoin ();
face = tossacoin ();
```

The previous script produces the following output.

```
Tail
Head
Tail
Tail
```

Assume that we are tossing a fair coin 10 times. What is the probability that we get exactly 5 heads ?

This is a Bernoulli process, where the number of trials is $n = 10$ and the probability is $p = 5$. The probability of getting exactly $j = 5$ heads is given by the

binomial distribution and is

$$P(\text{"exactly 5 heads in 10 toss"}) = b(10, 1/2, 5) \quad (124)$$

$$= \binom{10}{5} p^5 q^{10-5}, \quad (125)$$

where $p = 1/2$ and $q = 1 - p$. The expected probability is therefore

$$P(\text{"exactly 5 heads in 10 toss"}) \approx 0.2460938. \quad (126)$$

The following Scilab session shows how to perform the simulation. Then, we perform 10000 simulations of the process. The `floor` function is used in combination with the `rand` function to generate integers in the set $\{0, 1\}$. The `sum` allows to count the number of heads in the experiment. If the number of heads is equal to 5, the number of successes is updated accordingly.

```
-->rand("seed",0);
-->nb = 10000;
-->success = 0;
-->for i = 1:nb
-->  faces = floor ( 2 * rand(1,10) );
-->  nbheads = sum(faces);
-->  if ( nbheads == 5 ) then
-->    success = success + 1;
-->  end
-->end
-->pc = success / nb
 pc  =
    0.2507
```

The computed probability is $P = 0.2507$ while the theory predicts $P = 0.2460938$, which means that there is lest than two significant digits. It can be proved that when we simulate an experiment of this type n times, we can expect that the error is less or equal to $\frac{1}{\sqrt{n}}$ at least 95% of the time. With $n = 10000$ simulations, this error corresponds to 0.01, which is the accuracy of our experiment.

4 Acknowledgments

I would like to thank John Burkardt for his comments about the numerical computation of the permutation function.

5 References and notes

The material for section 1.1 is presented in [6], chapter 1, "Set, spaces and measures". The same presentation is given in [3], section 1.2, "Discrete probability distributions". The example 1.2 is given in [3].

The equations 19, 16 and 21 are at the basis of the probability theory so that in [9], these properties are stated as axioms.

In some statistics books, such as [4] for example, the union of the sets A and B is denoted by the sum $A + B$, and the intersection of sets is denoted by the product AB. We did not use these notations in this document.

Combinatorics topics presented in section 2 can be found in [3], chapter 3, "Combinatorics". The example of Poker hands is presented in [3], while the probabilities of all Poker hands can be found in Wikipedia [11].

The gamma function presented in section 2.3 is covered in many textbook, as in [1]. An in-depth presentation of the gamma function is done in [10].

Some of the section 3 which introduces to random number generators is based on [5].

References

[1] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover Publications Inc., 1972.

[2] George E. Andrews, Richard Askey, and Ranjan Roy. *Special Functions*. Cambridge University Press, Cambridge, 1999.

[3] Charles Grinstead, M. and J. Snell, Laurie. *Introduction to probabilities, Second Edition*. American Mathematical Society, 1997.

[4] J. M. Hammersley and D. C. Handscomb. *Monte Carlo Methods*. Chapman and Hall, 1964.

[5] D. E. Knuth. *The Art of Computer Programming, Volume 2, Seminumerical Algorithms*. Third Edition, Addison Wesley, Reading, MA, 1998.

[6] M. Loève. *Probability Theory I, 4th Edition*. Springer, 1963.

[7] Michael A. Malcolm and Cleve B. Moler. Urand: a universal random number generator. Technical report, Stanford, CA, USA, 1973.

[8] Dmitry Panchenko. Introduction to probability and statistics. Lectures notes taken by Anna Vetter.

[9] M. Ross, Sheldon. *Introduction to probability and statistics for engineers and scientists*. John Wiley and Sons, 1987.

[10] Pascal Sebah and Xavier Gourdon. Introduction to the gamma function.

[11] Wikipedia. Poker probability — wikipedia, the free encyclopedia, 2009.

Index

Bernoulli, 31
binomial, 26

combination, 26
combinatorics, 12
complementary, 2
conditional
    distribution function, 9
    probability, 9

disjoint, 2

event, 4

factorial, 14, 21
fair die, 9

gamma, 16
grand, 34

intersection, 2

outcome, 4

permutation, 13
permutations, 23
poker, 29

rand, 34
random, 4
rank, 29

sample space, 4
seed, 35
subset, 2
suit, 29

tree diagram, 12

uniform, 8
union, 2

Venn, 3