UNIVERSITY OF LIVERPOOL

MATH368: Stochastic Theory and Methods in Data Science

— Lecture Notes —

Dr. Andreas Alpers

Academic year 2020-2021

Contents

1 Review of essential background from probability/statistics **1**
1.1 Experiments, events, probability . . . 1
1.2 Random variables . . . 1
1.3 Independence and conditioning . . . 5
1.4 Law of large numbers and central limit theorem . . . 7
1.5 Elements of stochastic processes in time and space . . . 8
1.6 Exercises . . . 11

2 Simulation: theory and practice **12**
2.1 Direct Monte Carlo methods . . . 12
2.2 Variance reduction techniques, importance sampling methods . . . 16
2.3 Exercises . . . 20

3 Pseudo random number generator, simulation methods for univ. distr. **22**
3.1 Sampling from a uniform univariate distribution . . . 22
3.2 Analyzing random number generators . . . 24
3.3 General methods for sampling from non-uniform distributions . . . 25
3.3.1 Cdf inversion . . . 25
3.3.2 Rejection sampling . . . 28
3.4 Sampling from a normal distribution . . . 32
3.4.1 Central Limit algorithm . . . 32
3.4.2 Box-Muller algorithm . . . 33
3.5 Exercises . . . 35

4 Markov Chain Monte Carlo (MCMC) methods **38**
4.1 Markov Chains . . . 38
4.2 Metropolis-Hastings . . . 43
4.3 Other samplers . . . 47
4.4 Simulated annealing . . . 48
4.5 An example of a Gibbs sampler . . . 49
4.6 Exercises . . . 52

5 Learning theory and methods **54**
5.1 The process of learning . . . 54
5.2 Supervised learning . . . 57
5.2.1 Vapnik–Chervonenkis dimension . . . 57
5.2.2 Support vector machines . . . 59
5.2.3 Naïve Bayes classifier . . . 61
5.2.4 Decision trees . . . 63
5.2.5 Neural networks . . . 65
5.3 Unsupervised learning . . . 65
5.3.1 Clustering . . . 65
5.3.2 Learning on huge feature spaces . . . 70
5.4 Reinforcement learning and Markov decision processes . . . 74
5.4.1 Dynamic programming . . . 75
5.5 Neural networks: Hopfield networks, Boltzmann machines . . . 77
5.5.1 Neural networks . . . 77
5.5.2 Hopfield networks . . . 78
5.5.3 Boltzmann machines . . . 81
5.5.4 Solving optimization problems with neural networks . . . 81
5.6 Exercises . . . 84

6 Epilogue **87**

1 Review of essential background from probability/statistics

We begin by reviewing some essentials from probability/statistics that we will need throughout this course. We will keep this rather briefly as familiarity with these topics is assumed. Good text books on this subject are, for instance, [1, 2, 3]. A summary of the general notation that will be used throughout this text can be found in Table 1.

Symbol	Meaning
$\mathbb{N}$	natural numbers: $1, 2, 3, \ldots$
$\mathbb{N}_0$	non-negative integers: $\mathbb{N} \cup \{0\}$
$\mathbb{Z}$	integers
$\mathbb{R}$	set of real numbers
$\mathbb{R}^d$	Euclidean space of dimension d with norm denoted by $\|\| \cdot \|\|$

Table 1: General notation.

1.1 Experiments, events, probability

Random phenomena are observed by means of **experiments** (performed either by man or nature). Each experiment results in an **outcome** (also called **sample point**). The collection of all possible outcomes ω is called the **sample space** Ω. Any subset A of the sample space Ω can be regarded as a representation of some **event**. An outcome ω **realizes** an event if $\omega \in A$. In probability theory, one assigns to each event $A \in \mathcal{F}$ (where $\mathcal{F} = \Omega$ or a collection of subsets of Ω satisfying certain axioms making it a so-called σ-field) a number, the so-called **probability** of the event. Formally, a probability on $(\Omega, \mathcal{F})$ is a mapping $\Pr : \mathcal{F} \to \mathbb{R}$ satisfying the axioms

(i) $0 \leq \Pr(A) \leq 1$, for any $A \in \mathcal{F}$,

(ii) $\Pr(\Omega) = 1$,

(iii) $\Pr(\bigcup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} \Pr(A_i)$, for any countable sequence of disjoint $A_1, A_2, \cdots \in \mathcal{F}$.

The triple $(\Omega, \mathcal{F}, \Pr)$ is called **probability space**. It should be noted that the simple axioms (i)-(iii) determine completely how probabilities operate. For instance, the probability that both events A and B occur is $\Pr(A \cap B) = \Pr(A) + \Pr(B) - \Pr(A \cup B)$. However, how should we interpret probabilities in the real-word? Does probability measure the physical tendency of something to occur or is it a measure of how strongly one believes it will occur, or does it draw on both these elements? This brings us to philosophical questions. There are mainly two broad categories of probability interpretations, which can be called 'frequency' and 'Bayesian' probabilities. In the former category, probability is interpreted as long-run frequency of the occurrence of each potential outcome when the identical experiment is repeated indefinitely. In the latter category probability is viewed as a numerical measure of subjective uncertainty on the experimental result[1].

Table 2 summarizes some basic probability theory notation that we use.

1.2 Random variables

Informally, one should think of random variable as a variable whose values depend on outcomes of a random phenomenon.

Definition 1.1. *Given any probability space $(\Omega, \mathcal{F}, \Pr)$, a **random variable** is a real-valued function X on Ω such that, for all real numbers t, the set $\{\omega \in \Omega : X(\omega) \leq t\}$ belongs to $\mathcal{F}$.*

[1] If you are interested in finding out more about these philosophical issues, I recommend https://plato.stanford.edu/entries/probability-interpret/.

Name	Symbol	Discrete	Continuous
cdf	F_X	$\Pr(X \leq x)$	$\Pr(X \leq x)$
pdf	f_X	$\Pr(X = x)$	$\frac{d}{dx}F_X(x)$
Probability	$\Pr(a < X \leq b)$	$\sum_{a<x\leq b} f_X(x) = F_X(b) - F_X(a)$	$\int_a^b f_X(x)dx = F_X(b) - F_X(a)$
Expectation	$\mathbb{E}[r(X)]$	$\sum_{x\in\Omega} r(x)f_X(x)$	$\int_{-\infty}^{\infty} r(x)f_X(x)dx$
Mean	$\mu = \mathbb{E}[X]$	$\sum_{x\in\Omega} xf_X(x)$	$\int_{-\infty}^{\infty} xf_X(x)dx$
Variance	$\text{var}(X)$	$\mathbb{E}[(X-\mu)^2]$	$\mathbb{E}[(X-\mu)^2]$

Table 2: Basic probability theory notation.

Example 1.1. *Consider the experiment of tossing a die once. The possible outcomes are* $\omega = 1, 2, 3, 4, 5, 6$, *and the sample space is the set* $\Omega = \{1, 2, 3, 4, 5, 6\}$. *Take for* X *the identity function* $X(\omega) = \omega$. *In that sense,* X *is a random number obtained by the experiment of tossing a die.*

Example 1.2. *Consider the experiment of tossing a coin an infinite number of times. As sample space* Ω *one can take the collection of all sequences* $\omega = \{a_i\}_{i\geq 1}$ *with* $a_i = 0$ *or* $a_i = 1$ *depending on whether the* i*th toss results in heads or tails. Define* X_i *to be the random number obtained at the* i*th toss:*

$$X_i(\omega) = a_i.$$

In the following, we will often use the notation $\Pr(X \leq a)$ and $\Pr(X \in A)$, where $a \in \mathbb{R}$ and A is a measurable set. This notation is an abbreviation for $\Pr(X \leq a) = \Pr(\{X \leq a\}) = \Pr(\{\omega \in \Omega : X(\omega) \leq a\})$ and $\Pr(X \in A) = \Pr(\{X \in A\}) = \Pr(\{\omega \in \Omega : X(\omega) \in A\})$, respectively.

Random variables can be discrete or continuous (or mixed).

Definition 1.2. *A* **discrete random variable** *is one that can assume only finitely many or countably infinitely many outcomes (but only one at a time, of course).*

Definition 1.3. *The function* f_X *that associates with each possible value of the (discrete) random variable* X *the probability of this value is called the* **probability distribution function (pdf)** *of* X.

In some literature, pdfs are called probability mass functions. Note that f_X satisfies: (i) $f_X(x_k) \geq 0$ for all x_k, and (ii) $\sum_{k=1}^{\infty} f_X(x_k) = 1$.

Definition 1.4. *The function* F_X *that associates with each real number* x *the probability* $\Pr(X \leq x)$ *that the random variable* X *takes on a value smaller or equal to this number, i.e.,* $F_X(x) = \sum_{x_k\leq x} f_X(x_k)$, *is called the* **cummulative distribution function (cdf)** *of* X.

It can be shown that F_X is non-decreasing and right-continuous.

Definition 1.5. *A* **continuous random variable** *is a random variable that may take an uncountably infinite number of values.*

Definition 1.6. *The* **probability density function (pdf)** *of a continuous random variable* X *is a function* f_X *defined for all* $x \in \mathbb{R}$ *and having the following properties:*

(a) $f_X(x) \geq 0$ *for any real number* x;

(b) if A *is any subset of* $\mathbb{R}$, *then*

$$\Pr(X \in A) = \int_A f_X(x)dx.$$

Note that the pdf is different from the pdf of a discrete random variable. Indeed, $f_X(x)$ does *not* give the probability that the random variable X takes on the value x. What can be said is that by

$$f_X(x)\varepsilon \approx \Pr(x - \varepsilon/2 \leq X \leq x + \varepsilon/2),$$

for small $\varepsilon > 0$, we have that $f_X(x)\varepsilon$ is approximately equal to the probability that X takes on a value in an interval of length ε about x.

Definition 1.7. *The* **cummulative distribution function (cdf)** F_X *of a continuous random variable X is defined by*

$$F_X(x) = \Pr(X \leq x) = \int_{-\infty}^{x} f_X(u)du.$$

Note that, by definition, we have

$$\Pr(X = x) = \Pr(X \leq x) - \Pr(X < x) = \int_{-\infty}^{x} f_X(u)du - \int_{-\infty}^{x^-} f_X(u)du = 0$$

for any real number x, where x^- means that the range of the integral is the open interval $(-\infty, x)$.

It can be shown that the cdf of a continuous random variable is continuous.

Before we recall several important random variables, let us say a few words about histograms. **Histograms** are special kinds of bar charts often used to depict a series of values, say, $u_1, \ldots, u_n$ of outcomes in an experiment. A convenient subdivision of the x-axis is created containing the values, for example by means of the points $c_1 < \cdots < c_K$ and a parameter w, called **bin width.** They establish intervals, so called **bins**, $[c_1 - w/2, c_1 + w/2), [c_2 - w/2, c_2 + w/2), \ldots, [c_K - w/2, c_K + w/2)$. A count is made of how many of the values lie in each bin and a bar of that height divided by nw is depicted standing on this bin. Technically, this is called a *density histogram for equal size bin widths* (to distinguish it from other types of histograms), but since we will be using only this type of histogram we will usually omit the word 'density.' Note that the area of the bars in that histogram sums up to 1.

Example 1.3. *Consider* $u_1, \ldots, u_{20} = 1, 1, 3, 5, 7, 8, 8, 2, 3, 5, 6, 7, 7, 5, 6, 7, 4, 9, 1, 9$. *As bin width w set* $w := 1$, *and as bin centers set* $c_i := i$, $i = 1, \ldots, 9$. *The corresponding histogram is shown in Fig. 1, the corresponding matlab code is*

```
[f, c] = hist(u,c); bar(c, f / n*w);
```

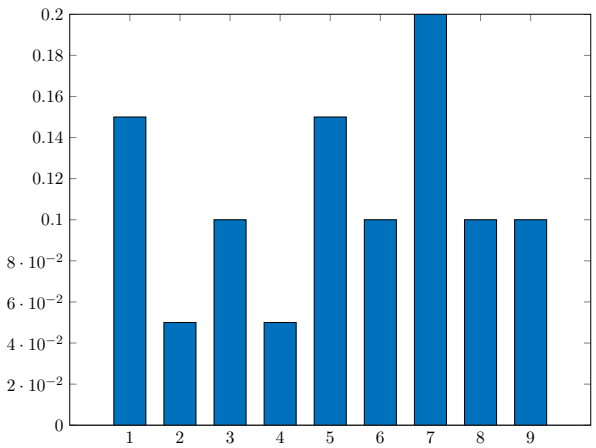


Figure 1: Histogram of $u_1, \ldots, u_{20} = 1, 1, 3, 5, 7, 8, 8, 2, 3, 5, 6, 7, 7, 5, 6, 7, 4, 9, 1, 9$.

Important random variables

Example 1.4. *Consider a* ***uniformly distributed discrete random variable*** $X \sim U(\{1, 2, \dots, n\})$, *i.e., the state space of* X *consists of the* n *numbers* $1, \dots, n$. *They all have equal probability, i.e.,*

$$f_X(x) = \begin{cases} \frac{1}{n} & : \quad x \in \{1, \dots, n\}, \\ 0 & : \quad \text{otherwise.} \end{cases}$$

Correspondingly, by summing up the individual probabilities, we obtain the cdf

$$F_X(x) = \begin{cases} 0 & : \quad x < 1, \\ \frac{k}{n} & : \quad k < x < k+1, \ k \in \{1, \dots, n-1\} \\ 1 & : \quad n \le x. \end{cases}$$

See top row of Fig. 2.

Example 1.5. *Consider a* ***uniformly distributed continuous random variable*** $X \sim U(a, b)$, *i.e., on the interval* (a, b) *the probability density function should be constant. As* $\int_{\mathbb{R}} f_X(x)dx = 1$ *is required, we see that*

$$f_X(x) = \begin{cases} \frac{1}{b-a} & : \quad a < x < b, \\ 0 & : \quad \text{otherwise.} \end{cases}$$

As cdf we have

$$F_X(x) = \int_{-\infty}^{x} f_X(u)du = \begin{cases} 0 & : \quad x \le a, \\ \frac{1}{b-a}(x-a) & : \quad a < x < b, \\ 1 & : \quad b \le x. \end{cases} \tag{1.1}$$

It is well known that $\mathbb{E}[X] = (a+b)/2$ *and* $\mathrm{var}(X) = \sigma^2 = (b-a)^2/12$.

Let us remark, because we will use this in the following rather often, that for $X \sim U(0, 1)$ we have

$$F_X(x) = \begin{cases} x & : \quad 0 < x < 1, \\ 0 & : \quad \text{otherwise,} \end{cases} \qquad \text{and} \qquad f_X(x) = \begin{cases} 1 & : \quad 0 < x < 1, \\ 0 & : \quad \text{otherwise.} \end{cases}$$

The single most important random variable type, next to uniform random variables, is the **normal (also known as Gaussian) random variable,** parametrized by a mean μ and variance σ^2 (see Fig. 3). If X is a normal random variable, we write $X \sim N(\mu, \sigma^2)$. The pdf of a normal $X \sim N(\mu, \sigma^2)$ is:

$$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}.$$

Theorem 1.1. *Let* $X \sim N(\mu, \sigma^2)$ *and* $a, b \in \mathbb{R}$ *with* $a > 0$. *Then,* $Y = aX + b \sim N(a\mu + b, (a\sigma)^2)$.

Proof. Let A be measurable and $B = (A - b)/a$. Then,

$$\begin{aligned} \Pr(Y \in A) &= \Pr(aX + b \in A) \\ &= \Pr(X \in B) \\ &= \frac{1}{\sigma\sqrt{2\pi}} \int_{x \in B} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \, dx \\ &\overset{(*_1)}{=} \frac{1}{\sigma\sqrt{2\pi}} \int_{y \in A} e^{-\frac{1}{2}\left(\frac{\frac{y-b}{a}-\mu}{\sigma}\right)^2} \frac{1}{a} \, dy \\ &= \int_{y \in A} \underbrace{\frac{1}{a\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{y-(a\mu+b)}{a\sigma}\right)^2}}_{=f_Y(y)} dy, \end{aligned}$$

where $(*_1)$ follows from the transformation theorem with $y = ax + b$ (therefore $dx = \frac{1}{a}dy$). □

In particular, if $X \sim N(0, 1)$ then $Y = \sigma X + \mu \sim N(\mu, \sigma^2)$. This will later help us to generate normally distributed random numbers, since this tells us that to generate random numbers from $N(\mu, \sigma^2)$ it suffices to generate number from $N(0, 1)$ (which then only need to be transformed via $Y = \sigma X + \mu$). The $N(0, 1)$ distribution is called the **standard normal distribution**.

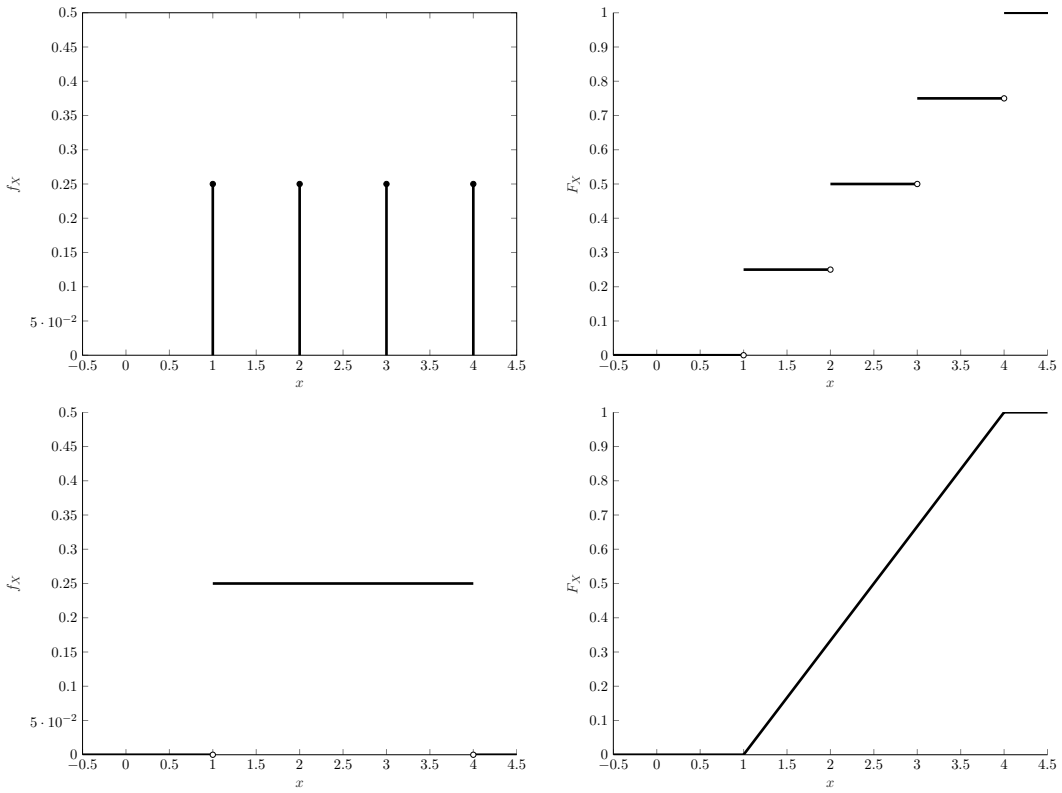


Figure 2: Uniform distribution. Top row: Discrete random variable $X \sim U(\{1, 2, 3, 4\})$. (left) pmf, (right) cdf. Bottom row: Continuous random variable $X \sim U(1, 4)$. (left) pdf, (right) cdf.

1.3 Independence and conditioning

The **conditional probability** of A given B is defined as

$$\Pr(A|B) := \frac{\Pr(A \cap B)}{\Pr(B)}. \tag{1.2}$$

The interpretation of this concept is that if we are told that event B has already occurred, the consideration of whether A occurs should be confined to the smaller universe – characterized by the occurrence of B. Therefore, we say that two events A and B are **independent** if, and only if,

$$\Pr(A \cap B) = \Pr(A)\Pr(B).$$

Note that the probabilities $\Pr(A|B)$ and $\Pr(B|A)$ are conceptually different quantities. While sometimes one of these quantities is known, one might want to compute the other. From (1.2), using

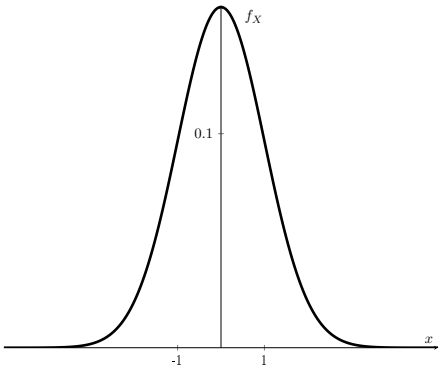


Figure 3: Probability density function for the normal $X \sim N(\mu, \sigma^2)$, with $\mu = 0$ and $\sigma^2 = 1$. With these parameters X is typically referred to as standard normal.

the fact that $\Pr(A \cap B)$ is symmetric in A and B, we have

$$\Pr(A|B)\Pr(B) = \Pr(A \cap B) = \Pr(B|A)\Pr(A).$$

Rearranging we obtain (one form of) **Bayes' formula**[2]:

$$\Pr(A|B) = \frac{\Pr(B|A)\Pr(A)}{\Pr(B)}.$$

Example 1.6. *Suppose a patient exhibits symptoms that make her physician concerned that she may have a particular disease. The disease is relatively rare in this population, with a prevalence of* 0.1% *(meaning it affects 1 out of every* $1,000$ *persons). The physician recommends a screening test that is rather expensive. Before agreeing to the screening test, the patient wants to know what will be learned from the test, specifically she wants to know the probability of disease, given a positive test result.*

The physician reports that the screening test is widely used and has a reported accuracy of 85%*. Hence, over the whole population, the test comes back positive* 15.07% *of the time and negative* 84.93% *of the time.*

Let A denote the event that a person has the disease, and let B denote the event that the test is positive. What we are looking for is the probability $\Pr(A|B)$*, i.e., given a positive test what is the probability that the person has the disease.*

From the data we know $\Pr(A) = 0.001$*. Also,* $\Pr(B|A) = 0.85$*, and* $\Pr(B) = 0.1507$*. Plugging this into Bayes' formula yields*

$$\Pr(A|B) = \frac{\Pr(B|A)\Pr(A)}{\Pr(B)} = \frac{0.85 \cdot 0.001}{0.1507} = 0.0056.$$

Hence, if the patient undergoes the test and it comes back positive, there is a 0.56% *chance that she has the disease. Also, note, however, that without the test, there is a* 0.1% *chance that she has the disease. In view of this, do you think the patient have the screening test?*

Two random variables X and Y are called **independent** if for any two subsets of real numbers $A, B \subseteq \mathbb{R}$, the events $\{X \in A\}$ and $\{Y \in B\}$ are independent events, hence if

$$\Pr(\{X \in A\} \cap \{Y \in b\}) = \Pr(\{X \in A\})\Pr(\{Y \in B\}).$$

Example 1.7. *Suppose, a coin is thrown twice. Let X be defined to be the number of heads that are observe. Then, a coin is thrown three times. This time the number of heads is recorded by Y. What is* $\Pr(\{X \leq 2)\} \cap \{Y \geq 1\})$*?*

Since X and Y are results of different independent coin tosses, the random variables X and Y are independent. Then,

$$\Pr(\{X \leq 2\} \cap \{Y \geq 1\}) = \Pr(\{X \leq 2\})\Pr(\{Y \geq 1\}) = 1 \cdot \frac{7}{8} = \frac{7}{8}.$$

Let us also recall what a **joint distribution** of two random variables X and Y is. First, the discrete case. If X and Y are discrete, their **joint pdf** is

$$f_{X,Y}(x, y) = \Pr(\{X = x\} \cap \{Y = y\}) \quad (=: \Pr(X = x \text{ and } Y = y)).$$

This is a function defined on discrete points in the x, y-plane; these points make up Ω. For an event A we hence have

$$\Pr(X, Y \in A) = \sum_{(x,y) \in A} f_{X,Y}(x, y).$$

For continuous random variables X and Y, the **joint cdf** is

$$F_{X,Y}(x, y) = \Pr(X < x, Y < y) \quad (=: \Pr(\{X < x\} \cap \{Y < y\})).$$

[2]Named after the English mathematician and reverend Thomas Bayes (1701 - April 7th, 1761)

The pdf is the derivative of the cdf. For a subset A of the x, y-plane,

$$\Pr(A) = \int\int_{(x,y)\in A} f_{X,Y}(x,y)\, dxdy.$$

Example 1.8. *A joint probability density can be created from any two univariate densities, simply by multiplying them together. Let X have density f_X and Y density f_Y; then $fX,Y(x,y) = f_X(x)f_Y(y)$ is a joint density. A concrete example of this is the joint pdf of rolling two dice, where*

$$\Pr(\text{Dice}_1 = i \text{ and } \text{Dice}_2 = j) = \Pr(\text{Dice}_1 = i)\Pr(\text{Dice}_2 = j).$$

1.4 Law of large numbers and central limit theorem

We briefly review here two fundamental results for convergence of random variables, which are widely used in practice and in the context of Monte Carlo simulations. Proofs of the results can be found in many textbooks on basic probability theory; see, e.g., [4].

Before we state the theorems, let us recall two notions of how a sequence of random variables becomes more and more 'stable' (i.e., converges to another random variable). Let $X_1, X_2, \ldots$ be a sequence of random variables and let F_1, $F_2, \ldots$ be the corresponding sequence of cdfs. If the cdfs become more and more similar to the cdf F of a common random variable X as $n \to \infty$, then we say that they converge to X in *distribution*. Mathematically, this means

$$\lim_{n\to\infty} F_n(x) = F(x), \quad \text{at all values of } x \text{ where } F \text{ is continuous.}$$

Note that although we say a sequence of random variables converges in distribution, it is, by definition, really the cdfs that converge, not the random variables. In this way this convergence is quite different from convergence almost surely, which we recall in the following.

We say that X_n converge to X *almost surely* (abbreviated as a.s.) if

$$\Pr\left(\{\omega \in \Omega \,:\, \lim_{n\to\infty} X_n(\omega) = X(\omega)\right) = 1.$$

This type of convergence is similar to pointwise convergence of a sequence of functions, except that the convergence need not occur on a set with probability 0 (hence the 'almost sure').

It can be shown that almost sure convergence implies convergence in distribution.

Example 1.9. *Take the probability space on which all random variables are defined as that corresponding to the $U(0,1)$ distribution. Thus $\Omega = (0,1)$, and the probability of any interval in Ω is its length. Define*

$$X_n(\omega) = \begin{cases} 0 & : \quad 0 < \omega < \frac{1}{2} - \frac{1}{n}, \\ n & : \quad \frac{1}{2} - \frac{1}{n} \le \omega < \frac{1}{2} + \frac{1}{n}, \\ 0 & : \quad \frac{1}{2} + \frac{1}{n} \le \omega < 1. \end{cases}$$

For any any $\omega < 1/2$ select N_1 so that $1/N_1 < 1/2 - \omega$. If $n \ge N_1$, then $\omega < 1/2 - 1/n$, hence $X_n(\omega) = 0$. For any any $\omega > 1/2$ select N_2 so that $1/N_2 < \omega - 1/2$. If $n \ge N_2$, then $\omega > 1/2 + 1/n$, hence $X_n(\omega) = 0$. In summary,

$$\{\omega : \lim_{n\to\infty} X_n(\omega) = 0\} = \{\omega : 0 < \omega < 1\},$$

which implies $\Pr(\{\omega : \lim_{n\to\infty} X_n(\omega) = 0\}) = 1$ and therefore $X_n \to 0$ almost surely.

Let us now discuss the law of large numbers and the central limit theorem. Both theorems come in different versions (depending on the assumptions imposed on the random variables). The law of large numbers that we state here is, in fact, the strong law of large numbers.

Basically, both the law of large numbers and the central limit theorem give us a picture of what happens when we take many independent samples from the same distribution.

Theorem 1.2 (Law of large numbers and central limit theorem).
Let $X_1, X_2, \ldots, X_n$ be independent random samples from a distribution with finite mean μ and finite variance σ^2. Then,

$$\text{(Law of large numbers:)} \qquad \lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} X_i = \mu \text{ a.s., and}$$

$$\text{(Central limit theorem:)} \qquad \sqrt{n}\left(\frac{1}{n}\sum_{i=1}^{n} X_i - \mu\right) \to N(0,\sigma^2) \text{ in distribution.}$$

The law of large numbers tells us two things: First, the average of many independent samples is (with high probability) close to the mean of the underlying distribution. And, second, this histogram of many independent samples is (with high probability) close to the graph of the pdf of the underlying distribution.

The central limit theorem, on the other hand, says that the average (or scaled average) of many independent copies of a random variable is approximately a normal random variable. It gives a sense of how fast $\frac{1}{n}\sum_{i=1}^{n} X_i$ approaches μ as n increases.

We remark that as $\sqrt{n}(\frac{1}{n}\sum_{i=1}^{n} X_i - \mu) \to N(0,\sigma^2)$ in distribution by the central limit theorem, we obtain by dividing by $\sqrt{n}$ and Theorem 1.1 the result that $\frac{1}{n}\sum_{i=1}^{n} X_i - \mu$ is approximately an $N(0,\sigma^2/n)$ random variable for large n.

An illustration of the law of large numbers and the central limit theorem is given in Fig. 4 and 5, respectively.

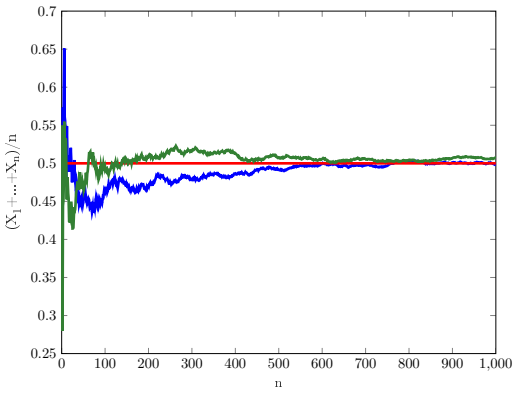


Figure 4: Illustration of the law of large numbers. $X_1, \ldots, X_n \sim U(0,1)$. Theoretical sample mean $\mu = 0.5$ is depicted in red. For two different realizations, $\frac{1}{n}\sum_{i=1}^{n} X_i$ are plotted (in blue and green).

1.5 Elements of stochastic processes in time and space

By a *stochastic process,* we shall mean a family of random variables $\{X_t\}$, where t is a point in a space T called the *parameter space,* and where, for each $t \in T$, X_t is a point in a space S called the *state space.*

The parameter t is often interpreted as 'time.' For example, X_t can be the price of a financial asset at time t. If $T = \mathbb{N}$ then we have nothing else then a sequence of random numbers. Sequences of random numbers are thus a special case of random processes. It may also happen that t should be interpreted as 'space.' For instance, X_t might be the water temperature at a location $t = (u,v,w) \in \mathbb{R}^3$ in the ocean. Or $t = (u,v) \in \mathbb{R}^2$ might represent a pixel location in a computer image (which is often usful in image processing applications). Also, mixed interpretations as 'space-time' are possible. For example, X_t with $t = (u,v,w,s) \in \mathbb{R}^4$ might be the ocean temperature measured at $(u,v,w) \in \mathbb{R}^3$ at time $s \in \mathbb{R}$.

The family $\{X_t\}$ may be thought of as the path of a particle moving 'randomly' in space S, its position at time t being X_t. A record of one of these paths is called a *realization* or *sample path* of the process.

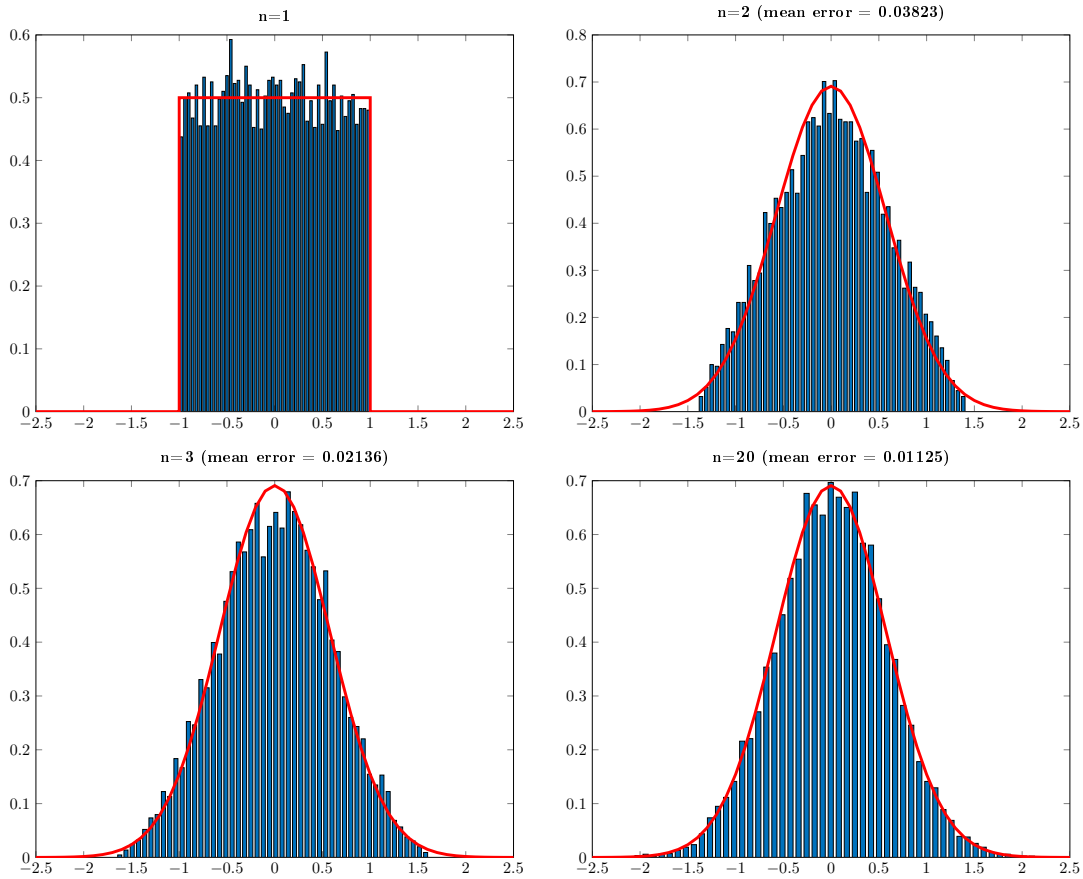


Figure 5: Illustration of the central limit theorem: Histograms of $\sqrt{n}\cdot\frac{1}{n}\sum_{i=1}^{n} X_i$ for $n = 1, 2, 3, 20$ where the $X_i \sim U(-1, 1)$. Number of samples is 10^4. For $n = 2, 3, 20$ the pdf for the normal $X \sim N(0, \sigma^2)$ with $\sigma^2 = \frac{1}{12}(1 + 1)^2 = 1/3$ is shown in red.

Example 1.10. *Let us consider the problem of modeling the score during a football match as a stochastic process.*

The state space S needs to present all possible values the score can take. Hence, a suitable choice is $S = \{(x, y) : x, y = 0, 1, 2, \dots\}$. Measuring times in minutes we can take as parameter space T the interval $[0, 90]$ (not considering overtimes). The process starts in state $(0, 0)$, and transition takes place between the states of S whenever a goal is scored. A goal increases x or y by one, so the score (x, y) will then go to $(x + 1, y)$ or $(x, y + 1)$.

Example 1.11. *Let $X_1, X_2, \dots$ be a sequence of independent and identically distributed (i.i.d.) random variables. The process $\{X_i\}$ is sometimes called* i.i.d. noise. *The parameter space is $T = \mathbb{N}$ and the state space is $S = \mathbb{R}$. A realization of this process is shown in Fig. 6(a). A histogram of $20,000$ realizations of $X_{1,000}$ is shown in Fig. 6(c).*

Example 1.12. *Let $Y_1, Y_2, \dots$ be a sequence of independent and identically distributed random variables. Define*

$$X_t := X_{t-1} + Y_t, \qquad t \in \mathbb{N}, \quad X_0 = 0.$$

The process $\{X_t\}$ is called random walk. *The parameter space is $T = \mathbb{N}$ and the state space is $S = \mathbb{R}$. A sample path of this process is shown in Fig. 6(b). A histogram of $20,000$ realizations of $X_{1,000}$ is shown in Fig. 6(d).*

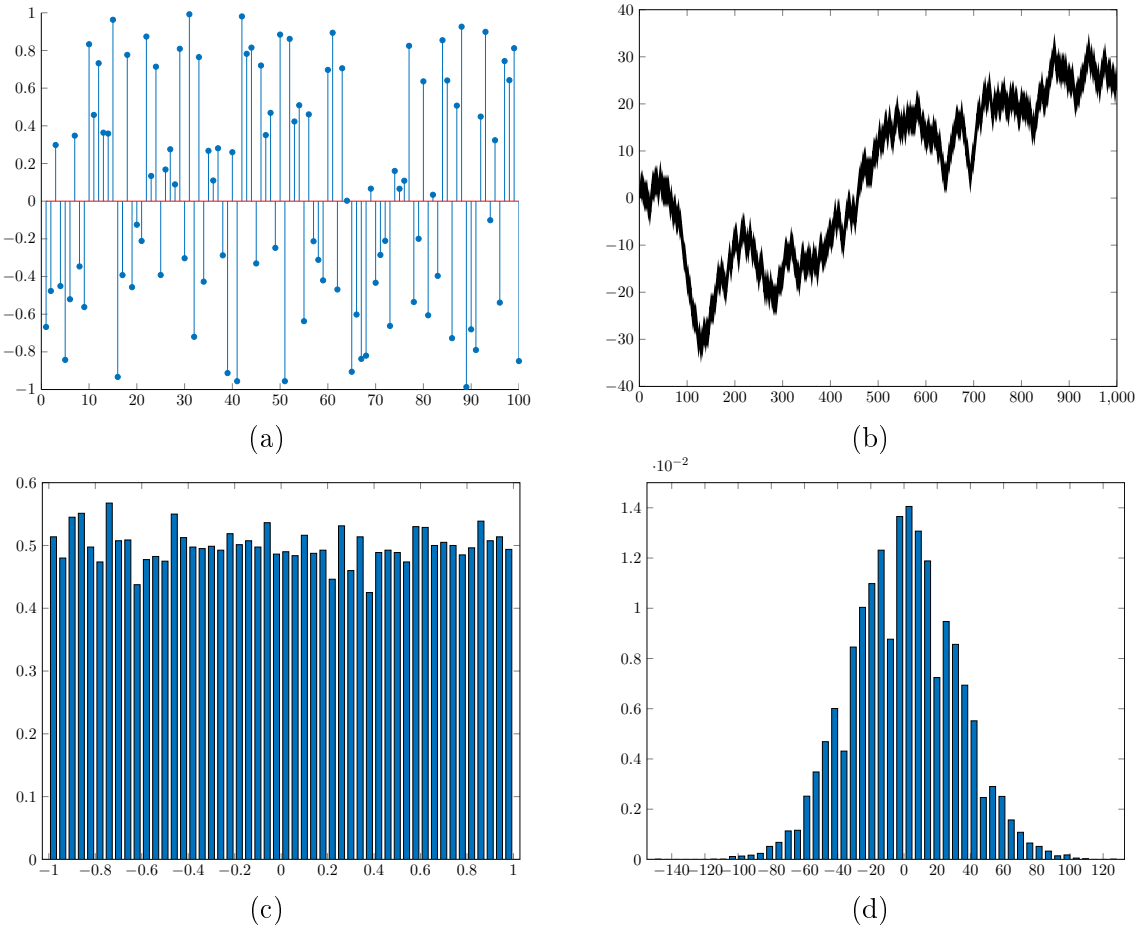


Figure 6: I.i.d noise and random walks: (a) sample path of i.i.d. noise, (b) sample path of a random walk, (c) histogram of $20,000$ realizations of i.i.d. noise variable $X_{1,000}$, (d) histogram of $20,000$ realizations of random walk variable $X_{1,000}$.

1.6 Exercises

Exercise 1.1. Let A and B two events. Prove from the axioms for Pr that

$$\Pr(A \cup B) = \Pr(A) + \Pr(B) - \Pr(A \cap B).$$

Exercise 1.2. An early detection test for HIV is 98% accurate. In a certain community, 3% of the population has HIV. What is the probability that a person with a positive test for HIV actually has the disease? Simulate this by selecting a person at random from the community and deciding whether the person has HIV. Then administer the test and decide whether it was positive. Keep a counter for the times the test is positive and for the times the test is positive and the person has HIV.

Exercise 1.3. Let $X_1, \ldots, X_n$ be exponentially distributed random variables (i.e., their pdfs are $f_{X_i}(t) = \lambda e^{-\lambda t}$) with $\lambda = 0.2$. Using matlabs `exprnd` command draw `nTrials` $= 10^4$ samples of each X_i and plot a histogram of

(i) X_1 and

(ii) $\sqrt{n} \cdot \left(\frac{1}{n}\sum_{i=1}^{n} X_i - \mu\right)$ for $n = 1$, 5, and 30.

How do your plots compare with the results predicted by the central limit theorem?

Exercise 1.4. Let X be a standard Cauchy random variable, i.e., X has the pdf

$$f_X(x) = \frac{1}{\pi}\frac{1}{1+x^2}, \qquad x \in \mathbb{R}.$$

Let $Y = |X|$. It can be shown that X and Y have no finite mean.

(a) Plot the pdf of X.

(b) Give plots of four realizations of $\frac{1}{n}\sum_{i=1}^{n} Y_i$. What do you observe?

(Hint: In matlab, `nTrials` standard Cauchy random variables can be generated by the command `tan(pi*(rand(1,nTrials)-0.5))`.)

Exercise 1.5. Let X be a random variable, and let $a_1, a_2, \ldots$ be a sequence of real numbers converging to zero. Define $X_n := a_n X$, $n \in \mathbb{N}$. Show that X_n converges almost surely to zero.

2 Simulation: theory and practice

Many times we wish to simulate the results of an experiment by using a computer and random variables, using the (pseudo-)random number generators available on computers. This is known as **Monte Carlo simulation**[3].

There can be several reasons why one wants to do that. For instance,

- The experiment could be quite complicated to set up (or being controlled) in practice;
- Monte Carlo methods are generally very simple to implement.
- They can be used to solve a very wild range of problems, even problems that have no inherent probabilistic structure, as, e.g., computation of multivariate integrals and solving linear systems of equations. They can also be used to solve optimization problems.

In general, Monte Carlo methods can be divided into two types: *direct (or simple) Monte Carlo* and *Markov Chain Monte Carlo (MCMC)*.

- **Direct Monte Carlo**: In this type of Monte Carlo the samples X_i that we generate are an i.i.d. sequence. So the strong law of large numbers tells us that the average of the X_i, i.e., the sample mean $\frac{1}{n}\sum_{i=1}^{n} X_i$, will converge to the mean of X as $n \to \infty$. Furthermore, the central limit theorem tells us a lot about the error in our computation.
- **Markov Chain Monte Carlo**: These methods construct a Markov Chain whose stationary distribution is the probability measure we want to simulate. We then generate samples of the distribution by running the Markov Chain. As the chain runs we compute the value X_n of our random variable at each time step n. The samples X_n are not independent, but there are theorems that tell us the sample mean still converges to the mean of our random variable.

2.1 Direct Monte Carlo methods

As an initial example, let us consider the problem of computing an integral.

Direct Monte Carlo: Integration Consider the problem of computing a (possibly multi-dimensional) definite integral

$$\mu := \int_D g(x)\, dx, \tag{2.1}$$

for some $g : \mathbb{R}^d \to \mathbb{R}^n$ and interval $D = [0,1]^d \subseteq \mathbb{R}^d$ (considering such a D makes our presentation easier — by transformations this is typically not a real restriction). Let X be a random variable that is uniformly distributed on D. Then, the expected value of $g(X)$ is, by definition,

$$\mathbb{E}[g(X)] = \int_D g(x)\, dx = \mu.$$

Now, suppose we can draw independent and identically distributed random samples $X_1, \ldots, X_n$ uniformly from D (by a computer), and we set $\hat{\mu}_n := \frac{1}{n}\sum_{i=1}^{n} g(X_i)$. Then the law of large numbers tells us (for 'nicely behaving' g and D) that almost surely

$$\lim_{n\to\infty} \hat{\mu}_n = \mathbb{E}[g(X)] = \mu.$$

We can therefore approximate μ by simulation (drawing $X_1, \ldots, X_N$ and computing the mean).

Let's analyze the accuracy of this approach. Let's assume that the assumptions of the central limit theorem are met. Then the theorem tells us that $\sqrt{n}(\hat{\mu}_n - \mu)$ for $n \to \infty$ converges in distribution to a

[3] The concept was first popularized right after World War II. Mathematician Stanislaw Ulam, working on nuclear weapons projects at the Los Alamos National Laboratory, coined the term in reference to an uncle who loved playing the odds at the Monte Carlo casino. A nice read about the beginnings of the Monte Carlo is the article https://la-science.lanl.gov/cgi-bin/getfile?00326866.pdf by N. Metropolis.

random variable $X \sim N(0, \sigma^2)$. And, by Thm. 1.1, we therefore have for large n that $\mathbb{E}[\hat{\mu}_n] = \mathbb{E}[\mu] = \mu$ and $\mathbb{E}[(\hat{\mu}_n - \mu)^2] = \mathrm{var}(\hat{\mu}_n) = \sigma^2/n$, where σ^2 is the variance of $g(X)$. The quantity

$$\sqrt{\mathbb{E}[(\hat{\mu}_n - \mu)^2]} = \sigma/\sqrt{n} \tag{2.2}$$

is called the **root mean square error** (RMSE) of $\hat{\mu}_n$.

As the RMSE is $\sigma/\sqrt{n}$ one can say that the RMSE is of the order $O(1/\sqrt{n})$ as $n \to \infty$ (this is *Landau O-notation*[4]). To obtain one more decimal digit of accuracy (i.e., an RMSE multiplied by a factor of 1/10) one therefore needs a 100-fold increase in computation. Remarkably, this error does not depend on d (as for instance, in other deterministic approaches).

Let us look at this in more detail at the case $d = 1$. Here one can approximate μ deterministically by

$$\tilde{\mu}_n := \frac{1}{n} \sum g(i/n).$$

This method may be called Riemann approximation. When g is reasonably smooth, the Riemann approximation has an error rate of $O(1/n)$, which is better than in direct Monte Carlo. There are even better methods, based on Simpson's rule and or Newton-Cotes rules, but all these methods do not scale well as the dimensionality of D increases.

For example, in a 10-dimensional space with $D = [0, 1]^{10}$, we will have to evaluate $O(n^{10})$ grid points in order to achieve an accuracy of $O(1/n)$ in the Riemann approximation of μ. Monte Carlo would only need $O(n)$ points to achieve an accuracy of $O(1/\sqrt{n})$.

We remark that the central limit theorem can also be used to compute confidence intervals for the results obtained by direct Monte Carlo. The interested reader can find more information for instance in [5, Chapter 1].

One should also be aware of the following. Do not confuse the σ in (2.2) with the σ of the random samples $X_1, \ldots, X_n$ that are drawn. The σ in (2.2) is the σ of $g(X)$, i.e., $\sigma^2 = \mathrm{var}(g(X))$ as we have seen in connection with (2.2). Often $\mathrm{var}(g(X))$ is not known and needs to be estimated from the data. In most of our examples, however, we can compute it explicitly since

$$\mathrm{var}(g(X)) = \mathbb{E}[(g(X) - \mu)^2] \overset{(*)}{=} \mathbb{E}[g(X)^2] - \mu^2,$$

and $g(X)$ (hence $\mathbb{E}[g(X)^2]$) and μ are known (note, $(*)$ follows by standard manipulations; $\mathbb{E}[(g(X) - \mu)^2] = \mathbb{E}[g(X)^2 - 2g(X)\mu + \mu^2] = \mathbb{E}[g(X)^2] - 2\mathbb{E}[g(X)]\mu + \mathbb{E}[\mu^2] = \mathbb{E}[g(X)^2] - 2\mu^2 + \mu^2 = \mathbb{E}[g(X)^2] - \mu^2$ using only linearity of expectation). We will come back to this point later when we discuss importance sampling.

It is time to wrap things up.

Direct Monte Carlo for computing integrals of the form (2.1):

(1) Choose a large n.

(2) Draw independent and identically distributed random samples $X_1, \ldots, X_n$ uniformly from D.

(3) Return $\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} g(X_i)$ as an estimate of the integral.

Example 2.1. *Suppose we want to compute*

$$\mu = \int_0^1 x^3 \, dx.$$

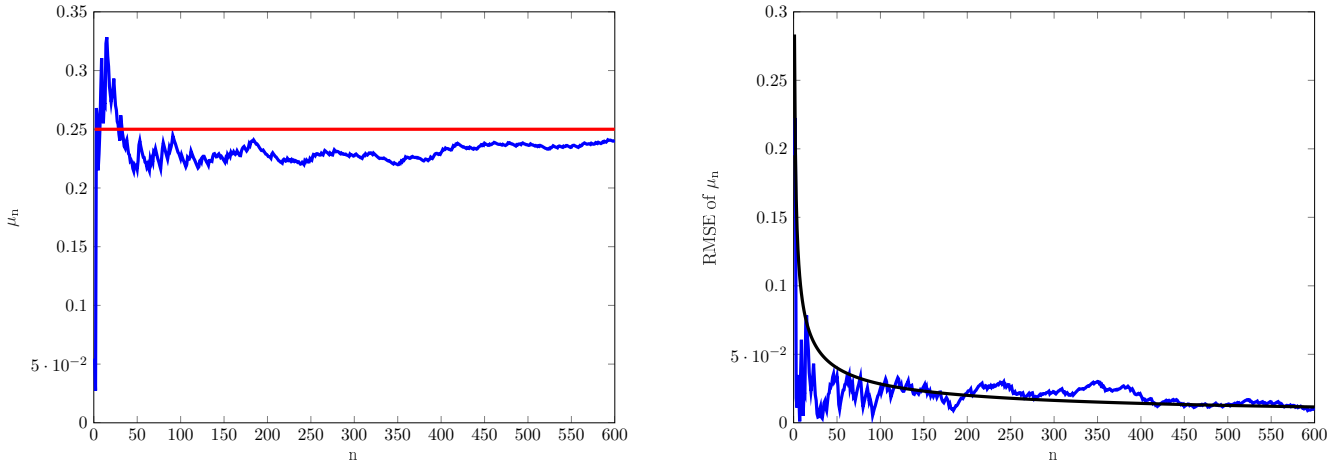
Of course, we know that we should obtain $\mu = [\frac{1}{4}x^4]_0^1 = \frac{1}{4} = 0.25$. *Let's implement the direct Monte Carlo method. As* $\sigma^2 = \mathrm{var}(g(X)) = \mathbb{E}[g(X)^2] - \mu^2 = \int_0^1 x^6 \, dx - 1/16 = 1/7 - 1/16 = 9/112$, *we can also plot the RMSE* $\sigma/\sqrt{n}$.

[4] One says, f is in $O(g)$ if, and only if there is a positive real number C and a real number x_0 such that $|f(x_0)| \leq Cg(x)$ for all $x \geq x_0$.

Here is the matlab code. Fig. 7 shows a sample path of the $\hat{\mu}_n$, the realized RMSE, and the theoretical RMSE $\sigma/\sqrt{n}$.

```
nTrials=6*10^2;

mu=0.25;
sigma2=1/7-mu^2;

u=rand(1,nTrials);
for n=1:nTrials
    muhat(n)=mean(u(1:n).^3); %here the function g(x)=x^3 comes in
end;
figure, plot(1:nTrials,muhat,'b','LineWidth',2); xlabel('n'); ylabel('\mu_n');
hold on; plot(1:nTrials,mu*ones(1,nTrials),'r','LineWidth',2);

figure, plot(1:nTrials,abs(muhat-mu),'b','LineWidth',2); xlabel('n'); ylabel('RMSE of \mu_n'); hold on;
plot(1:nTrials,sqrt(sigma2)./sqrt(1:nTrials),'k','LineWidth',2);
```

MCintegralexample1.m



(a) (b)

Figure 7: Direct Monte Carlo for Example 2.1. (a) Sample path of $\hat{\mu}_n$ as a function of n (blue) and $\mu = 0.25$ (red), (b) corresponding (realized) RMSE as a function of n (blue) and plot of $\sigma/\sqrt{n}$ (black).

Let's consider a second example.

Direct Monte Carlo: Buffon needle problem (toy example) The French nobleman, Comte de Buffon[5], posed the following problem in 1777:

Suppose that you drop a short needle on a ruled paper — what is then the probability that the needle comes to lie in a position where it crosses one of the lines?

It is intuitively clear that the probability should be an increasing function of the length of the needle and a decreasing function of the spacing between the parallel lines. But that the probability, as we will see next, will depend on π is perhaps quite unexpected.

We denote, in the following, the distance between the parallel lines of the ruled paper by d and the length of the needle by L, respectively. A short needle in our context satisfies $L \leq d$. Note that such a needle cannot cross two lines at the same time. Now assume we drop such a short needle. How can we check whether the needle crosses one of the parallel lines?

[5]Comte de Buffon (1707–1788)

Well, let's introduce two parameters X and θ, where X denotes the distance of the midpoint of the needle to the nearest of the parallel lines and θ denotes the acute[6] angle between the needle an the parallel lines. It is easy to see that the needle crosses one of the lines if and only if $X \leq \frac{L}{2}\sin(\theta)$; see Fig. 8.

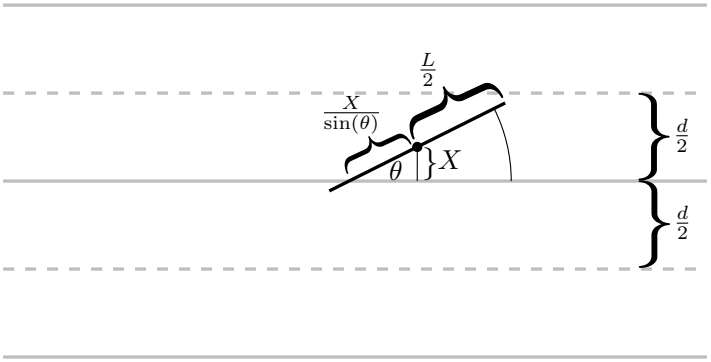


Figure 8: Sketch of the Buffon needle problem. The needle crosses a line if and only if $X/\sin(\theta) \leq \frac{L}{2}$, or, in other words, $X \leq \frac{L}{2}\sin(\theta)$.

Now we can model Buffon's experiment. Based on our notation, we can consider X to be a uniform random variable over $0 \leq X \leq d/2$. Its probability density function is

$$f_X(x) = \begin{cases} \frac{2}{d} & : \quad 0 \leq x \leq d/2, \\ 0 & : \quad \text{otherwise.} \end{cases}$$

Also θ can be considered to be a uniform random variable over $0 \leq \theta \leq \pi/2$, hence with probability density function

$$f_\theta(\theta) = \begin{cases} \frac{2}{\pi} & : \quad 0 \leq \theta \leq \pi/2, \\ 0 & : \quad \text{otherwise.} \end{cases}$$

The two random variables, X and θ, are independent. Therefore the joint probability density function of (X, θ) is

$$f_{X,\theta}(x,\theta) = \begin{cases} \frac{4}{\pi d} & : \quad 0 \leq x \leq d/2 \text{ and } 0 \leq \theta \leq \pi/2, \\ 0 & : \quad \text{otherwise.} \end{cases}$$

The probability that the short needle crosses one of the parallel lines is therefore

$$\begin{aligned} \Pr(\text{short needle crosses a line}) &= \int_0^{\pi/2} \int_0^{\frac{L}{2}\sin(\theta)} f_{X,\theta}(x,\theta)\, dx\, d\theta \\ &= \int_0^{\pi/2} \int_0^{\frac{L}{2}\sin(\theta)} \frac{4}{\pi d}\, dx\, d\theta \\ &= \frac{4}{\pi d} \cdot \left[-\frac{L}{2}\cos(\theta)\right]_0^{\pi/2} \\ &= \frac{2L}{\pi d}. \end{aligned} \tag{2.3}$$

So, this solves Buffon's needle problem. But there is more to the story. We can actually use the previous results to devise an experiment to approximate π in a random fashion[7].

[6] The acute angle ('acute' meaning 'small') is the angle in the range between 0 and $\pi/2$.

[7] The observant reader will notice that the computer simulation makes explicit use of π in generating the random numbers (more precisely, to generate θ) — so in a sense this is a little bit of cheating. This, however, can be fixed. See, e.g., https://www.scirp.org/journal/paperinformation.aspx?paperid=74541.

Suppose we set up an experiment (or a computer simulation) of throwing such a needle. The values L and d can therefore assumed to be known. Let $Y_1, \ldots, Y_n$ be an i.i.d. sequence of random variables (functions of θ and X) with Y_i describing whether in the ith throw the short needle hits the line ($Y_i = 1$) or whether it doesn't ($Y_i = 0$). Let $A = \{1\}$ denote the event that the needle hits the line. Then,

$$\mu := \mathbb{E}[Y_i] = \int_{\mathbb{R}^2} Y_i(y) f_{Y_i}(y)\, dy = \int_A f_{Y_i}(y)\, dy = \Pr(A).$$

Now, by the law of large numbers, we know that the sample mean $\hat{\mu}_n = \frac{1}{n}\sum_{i=1}^n Y_i$ will converge towards $\mu = \Pr(A) = \frac{2L}{d} \cdot \frac{1}{\pi}$. Since we can record $\hat{\mu}_n$ in the experiment we can therefore use, for large n, the formula

$$\hat{\mu}_n \approx \frac{2L}{d} \cdot \frac{1}{\pi}$$

for estimating π. The value of π is therefore approximately $2L/d\hat{\mu}_n$.

The following matlab code gives an implementation of this direct Monte Carlo method.

```
L=1; d=2; nTrials=10000;
x=0.5*d*rand(1,nTrials);  %draws x
theta=0.5*pi*rand(1,nTrials); %draws theta
hits=x<=0.5*L*sin(theta); %1 if there is a hit, 0 otherwise
rel_freq=sum(hits)/nTrials %relative frequency
pi_est=nTrials*2*L/(sum(hits)*d) %estimate of pi
```

buffon1.m

An illustration is given in Fig. 9.

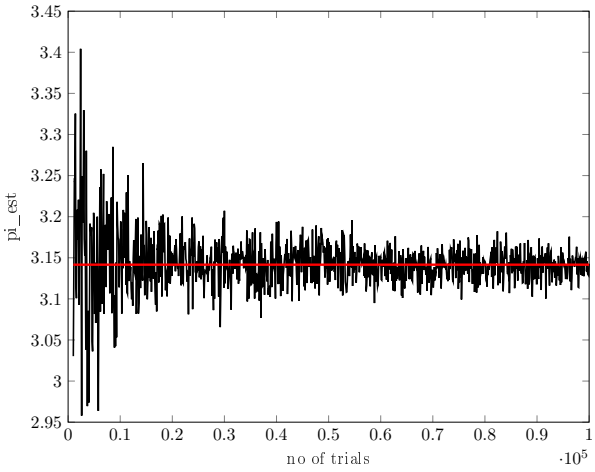


Figure 9: Simulating Buffon's experiment ($L = 1$, $d = 2$): A sample path of $\frac{2L}{d\hat{\mu}_n}$ (shown in black) approximating the value π (depicted in red).

We remark that there are faster ways of computing π. The formula

$$\frac{\pi}{4} = 4\arctan(1/5) - \arctan(1/239), \tag{2.4}$$

where $\arctan(x) = x - x^3/3 + x^5/5 - \cdots$, for $|x| < 1$, called Machin's[8] formula, remained the primary tool of π-hunters for centuries. Table 3 gives an indication of how quickly the approximations converge.

Nowadays there exist even faster methods for computing π. See, e.g., https://www.davidhbailey.com/dhbpapers/pi-quest.pdf.

2.2 Variance reduction techniques, importance sampling methods

We have seen that the RMSE in direct Monte Carlo for integration problems is of the order $O(\sigma/\sqrt{n})$. To increase the accuracy we can therefore increase n. However, sometimes there can be another way

[8] John Machin (bapt. c. 1686 - June 9, 1751) was a professor of astronomy at Gresham College, London.

n	estimate of π
1	3.183263598326360
2	3.140597029326060
3	3.141621029325035
4	3.141591772182177
5	3.141592682404399
6	3.141592652615309
7	3.141592653623555

Table 3: Estimating $\pi \approx 3.141592653589793$ by (2.4) using n terms in the evaluation of arctan by $\arctan(x) = x - x^3/3 + x^5/5 - \cdots$. Correct digits are shown in blue.

of achieving this. We may construct a new Monte Carlo problem with the same answer as the original one but with a smaller σ. Methods to do this are known as **variance reduction techniques**.

Out of the many variance reduction techniques from the literature (known under names such as *stratified sampling*, *control variates method*, *antithetic variates method*, and *Rao-Blackwellization*), we select here the concept of **importance sampling**.

Basic idea behind importance sampling The general idea behind importance sampling is to sample from a not necessarily uniform distribution, but rather from a distribution that somewhat resembles g (directing attention to important regions of the hypercube D, so to speak).

Suppose X is a random variable with pdf f_X such that $f_X(x) > 0$ on the set $\{x : g(x) \neq 0\}$ and $f_X(x) = 0$ for $x \notin D$. Let Y be the random variable $Y := g(X)/f_X(X)$. Then,

$$\mu := \int_D g(x)\,dx = \int_D \frac{g(x)}{f_X(x)} f_X(x)\,dx = \mathbb{E}_{f_X}[Y],$$

where, to emphasize that the pdf is f_X, we added the subscript f_X to $\mathbb{E}$.

Now, similarly as for direct Monte Carlo for uniformly distributed $X_1, \ldots, X_n$, we can obtain an estimate of $\mu = \mathbb{E}_{f_X}[Y]$ by computing

$$\hat{\mu}_n = \frac{1}{n}\sum_{i=1}^{n} Y_i = \frac{1}{n}\sum_{i=1}^{n} \frac{g(X_i)}{f_X(X_i)};$$

this time the $X_1, \ldots, X_n$ are generated from the distribution with pdf f_X. (One refers to f_X in this context also as **importance function**.) Of course, for this to work we need a method to sample from f_X. But assuming this can be achieved, we claim that for some importance functions it can happen that $\mathrm{var}_{f_X}(Y)$ can be smaller than the σ^2 that we have from the direct Monte Carlo method that uses no importance sampling.

Comparing

$$\begin{aligned}\mathrm{var}_{f_X}(Y) &= \mathbb{E}_{f_X}[Y^2] - \mathbb{E}_{f_X}[Y]^2 = \mathbb{E}_{f_X}[Y^2] - \mu^2 \\ &= \int_D \left(\frac{g(x)}{f_X(x)}\right)^2 f_X(x)\,dx - \mu^2 \\ &= \int_D \frac{g(x)^2}{f_X(x)}\,dx - \mu^2,\end{aligned}$$

with

$$\mathrm{var}(g(X)) = \int_D g(x)^2\,dx - \mu^2$$

from the direct Monte Carlo method using uniform samples, we see that f_X can indeed have an influence on the variance. (We can even see that $\mathrm{var}_{f_X}(Y)$ can be zero if we can take $f_X = g$. Besides

sampling from g this requires also that $g = f_X$ is normalized meaning that $\int_D f_X(x)\,dx = 1$, but then why would you want to compute the value μ of the integral if you know it will be 1?)

Importance sampling for computing integrals of the form (2.1):

(1) Choose a large n.

(2) Draw samples $X_1, \dots, X_n$ from a trial distribution with pdf f_X.

(3) Return

$$\hat{\mu}_n = \frac{1}{n}\sum_{i=1}^{n} \frac{g(X_i)}{f_X(X_i)}$$

as an estimate of the integral.

Example 2.2. *Suppose, we want to compute the following integral*

$$\mu = \int_0^1 \underbrace{\sqrt{1-x^2}}_{=:g}\, dx.$$

From basic calculus we find $\mu = [\frac{1}{2}(x\sqrt{1-x^2} + \arcsin(x))]_0^1 = \frac{1}{2}\arcsin(1) = \pi/4 \approx 0.78540$. *Let* X *be a random variable* $X \sim U(0,1)$. *Then,*

$$\mathrm{var}(g(X)) = \mathbb{E}[g(X)^2] - \mu^2 = 2/3 - \mu^2 = 2/3 - \pi^2/16 \approx 0.05.$$

Hence, the RMSE in direct Monte Carlo is

$$\sqrt{\mathrm{var}(\hat{\mu}_n)} = \frac{\sqrt{\mathrm{var}(g(X))}}{\sqrt{n}} \approx \frac{0.2236}{\sqrt{n}}.$$

Can we reduce this factor of 0.2236 *via importance sampling?*

It practice it can be quite tricky to find an importance function that does the job. For the sake of exposition, suppose we want to consider the function

$$f(x) := \frac{1-\alpha x^2}{1-\alpha/3}, \quad 0 < x < 1,$$

with $\alpha = 0.74$ *as importance function. Notice that, indeed,* $f(x) > 0$ *for* $x \in (0,1)$ *and*

$$\int_0^1 f(x)\,dx = \frac{1-\alpha/3}{1-\alpha/3} = 1.$$

We can therefore consider f *as the pdf* f_X *of a random variable* X.

Suppose, for the moment we can sample $X_1, \dots, X_n$ *from this distribution. Let* $Y := g(X)/f_X(X)$ *and* $Y_n := g(X_n)/f_X(X_n)$. *Then, again,* $\mathbb{E}_{f_X}[Y] = \mu$ *and*

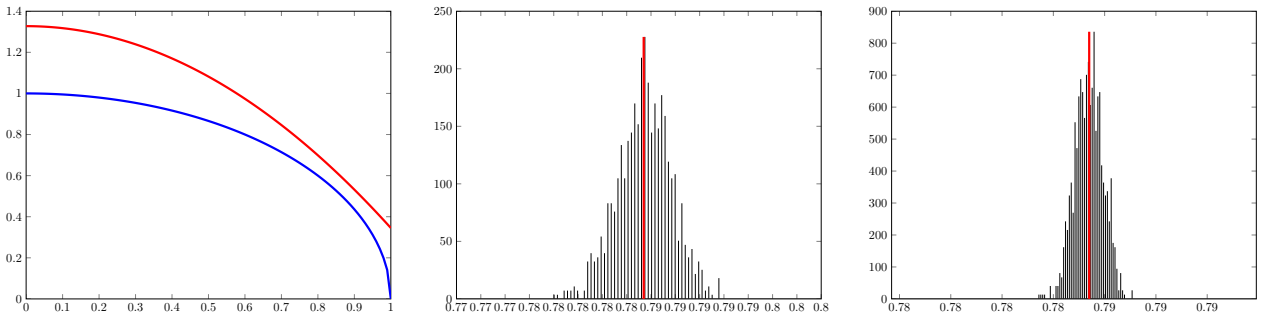
$$\mathrm{var}(Y) = \mathbb{E}_{f_X}\left[Y^2\right] - \mu^2. \tag{2.5}$$

Let's compute

$$\mathbb{E}_{f_X}[Y^2] - \mu^2 = (1-\alpha/3)^2 \cdot \int_0^1 \frac{(1-x^2)}{(1-\alpha x^2)^2}\frac{(1-\alpha x^2)}{(1-\alpha/3)}\,dx - \mu^2 \approx 0.0029,$$

and hence $\sqrt{\mathrm{var}(Y)} = \sqrt{0.0029} \approx 0.0539$. *In other words, the RMSE decreased in this case by importance sampling from* $0.2236/\sqrt{n}$ *to* $0.0539/\sqrt{n}$. *We can also see this effect in Fig. 10 (where samples from* f_X *are drawn by a method called rejection sampling, which we will discuss in later chapters).*

We have seen that in direct Monte Carlo we need random numbers. Until now, we have just used matlab's built-in functions. But how do they actually work? And how can we draw samples from more complicated distributions? The basics of generating pseudo random numbers are discussed next.



(a) (b) (c)

Figure 10: Illustration of Example 2.2: (a) Plots of g (blue), f (red); (b) Histogram of $\hat{\mu}_n$ (for $n = 10^4$ based on `nTrials` $= 10^3$ simulations) via direct Monte Carlo, sample mean μ shown in red; (c) same as in (b) but now importance sampling with density f was used.

2.3 Exercises

Exercise 2.1. Consider the random variable $Y := e^{-X}$, where X is a random variable uniformly distributed on the interval $(0, 1)$. Give precise values of $\mathbb{E}[Y]$ and $\mathrm{var}(Y)$.

Exercise 2.2. Consider the task of computing

$$\int_a^b 2x \, dx,$$

for a given interval $(a, b) \subseteq \mathbb{R}$.

(a) Devise a direct Monte Carlo method for approximating this integral.

(b) Give a precise value of $\mathbb{E}[g(X)]$ and $\mathrm{var}(g(X))$ for your method from (a).

Exercise 2.3. Consider the task of computing

$$\int_0^1 \frac{e^{-x}}{1+x^2} \, dx.$$

(a) Approximate the integral via matlab's `integrate` command.

(b) Approximate the integral via direct Monte Carlo. Give plots of $\hat{\mu}_n$ and the (theoretical) RMSE as a function of n.

Exercise 2.4. Consider the Buffon problem with a *long* needle, i.e., for $L \geq d$.

(a) Determine the probability that such a needle crosses at least one of the parallel lines.

(b) Convince yourself that, for $L = d$, your probability equals the one obtained for a short needle.

Exercise 2.5. Write a matlab simulation of the Buffon needle problem with a long needle. Assume the spacing between the horizontal lines is $d = 1$ and the length of the needle is $L = 2$. Give a plot of the relative frequency of line hits as a function of n.

Exercise 2.6. Provide matlab type pseudocode for approximating

$$a = \int_0^1 \sin^2(1/x) \, dx$$

by direct Monte Carlo based on $100,000$ samples.

Exercise 2.7. Instead of dropping a needle consider the problem of dropping a circle of radius r onto a plane covered with a set of parallel lines that are distance d apart. Assume $2r \leq d$.

(a) What is the probability that the circle crosses at least one of the parallel lines?

(b) By what factor is a straight needle of length $L \leq d$ more likely to cross one of the parallel lines than if it is bent into a circle?

Exercise 2.8. Suppose, we want to estimate

$$\int_0^1 g(x) \, dx = \int_0^1 \cos\left(\frac{\pi}{2}x\right) \, dx.$$

(a) Express the integral in terms of the expectation of a random variable.

(b) To apply importance sampling, we approximate g by a second degree polynomial. Since g is even and equals zero at $x = 1$ and one at $x = 0$, it is natural to take the importance function f of the form $f(x) = \lambda(1 - x^2)$. Determine the value λ so that the constraint $\int_0^1 f(x)\,dx = 1$ is satisfied.

(c) Calculate the variance of $g(X)/f(X)$ and show that we have reduced the variance by a factor of 100.

Exercise 2.9. The (in-)famous *Monty Hall problem* is the following. A car is randomly placed behind one of the three doors. The contestant randomly chooses one of the doors. If the contestant chooses the door with the car, the host randomly chooses one of the remaining doors to open and then gives the contestant a chance to either stick with the original choice or switch. On the other hand, if the contestant chooses a door with a goat, the host opens the other door with a goat and again gives the contestant a chance to either stick with the original choice or switch.

Simulate this version of the game to obtain an estimate of the winning probability if the contestant sticks with the original choice and also if the contestant switches.

(a) Provide matlab code for your simulation.

(b) Your estimate of the winning probability if the contestant sticks with the original choice is

$p_{stick} =$

(c) Your estimate of the winning probability if the contestant switches is

$p_{switch} =$

3 Pseudo random number generator, simulation methods for univ. distr.

Random numbers can be generated from truly random physical processes (radioactive decay, thermal noise, roulette wheel). The RAND cooperation, for instance, published in 1955 a book with a million random numbers, obtained using an electric 'roulette wheel.' This book is now publicly available on http://www.rand.org/publications/classics/randomdigits/.

However, physical random numbers are generally not very useful for Monte Carlo, because the sequence is not repeatable, the generators are often slow, and it can be complicated to feed these random numbers into the computer.

We therefore need a way of generating 'random' numbers by a computer. These numbers are often called **pseudo random numbers** to stress the fact that they are not really random as they are generated deterministically. In the following we will usually omit the word 'pseudo' since we discuss only these random numbers anyway.

3.1 Sampling from a uniform univariate distribution

Middle-square and other middle-digit techniques One of the earliest recorded methods is the *middle-square* method by John von Neumann (1946). The idea is the following.

> To generate a sequence of n-digit random numbers, an n-digit starting value is created and squared, producing a $2n$-digit number. If the result has fewer than $2n$ digits, leading zeroes are added to compensate. The middle n digits of the result would be the next number in the sequence, and returned as the result. This process is then repeated to generate more numbers.

Example 3.1. *Suppose, we want to generate 4-digit (integer) numbers, i.e., $n = 4$. Let's take as initial seed $v_0 = 1234$. Then, we obtain:*

$$\begin{aligned} v_0 &= \underline{1234} \overset{\text{squaring}}{\longrightarrow} 01\mathbf{5227}56 \overset{\text{extract}}{\longrightarrow} 5227, \\ v_1 &= \underline{5227} \overset{\text{squaring}}{\longrightarrow} 27\mathbf{3215}29 \overset{\text{extract}}{\longrightarrow} 3215, \\ v_2 &= \underline{3215} \overset{\text{squaring}}{\longrightarrow} 10\mathbf{3362}25 \overset{\text{extract}}{\longrightarrow} 3362, \\ &\dots \end{aligned}$$

The following gives a MATLAB implementation of this approach.

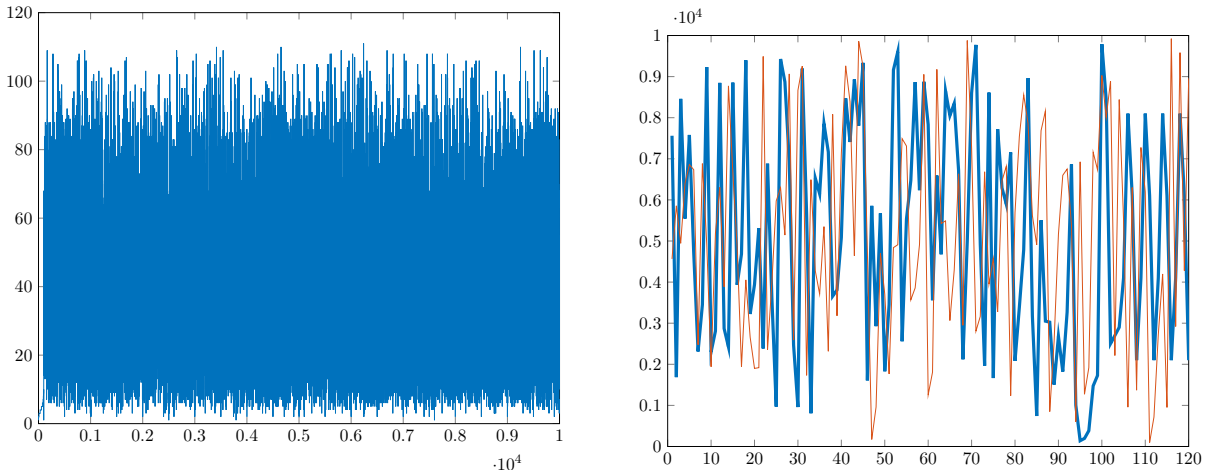
```
function [z] = vonNeumannMiddleSquare(nnumbers,ndigits,seed)
%::: seed must be an ndigit natural number
%::: returned numbers z(1),z(2),..z(nnumbers) are ndigit-digit integer numbers
  fstring=sprintf('%%0%d.f',2*ndigits); n2=ndigits/2;

  x(1)=seed;
  for i=1:nnumbers
      x(i+1)=x(i)^2; %square the number
      s=num2str(x(i+1),fstring); %add leading zeros is neccessary
      x(i+1)=str2num(s(n2+1:end-n2)); %extract middle n digits
  end;
  z=x(2:end);
end
```

vonNeumannMiddleSquare.m

Fig. 11 shows some results for $n = 4$. As initial seed $v_0 = 5810$ is used since it gives a rather long chain of non-repeating numbers. After the 108th generated number, the numbers repeat (actually, with a fairly short cycle of 4100, 8100, 6100, 2100, 4100, ...). By the way, there a five numbers, namely 0, 100, 2500 3792, and 7600, which, if taken as seed, generate no further numbers. Also interesting to note is that the longest runner up is 8 while the longest runner down is 11.

Let us prove a fact about the middle-square method, which is rather undesirable.

Theorem 3.1 ([6]). *Consider the middle-square method for generating n-digit numbers, n even. If, for some k, the most significant $n/2$ digits of v_k are zero, then the succeeding $v_{k+1}, v_{k+2}, \dots$ will get smaller and smaller until zero occurs repeatedly.*



(a) (b)

Figure 11: Middle-square method for $n = 4$. (a) Plot of the number of generated numbers to first repetition as a function of seed (average: 43.7, maximum: 111), (b) sample path for seed $v_0 = 5810$ (blue), randomly drawn integers by matlab's `randi` for comparison (red).

Proof. Since $v_k < 10^{n/2}$ we have $v_k^2 < 10^n$. Therefore, $v_{k+1} = \lfloor v_k^2/10^{n/2} \rfloor \leq v_k^2/10^{n/2}$ (note that v_k^2 has $2n$ digits including leading zeros). If $v_k > 0$, then $v_{k+1} \leq v_k^2/10^{n/2} < v_k 10^{n/2}/10^{n/2} = v_k$. □

As an illustration consider the middle-square method for generating $n = 4$ digit numbers (note: you always needs to specify n). Starting with $v_0 = 0099$ we obtain the following strictly decreasing sequence 0098, 0096, 0092, 0084, 0070, 0049, 0024, 0005, 0000.

The middle-square method was invented at a time when computers were just starting. At that time this method was extensively used since it was simpler and faster than any other method. However, by current standards, its quality is quite poor, at least when applied to numbers in the decimal number system. Donald Knuth ([6]) remarks: *'[...] experience showed that the middle-square method can give usable results, but it is rather dangerous to put much faith in it until after elaborate computations have been performed.'*

We remark that instead of squaring number (which is computationally usually a fast thing to perform) one could modify the middle-square method to utilize other functions, e.g., sin or log . Sometimes one needs to pay special attention to make correct use of the domains and ranges of these functions. We do not pursue this here any further. Generally, all these methods tend to suffer from fundamental problems, one of which is a short period and rapid cycling for most initial seeds.

Linear congruential random number generators (LCRNGs) Most random-number generators in use today are linear congruential random number generators (LCRNGs).

They produce a sequence of integers between 0 and $m-1$ according

$$v_{k+1} = (av_k + c) \bmod m, \qquad k = 0, 1, 2, \ldots \tag{3.1}$$

where a is called the *multiplier*, c the *increment*, and m the *modulus*. (n mod m is the difference of n and the largest integer multiple of m less or equal to n.) The initial number v_0, which usually needs to be provided by the user, is typically referred to as *seed*.

Note that the generated numbers are typically elements in $\{0, 1, \ldots, m-1\}$. By dividing by m, i.e., by setting

$$u_k = \frac{v_k}{m}$$

the numbers can be transformed to be real numbers in $[0, 1)$.

Sequences of numbers $v_0, v_1, \ldots$ generated by (3.1) are called *Lehmer*[9] *sequences.*

A frequently recommended generator of this form, due to M. Marsaglia (1972), is:

$$v_{k+1} = (69069 v_k + 1) \bmod 2^{32}. \tag{3.2}$$

This generator has in fact a *maximal period*, i.e., the numbers repeat after m numbers have been generated.

A large class of LCRNGs are the so-called *Mersenne*[10] *generators.* They use Mersenne prime moduli, i.e., m is a prime of the form $m = 2^p - 1$, where p is also a prime. These generators can have periods of length $m - 1$, and it can also be shown that they never generate a zero. Coincidentally, $m = 2^{31} - 1 = 2,147,483,647$ is a Mersenne prime, the largest prime number of this form that can be stored in a full-word (32-bit)integer computer.

The following Lewis-Goodman-Miller generator, introduced in 1969, is a widely studied Mersenne generator:

$$v_{k+1} = 7^5 v_k \bmod (2^{31} - 1).$$

Problems with number generators Let us have a look at `RANDU`, which is a LCRNG developed by IBM in the 1950's, and which for many years was the most widely used random number generator in the world. Leaving some minor technical details aside, the numbers are generated as

$$v_{k+1} = (2^{16} + 3) v_k \bmod 2^{31}. \tag{3.3}$$

The generator in (3.3) can be written to express the relationship among three successive members of the output sequence:

$$\begin{aligned} v_k &\equiv (2^{16} + 3) v_{k-1} \bmod 2^{31} \\ &\equiv (2^{16} + 3)^2 v_{k-2} \bmod 2^{31} \\ &\equiv (2^{32} + 6 \cdot 2^{16} + 9) v_{k-2} \bmod 2^{31} \\ &\equiv (6 \cdot (2^{16} + 3) - 6 \cdot 3 + 9) v_{k-2} \bmod 2^{31} \\ &\equiv (6 \cdot (2^{16} + 3) v_{k-2} - 9 v_{k-2}) \bmod 2^{31} \\ &\equiv (6 v_{k-1} - 9 v_{k-2}) \bmod 2^{31}, \end{aligned}$$

i.e.,

$$v_k - 6 v_{k-1} + 9 v_{k-2} = C 2^{31},$$

where C is an integer. Note that $0 \le v_k < 2^{31}$ by construction. Hence, the maximum integer that we can obtain by $v_k - 6v_{k-1} + 9v_{k-2}$ is smaller than $2^{31} - 0 + 9 \cdot 2^{31} = 10 \cdot 2^{31}$. Similarly, the smallest number that we can obtain is larger than $0 - 6 \cdot 2^{31} + 0 = -6 \cdot 2^{31}$. This leaves only the 15 possibilities $C \in \{-5, -4, \ldots, 8, 9\}$. Consequently, all triples must lie on no more than 15 hyperplanes in $\mathbb{R}^3$.

Fig. 12 shows a plot of subsequent triples of generated numbers viewed from different perspective. The 15 hyperplanes are clearly visible in Fig. 12(b).

3.2 Analyzing random number generators

Of course, we would like our random numbers to be as random as possible. This, however, cannot be checked. We can only detect non-randomness. Over the years, there have been developed many different statistical tests for detecting nonrandomness. Typical random numbers need to pass all those tests. For lack of space, we are not going into details here. The interested reader can find information about the so-called *chi-square* and the *Kolmogorov-Smirnov* test in [7].

A quick indication whether the generated numbers behave somewhat random can be found by the following procedures.

(a) Plot a *sample path,* i.e., if r_i is the ith random number generated, plot r_i versus i.

[9] Derrick Lehmer (23. Feb. 1905 - 22. May 1991)
[10] Marin Mersenne (8 Sept. 1588 - 1 Sept. 1648)

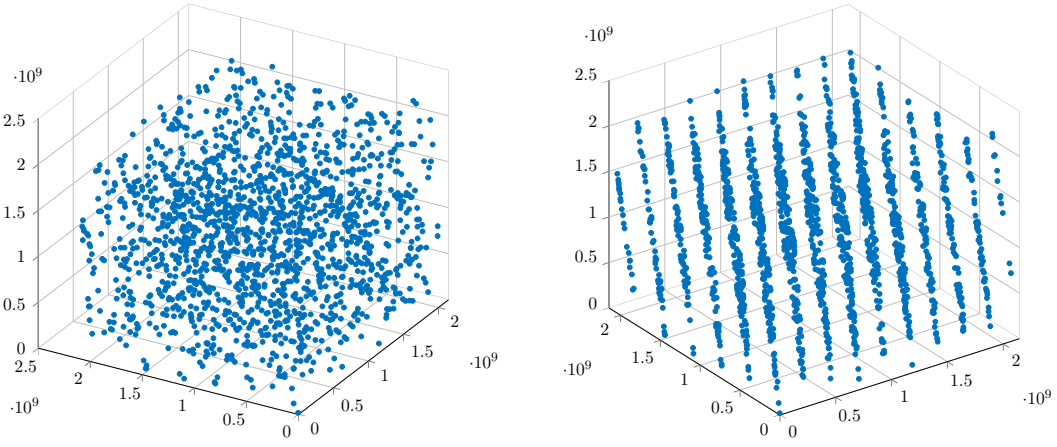


Figure 12: Subsequent triples of IBM's `RANDU` viewed from two perspectives.

(b) Plot a *histogram*.

(c) Plot a *correlation plot*, i.e., plot (r_i, r_{i+1}) (or, (r_i, r_{i+1}, r_{i+2}) or even larger tuples).

3.3 General methods for sampling from non-uniform distributions

3.3.1 Cdf inversion

This method goes back to the beginnings of Monte Carlo. It was proposed by John von Neumann[11] in a letter to Stanislav Ulam[12] discussing their 'random numbers work'[13].

Let us first discuss the discrete case.

Theorem 3.2. *Let $U \sim U(0,1)$, and let Y be a discrete random variable with cdf F. Let*

$$F^-(u) = \min\{t \in \mathbb{R} : F(t) \geq u\}$$

be the so-called generalized inverse of F. Then, the discrete random variable $X = F^-(U)$ has cdf F.

Proof. Let us first convince ourselves that the minimum in $\min\{t \in \mathbb{R} : F(t) \geq u\}$ is in fact attained.

Consider for a given $u \in (0,1)$ the set $I_u = \{t \in \mathbb{R} : F(t) \geq u\}$. Note that I_u is non-empty, since $u < 1$ and $F(y) \to 1$ as $y \to \infty$. I_u has a finite left endpoint, say η_u, because $u > 0$ and $F(y) \to 0$ as $y \to -\infty$. Finally, $\eta_u \in I_u$, since F is a cdf and therefore right-continuous (consider $y_n = \eta_u + 1/n$, $n = 1, 2, \ldots$. Then, $u \leq F(y_n)$ for all n, hence $u \leq \lim_n F(y_n) = F(\eta_u)$ implying $\eta_u \in I_u$). In summary, the minimum in $\min\{t \in \mathbb{R} : F(t) \geq u\}$ is indeed attained.

We claim

$$\{(t,u) \in \mathbb{R} \times (0,1) : F^-(u) \leq t\} = \{(t,u) \in \mathbb{R} \times (0,1) : u \leq F(t)\}. \tag{3.4}$$

Taking an element from the left set, i.e., $(t,u) \in \mathbb{R} \times (0,1)$ satisfying $F^-(u) \leq t$, we have

$$F(t) \overset{\text{non-decr.}}{\geq} F(F^-(u)) \overset{\text{def.}}{=} F(\min\{t \in \mathbb{R} : F(t) \geq u\}) \overset{\text{def.}}{\geq} u$$

and (t,u) is contained in the right set. Conversely, for $(t,u) \in \mathbb{R} \times (0,1)$ satisfying $u \leq F(t)$ we have

$$F^-(u) \overset{\text{non-decr.}}{\leq} F^-(F(t)) \overset{\text{def.}}{=} \min\{r \in \mathbb{R} : F(r) \geq F(t)\} \overset{t \text{ in set}}{\leq} t,$$

proving (3.4).

[11] Dec. 28, 1903 - Feb. 8, 1957

[12] 13 April 1909 - 13 May 1984

[13] See R. Eckhardt. Stan Ulam, John von Neumann and the Monte Carlo method. Los Alamos Science, pages 131–143, 1987. Special Issue. http://www-star.st-and.ac.uk/~kw25/teaching/mcrt/MC_history_3.pdf

Now we can complete the proof:

$$\Pr(X \le t) \overset{\text{def.}}{=} \Pr(F^-(U) \le t) \overset{(3.4)}{=} \Pr(U \le F(t)) \overset{\text{def.}}{=} F_U(F(t)) \overset{(1.1)}{=} F(t).$$

□

Now let us take a closer look at Theorem 3.2. In fact, the theorem gives us a method of sampling from a distribution with cdf F. All we need to do is generate $U \sim U(0,1)$ and compute the generalized inverse $F^-(U)$. But how do we compute $F^-(U)$?

For notation, suppose Y (and also the to-be generated X) takes values $x_1, \ldots, x_N$ with probabilities $p_1, \ldots, p_N$. Then,

$$F(t) = \sum_{k\,:\,x_k \le t} p_k,$$

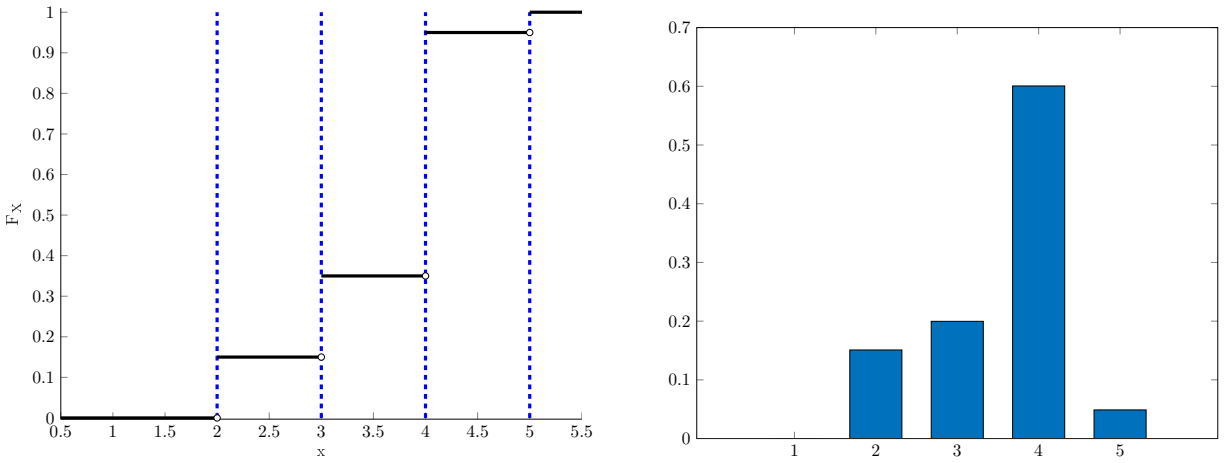
with the convention that the empty sum yields value 0. For $u \in (0,1)$, we therefore have $F^-(u) = x_k$ if and only if $F(x_{k-1}) < u \le F(x_k)$.

Hence ***cdf inversion for discrete random variables*** reduces to the following:

(a) Generate $U \sim U(0,1)$.

(b) Set $X = x_k$ if $F(x_{k-1}) < U \le F(x_k)$.

Example 3.2. *(From [7]) Suppose , we want to simulate the number of eggs laid by a shorebird during breeding season. These birds have $X = 2, 3, 4,$ or rarely 5 eggs per clutch. Suppose, the observed frequencies of these sizes are $p_2 = 0.15$, $p_3 = 0.20$, $p_4 = 0.60$, and $p_5 = 0.05$, respectively. The cdf of X is shown in Fig. 13(a).*



(a)

(b)

Figure 13: (a) Cdf for Example 3.2, (b) Histogram for 10,000 samples.

Let $U \sim U(0,1)$ a uniform sample. Starting from the point $(0, U)$ (on the y-axis), proceed to the right until you encounter a jump in the cdf, a vertical dashed line segment. Now, proceed down to the x-axis, and return this value as the selection. As indicated along the y-axis, '2' is selected with probability 0.15, since its interval occupies this fraction of the unit interval. Similarly, '3' is selected with 0.2 probability, since its interval occupies this fraction of the unit interval, and so on.

Example Matlab code for generating samples from this distribution is given below. A histogram obtained by executing this code is shown in Fig. 13(b).

```
nTrials=10^4;
b(2)=0.15; b(3)=b(2)+0.2; b(4)=b(3)+0.6;
U=rand(1,nTrials);
w2=(U<=b(2))*2;
w3=(U>b(2) & U<=b(3))*3;
w4=(U>b(3) & U<=b(4))*4;
w5=(U>b(4))*5;

w=w2+w3+w4+w5; %vector with the right frequencies

[f, x] = hist(w,1:5); dx = diff(x(1:2)); bar(x, f / sum(f * dx));  %plots the histogram
```

cdfinvdiscrexample1.m

Example 3.3. *Suppose X is geometric with parameter p such that $\Pr(X = k) = (1-p)^{k-1}p$, for $k \in \{1, 2, \dots\}$. Then,*

$$\begin{aligned} F_X(t) &= \sum_{k=1}^{\lfloor t \rfloor} (1-p)^{k-1}p \\ &= p\left(1 + (1-p) + (1-p)^2 + \cdots + (1-p)^{\lfloor t \rfloor - 1}\right) \\ &= p \cdot \frac{1-(1-p)^{\lfloor t \rfloor}}{1-(1-p)} \\ &= 1 - (1-p)^{\lfloor t \rfloor}. \end{aligned}$$

Therefore, by cdf inversion we can generate X as follows:

(a) Generate $U \sim U(0,1)$.

(b) Set $X = k$ if $1-(1-p)^{k-1} < U \leq 1-(1-p)^k$.

It turns out that, for this distribution, we can compute step (b) more efficiently. In fact, step (b) can be replaced by

(b') Set $X = \lfloor \ln(U)/\ln(1-p) \rfloor + 1$.

To verify this note that step (b) can be reformulated as saying that we are looking for:

$$\begin{aligned} k &= \min\{k \in \mathbb{N} : U \leq 1-(1-p)^k\} \\ &= \min\{k \in \mathbb{N} : (1-p)^k \leq 1-U\} \\ &\overset{(*_1)}{=} \min\{k \in \mathbb{N} : (1-p) \leq (1-U)^{1/k}\} \\ &\overset{(*_2)}{=} \min\{k \in \mathbb{N} : \ln(1-p) \leq \frac{1}{k}\ln(1-U)\} \\ &\overset{(*_3)}{=} \min\{k \in \mathbb{N} : k \geq \frac{\ln(1-U)}{\ln(1-p)}\} \\ &\overset{(*_4)}{=} \min\left\{k \in \mathbb{N} : k \geq \left\lceil \frac{\ln(1-U)}{\ln(1-p)} \right\rceil\right\} \\ &= \left\lceil \frac{\ln(1-U)}{\ln(1-p)} \right\rceil, \end{aligned}$$

where $(*_1)$ and $(*_2)$ follow from the fact that $\log$ and, respectively, $(\cdot)^{1/k}$ are non-decreasing, $(*_3)$ follows from the fact that $\ln(1-p)$ is negative, and $(*_4)$ follows from the fact that k is an integer.

A histogram of $10,000$ samples from the geometric distribution with $p = 0.2$ is shown in Fig. 14.

Let us now turn to the continuous case.

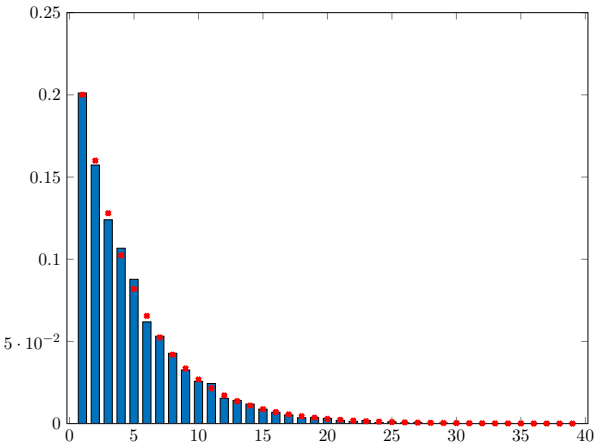


Figure 14: Histogram (obtained by cdf inversion) of the geometric distribution, $p = 0.2$. Graph of $f(k) = (1-p)^{k-1}p$ is plotted in red.

Theorem 3.3. *Let $U \sim U(0,1)$, and let Y be a continuous random variable with invertible cdf F. Then, the continuous random variable $X = F^{-1}(U)$ has cdf F.*

Proof. Let $u \in (0,1)$ and $I_u = \{t \in \mathbb{R} : F(t) \geq u\}$. As we assume that F is invertible, we have $F(F^{-1}(u)) = u \geq u$, i.e., $F^{-1}(u) \in I_u$. And there is no smaller element than $F^{-1}(u)$ contained in I_u since an invertible cdf, such as F, is strictly increasing. Hence,

$$F^{-}(u) = F^{-1}(u), \qquad \text{for all } u \in (0,1).$$

With this the rest of the proof from Theorem 3.2 carries over, word-by-word. □

Hence ***cdf inversion for continuous random variables*** is as follows:

(a) Generate $U \sim U(0,1)$.

(b) Return $F^{-1}(U)$.

Note that for cdf inversion an explicit formula for F^{-1} is needed. Such a formula is often not available (e.g., in the case of a normal distribution). However, there are cases where cdf inversion can be applied successfully. Let us look at two examples.

Example 3.4. *See Exercise 18.*

Example 3.5 (Exponential distribution)**.** *Consider the exponential distribution $f(t) = \lambda e^{-\lambda t}$ with parameter $\lambda > 0$. Its cdf is $F(t) = 1 - e^{-\lambda t}$, $t \in [0, \infty)$. This is a continuous, strictly increasing function, which is therefore invertible. Solving $u = 1 - e^{-\lambda t}$ for t we see that $F^{-1}(u) = -\ln(1-u)/\lambda$. Hence, for $U \sim U(0,1)$ the variable*

$$T = -\frac{1}{\lambda}\ln(1-U)$$

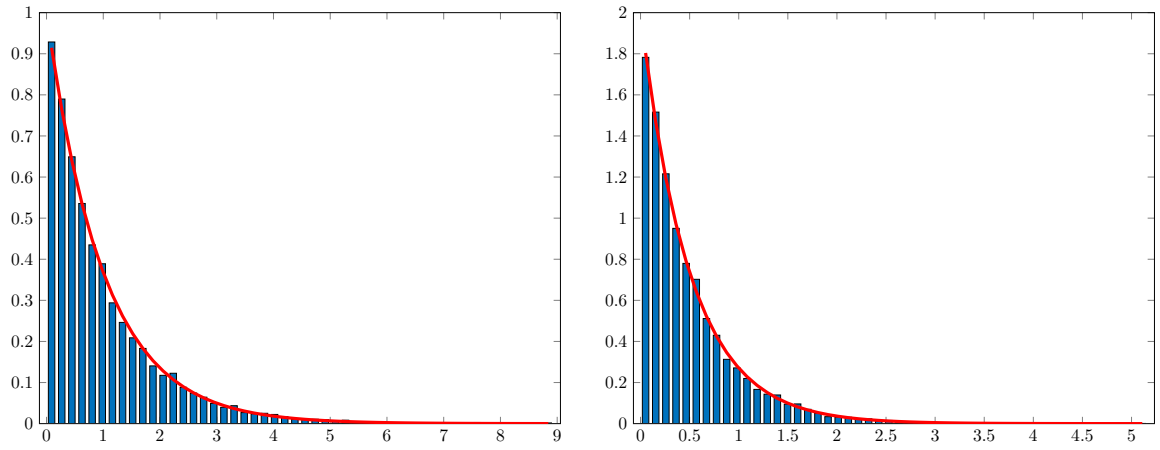
is the desired variable. Fig. 15 shows a histogram obtained by sampling $10,000$ times with this method.

Let us summarize. Cdf inversion for discrete random variables is a completely general method. It can be applied to any discrete probability distribution. The continuous version, however, only applies if an explicit formula for the inverse of the cdf is available.

3.3.2 Rejection sampling

Rejection sampling[14] is a general method for sampling from both discrete and continuous distributions.

[14] Again going back to work of John von Neumann. See R. Eckhardt. Stan Ulam, John von Neumann and the Monte Carlo method. Los Alamos Science, pages 131–143, 1987. Special Issue. http://www-star.st-and.ac.uk/~kw25/teaching/mcrt/MC_history_3.pdf



(a) (b)

Figure 15: Sampling from the exponential distribution via cdf inversion. Histogram of $10,000$ samples. The pdf $F'(t) = \lambda e^{-\lambda t}$ of the exponential function is plotted in red. (a) $\lambda = 1$, (b) $\lambda = 2$.

The general idea is to uniformly sample from a so-called *trial distribution* and then accept or reject them according to a criterion designed so that overall the returned outputs follow the correct distribution (sometimes called *target distribution*).

Let f denote the pdf from which we want to sample. Let g and M denote another pdf and constant, respectively, such that $f(x) \leq Mg(x)$ holds for all x.

Rejection sampling proceeds as follows:

(a) Generate Y having density $g(Y)$.

(b) Generate $U \sim U(0,1)$, independent of Y.

(c) If $U \leq f(Y)/Mg(Y)$, then return $X = Y$ ('accept Y'); otherwise ('reject') go back to (a).

To see that rejection sampling really works (i.e., that X is really a random variable with pdf f), we will need to look at the continuous and discrete distribution function case separately.

Common to both cases is that each time we loop through (a)-(c) a pair (Y, U) is generated. To be accepted the pair needs to satisfy $U \leq f(Y)/Mg(Y)$. Hence, any X returned as output has the same distribution as $(Y|U \leq f(Y)/Mg(Y))$; i.e., the conditional distribution of Y given that Y is accepted.

Proof for the continuous case

$$\Pr(X \leq x) = \Pr(Y \leq x | U \leq f(Y)/Mg(Y)) = \frac{\Pr(Y \leq x, U \leq f(Y)/Mg(Y))}{\Pr(U \leq f(Y)/Mg(Y))}.$$

Now, for the numerator of the above expression we have

$$\begin{aligned}
\Pr\left(Y \leq x, U \leq \frac{f(Y)}{Mg(Y)}\right) &= \int_{-\infty}^{\infty} \Pr\left((Y \leq x, U \leq \frac{f(Y)}{Mg(Y)}) | Y = y\right) g(y)dy \quad (\text{by}^{15}) \\
&= \int_{-\infty}^{x} \Pr\left(U \leq \frac{f(y)}{Mg(y)}\right) g(y)dy \\
&= \int_{-\infty}^{x} \frac{f(y)}{Mg(y)} g(y)dy \\
&= \frac{1}{M} \int_{-\infty}^{x} f(y)dy \\
&= \frac{1}{M} F(x),
\end{aligned}$$

where F is the cdf corresponding to f. For the denominator we obtain (similar to the first line of the above equations)

$$\Pr\left(U \leq \frac{f(Y)}{Mg(Y)}\right) = \int_{-\infty}^{\infty} \frac{f(y)}{Mg(y)} g(y) dy = \frac{1}{M}.$$

Hence $\Pr(X \leq x) = F(x)$, as required.

Proof for the discrete case

$$\Pr(X = x) = \Pr(Y = x | U \leq f(Y)/Mg(Y)) = \frac{\Pr(Y = x, U \leq f(Y)/Mg(Y))}{\Pr(U \leq f(Y)/Mg(Y))}.$$

Before we consider the numerator and denominator of the above expression, note that

$$\Pr(\text{accept}|Y = x) = \frac{f(x)}{Mg(x)} \qquad \text{(basic uniform cdf)} \tag{3.5}$$

Now,

$$\begin{aligned}
\Pr\left(Y = x, U \leq \frac{f(Y)}{Mg(Y)}\right) &= \Pr(\text{accept}|Y = x)\Pr(Y = x) \qquad \text{(by Bayes' rule)} \\
&= \Pr(\text{accept}|Y = x) g(x) \\
&= \frac{f(x)g(x)}{Mg(x)} \qquad \text{(using (3.5))} \\
&= \frac{1}{M} f(x).
\end{aligned}$$

Further,

$$\begin{aligned}
\Pr\left(U \leq \frac{f(Y)}{Mg(Y)}\right) &= \Pr(\text{accept}) \\
&= \sum_x \Pr(\text{accept}|Y = x)\Pr(Y = x) \qquad \text{(by the law of total probability}^{16}\text{)} \\
&= \sum_x \frac{f(x)}{Mg(x)} g(x) \qquad \text{(using (3.5))} \\
&= \frac{1}{M} \sum_x f(x) \\
&= \frac{1}{M} \qquad \text{(since } f \text{ is a pdf).}
\end{aligned}$$

Hence $\Pr(X = x) = f(x)$, as required.

Comments

(a) When we compute the acceptance rate, we divide by $g(y)$. Hence one should choose g such that $g(y) > 0$ for all y.

(b) Above we have calculated the acceptance rate of (Y, U), which is

$$\Pr\left(U \leq \frac{f(Y)}{Mg(Y)}\right) = 1/M.$$

[15] The continuous law of total probability states: If $A \subseteq \mathbb{R}^2$ is any event then

$$\Pr((U, Y) \in A) = \int_{-\infty}^{\infty} \Pr((U, Y) \in A | Y = y) f_Y(y)\, dy.$$

[16] If A is any event and $B_1, B_2, \ldots$ form a partition of Ω, then $\Pr(A) = \sum_i \Pr(A|B_i)\Pr(B_i)$.

This means that we will have to generate, on average[17], Mn draws from both the trial and uniform distribution to obtain n draws from the target distribution. Thus, generally, M should not be chosen too large — optimally as small as possible. Also the trial distribution function g should ideally be chosen to follow closely f, as this will make the acceptance probability larger.

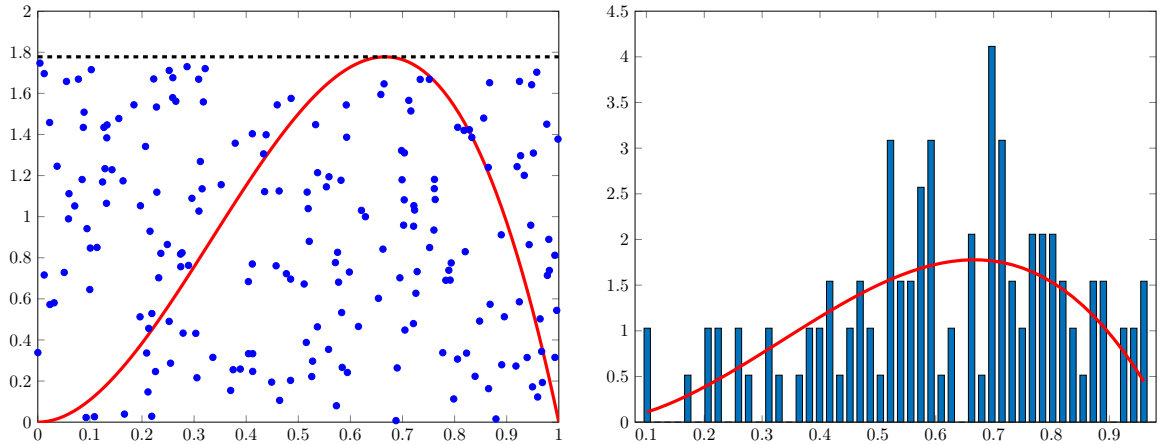
(c) We only need to know f and g up to a constant of proportionality. In many applications we will not know the normalizing constant for these densities, but we do not need them. That is, if $f^*(y) = c_f f(y)$ and $g^*(y) = c_g g(y)$, for all y, we can proceed with the algorithm using f^* and g^* even if we do not know the values of c_f and c_g.

Example 3.6 (Continuous distribution). *Let us consider the case, where the target pdf is*

$$f(x) = \begin{cases} 12x^2(1-x) & : \ x \in (0,1), \\ 0 & : \ \text{otherwise.} \end{cases}$$

(This is the so-called Beta distribution with parameters $\alpha = 3$ and $\beta = 2$.) Since the maximum of f occurs at $x_0 = 2/3$, where $f(x_0) = 16/9$, this means we can take $M = 16/9$, for $x \in [0,1]$, corresponding to a uniform density g over $(0,1)$.

In Fig. 16(a) we show f, M, and 200 points corresponding to trials $(Y, U \cdot Mg(Y))$. When the second coordinate $U \cdot Mg(Y)$ is smaller or equal to $f(Y)$, the point is accepted; otherwise it is rejected. For this particular sample, 111 points were accepted and 89 were rejected for a proportion $111/200 = 0.5550$ of acceptance, not too far from the theoretical one of $1/M = 9/16 = 0.5625$.



(a) (b)

Figure 16: Sampling from $f(x) = 12x^2(1-x)$ via rejection sampling. (a) plot of $(Y, U \cdot Mg(Y))$, i.e., the first component of any blue point under the red curve gives one sample of X, (b) histogram of the 111 samples from X.

Example 3.7 (Discrete distribution). *Let us assume we toss two coins (with H denoting 'heads' and T denoting 'tails'). The possible events (numbered for convenience with numbers from 1 to 4) are (1) HH, (2) HT, (3) TH, and (4) TT. We now want to select the event HH with probability $2/5$, and the other events each with probability $1/5$. How can we do this with rejection sampling?*

As trial distribution we can take the uniform distribution on the pair of coin tosses (which, of course has 4 outcomes), hence $Pr(\text{outcome is 'i'}) = g(i) = 1/4$, for any $i = 1, \ldots, 4$. Note our target distribution is $f(1) = 2/5$, $f(2) = f(3) = f(4) = 1/5$. Since we require $f(i) \leq Mg(i)$, we need $M \geq 4 \cdot 2/5 = 8/5$. Let's just choose the smallest value $M = 8/5$. The rejection method becomes:

(a) Generate $Y \sim U(\{1,2,3,4\})$.

(b) Generate $U \sim U(0,1)$, independent of Y.

[17] Note that the probability of an 'accept' after k Bernoulli trials follows a geometric distribution $p(1-p)^{k-1}$, where $p = 1/M$ is the probability of 'accept.' The mean of this distribution is M.

(c) If

$$U \leq f(Y)/Mg(Y) = \begin{cases} \frac{2/5}{2/5} = 1 & : \quad Y = 1 \\ \frac{1/5}{2/5} = 1/2 & : \quad Y = 2, 3, 4 \end{cases}$$

then return $X = Y$ ('accept Y'); otherwise ('rejection') go back to (a).

In other words, in step (c) we accept always $Y = 1$; in the other three cases we accept Y only with probability 1/2. Figure 17 shows a histogram obtained by this method.

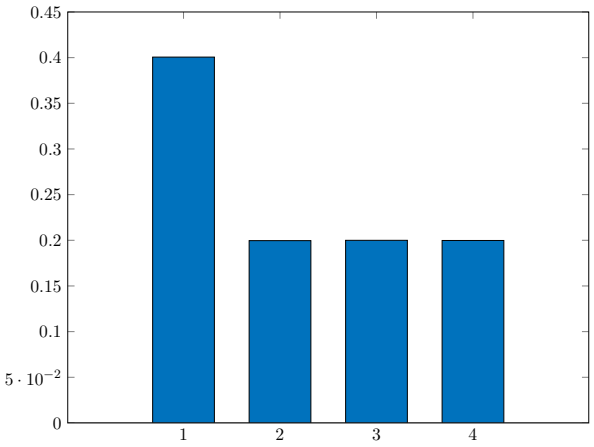


Figure 17: Rejection sampling from the discrete distribution of Example 3.7.

3.4 Sampling from a normal distribution

Suppose, now we want to sample from the normal distribution. We mentioned already that cdf inversion cannot (at least directly) be applied to sample from the normal distribution, because there is no close-form formula for the cdf of the normal distribution. So, what else can we do?

Well, what about rejection sampling? The answer is YES this can work. For instance, one can choose as trial distribution the exponential distribution (and use the fact that the normal distribution is symmetric) and then apply rejection sampling (for sampling from the exponential distribution, see Example 3.5).

One might also want to try to use as trial distribution the uniform distribution. There, however, we run into trouble, because that distribution does not exist (the interval from which we want to sample is unbounded). A way out of this is to approximate the normal distribution by a truncated normal distribution (hence, bounding the interval) and then use rejection sampling. Fig. 18 shows an example. The downside of this is, of course, that these random numbers are just approximations of the normal distribution.

Now, let us look at more efficient methods.

We give three methods for generating random variables $X \sim N(0, 1)$. Random variables $Y \sim N(\mu, \sigma^2)$ for the more general case can be obtained subsequently by setting

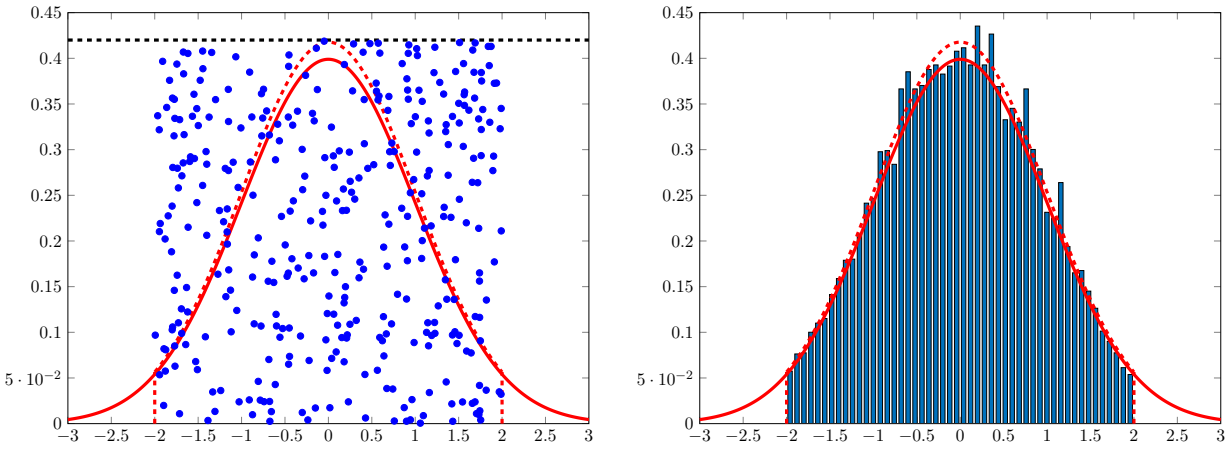
$$Y = \sigma X + \mu;$$

see Thm. 1.1.

The first method is an 'approximative method' in the sense that the samples follow an approximate normal distribution as a limit process is involved.

3.4.1 Central Limit algorithm

The method is as follows:



(a) (b)

Figure 18: Sampling 200 samples from a truncated normal distribution (red, dashed) via rejection sampling using as trial distribution a uniform distribution (black, dashed). Normal distribution shown as red solid curve. (a) plot of $(Y, U \cdot Mg(Y))$, (b) histogram.

Sample $U_1, \ldots, U_n \sim U(1,0)$ and return

$$X = \frac{\sum_{i=1}^{n} U_i - n/2}{\sqrt{n/12}}.$$

Recall that the mean and the variance of a $U(0,1)$ random variable are $1/2$ and $1/12$. Hence, by the central limit theorem, we see that X is approximately normal with parameters $\mu = 0$ and $\sigma = 1$. In practice, a frequent choice for n is $n = 12$ since this avoids the computation of a square root and divisions.

The following method, the Box-Muller algorithm, is an 'exact method' in the sense that the resulting numbers have essentially perfect accuracy.

3.4.2 Box-Muller algorithm

The *Box-Muller algorithm*[18] is as follows:

Sample random variables $U_1, U_2 \sim U(0,1)$ and set

$$X_1 = \cos(2\pi U_1)\sqrt{-2\ln(U_2)},$$
$$X_2 = \sin(2\pi U_1)\sqrt{-2\ln(U_2)}.$$

Theorem 3.4. *The variables X_1 and X_2 generated by the Box-Muller algorithm are independent normally distributed random variables with mean 0 and variance 1.*

Proof. For notational purposes, let $\varphi(U_1, U_2) = (\cos(2\pi U_1)\sqrt{-2\ln(U_2)}, \sin(2\pi U_1)\sqrt{-2\ln(U_2)})$. Then,

[18] The algorithm is named after G.E.P. Box (Oct. 18, 1919 - March 28, 2013) and M.E. Muller (June 1, 1928 - Dec. 3, 2018), who published this algorithm in a two-page paper in 1958.

$$
\begin{aligned}
Pr(X_1' \in A_1) \cdot Pr(X_2' \in A_2) &= \left(\frac{1}{\sqrt{2\pi}} \int_{A_1} e^{-x_1^2/2} dx_1\right) \cdot \left(\frac{1}{\sqrt{2\pi}} \int_{A_2} e^{-x_2^2/2} dx_2\right) \\
&\overset{(*_0)}{=} \frac{1}{2\pi} \int_{A_2} \left(e^{-x_2^2/2} \int_{A_1} e^{-x_1^2/2} dx_1\right) dx_2 \\
&\overset{(*_1)}{=} \frac{1}{2\pi} \int_{A_2} \int_{A_1} e^{-(x_1^2+x_2^2)/2}\, dx_1\, dx_2, \\
&\overset{(*_2)}{=} \frac{1}{2\pi} \int_0^{\infty} \int_0^{2\pi} \chi_A(r\cos(\theta), r\sin(\theta)) \cdot \exp(-r^2/2) \cdot r\, d\theta\, dr \\
&\overset{(*_3)}{=} \int_0^{\infty} \int_0^{1} \chi_A(\cos(2\pi u_1)\sqrt{2u_2}, \sin(2\pi u_1)\sqrt{2u_2}) e^{-u_2}\, du_1\, du_2 \\
&\overset{(*_4)}{=} -\int_0^{-\infty} \int_0^{1} \chi_A(\cos(2\pi u_1)\sqrt{-2u_2}, \sin(2\pi u_1)\sqrt{-2u_2}) e^{u_2}\, du_1\, du_2 \\
&\overset{(*_5)}{=} \int_0^{1} \int_0^{1} \chi_A(\cos(2\pi u_1)\sqrt{-2\ln(u_2)}, \sin(2\pi u_1)\sqrt{-2\ln(u_2)})\, du_1\, du_2 \\
&= \Pr(\varphi(U_1, U_2) \in A),
\end{aligned}
$$

where in $(*_0)$ and $(*_1)$ we use the linearity of the integral, in $(*_2)$ we substitute polar coordinates (two-dimensional integration) $x_1 \leftarrow r\cos(\theta)$, $x_2 \leftarrow r\sin(\theta)$, with Jacobian $J = \begin{pmatrix} \cos(\theta) & -r\sin(\theta) \\ \sin(\theta) & r\cos(\theta) \end{pmatrix}$, hence $d(x_1, x_2) = |J| dr d\theta = |r| dr d\theta = r dr d\theta$ in $(*_3)$ we substitute (one-dimensional integration) $\theta \leftarrow 2\pi u_1$ and $r \leftarrow \sqrt{2u_2}$, in $(*_4)$ we substitute $u_2 \leftarrow -u_2$, and in $(*_5)$ we substitute $u_2 \leftarrow \ln(u_2)$. □

Remarks:

(a) The above theorem can be rephrased as follows: Given r and θ uniformly distributed on $(0, 1)$ and $(0, 2\pi)$, respectively, then the variables $\sqrt{-2\ln(r)}\cos(\theta)$ and $\sqrt{-2\ln(r)}\sin(\theta)$ are independently standard Gaussian.

(b) In implementations one needs to take care that $U_2 \neq 0$ (since $\ln(0)$ is undefined). If the uniform random number generator returns values strictly larger than 0 (which is the case in matlab's `rand` and most other number generators in general) then this case does not occur.

(c) The algorithm outputs two uniformly distributed numbers. To produce these, it requires computation of 2 uniformly distributed numbers, 1 square roots, 1 logarithm, and 2 sin or cos evaluation.

3.5 Exercises

Exercise 3.1. Modify `vonNeumannMiddleSquare.m` such that instead of squaring, in line 9, it utilizes the sin function. (The generated numbers should be still integers.) What do you observe?

Exercise 3.2. For the LCRNG

$$v_{n+1} = (1 + 2v_n) \bmod 2^{32}$$

show that there exist starting points v_0 that are never in a cycle (that is, the sequence generated by the LCRNG does not loop back to the starting point).

Exercise 3.3. Make a complete examination of the middle-square method in the case of two-digit decimal numbers.

(a) Start the process out with any of the 100 possible values 00, 01, ..., 99. How many of these values lead ultimately to the repeating cycle $00 - 00 - \ldots$?

(b) How many possible final cycles are there?

(c) How long is the longest cycle?

(d) What starting value or values will give the largest number of distinct elements before the sequence repeats?

Exercise 3.4. Let X be a standard Cauchy random variable, i.e., the pdf of X is

$$f_X(x) = \frac{1}{\pi} \cdot \frac{1}{1+x^2}, \qquad x \in \mathbb{R}$$

(a) Compute the cdf of f_X.

(b) Based on cdf inversion, explain how you can sample a standard Cauchy random variable.

Exercise 3.5. Make the matlab code from Example 3.2. more efficient (even a tiny bit), and demonstrate this by comparing computation times.

Exercise 3.6. A key step in the proof of Theorem 3.2 was to show that

$$\{(t,u) \in \mathbb{R} \times (0,1) : F^-(u) \le t\} = \{(t,u) \in \mathbb{R} \times (0,1) : u \le F(t)\}. \tag{3.6}$$

This can be viewed as showing that the super-graph of F^- and the subgraph of F coincide. Under the assumptions of Theorem 3.2 we have $F^-(t) = \min\{t : F(t) \ge u\} = \inf\{t : F(t) \ge u\}$. Give an example of F and $F^-(t) = \inf\{t : F(t) \ge u\}$ (of course, not satisfying the assumptions of Theorem 3.2) where (3.6) does not hold.

Exercise 3.7. Consider $U(a,b)$, the uniform distribution over the interval (a,b). How does one sample from $U(a,b)$ by cdf inversion?

Exercise 3.8. For any fixed natural number n, devise an algorithm to obtain samples from the continuous probability distribution on $(0,1)$ whose pdf is $f_n(x) = nx^{n-1}$.

Exercise 3.9. Repeat Example 3.6 for the target pdf

$$f_X(x) = \begin{cases} 6x(1-x) & : \quad x \in (0,1), \\ 0 & : \quad \text{otherwise.} \end{cases}$$

Exercise 3.10.

(a) Describe a method (either by explaining the steps or giving matlab-type pseudocode) to generate random numbers from the set $\{1, 2, 3\}$. With probability 0.2 the number 1 should be drawn, with probability 0.3 the number 2 should be drawn, and with probability 0.5 the number 3 should be drawn.

(b) Consider the truncated normal distribution

$$f(x) = \begin{cases} \frac{2}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} & : \quad x \geq 0, \\ 0 & : \quad \text{otherwise.} \end{cases}$$

(i) Describe the steps of rejection sampling from f using as trial distribution the standard normal distribution

$$g(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}.$$

(ii) What is the acceptance rate of your algorithm from (i)?

Exercise 3.11. Consider the method of rejection sampling for a discrete pdf f with trial density g. Suppose that, for some reason, we cannot compute the normalizing factors of the density functions, i.e., instead of f and g we are given scaled versions $f^* = c_f f$ and $g^* = c_g g$, with $c_f, c_g > 0$. Suppose, however, in step (a) of the method we can draw samples from g.

Accepting Y in step (c) if $U \leq f^*(Y)/Mg^*(Y)$ will give a method that, in fact, draws samples from f. Show this by filling out the boxes in the following proof.

Proof: Clearly, we have

$$\Pr(X = x) = \Pr(Y = x | U \leq \square \cdot f(Y)/M \cdot \square \cdot g(Y)) = \frac{\Pr(Y = x, U \leq \square \cdot f(Y)/M \cdot \square \cdot g(Y))}{\Pr(U \leq \square \cdot f(Y)/M \cdot \square \cdot g(Y))}.$$

Before we consider the numerator and denominator of the above expression, note that

$$\Pr(\text{accept} | Y = x) = \square \cdot \frac{f(x)}{Mg(x)} \qquad \text{(basic uniform cdf)} \tag{3.7}$$

Now,

$$\begin{aligned} \Pr\left(Y = x, U \leq \square \cdot \frac{f(Y)}{Mg(Y)}\right) &= \Pr(\text{accept} | Y = x) \Pr(Y = x) \qquad \text{(by Bayes' rule)} \\ &= \Pr(\text{accept} | Y = x) \cdot \square \cdot g(x) \\ &= \square \cdot \frac{f(x) \cdot \square \cdot g(x)}{Mg(x)} \qquad \text{(using (3.7))} \\ &= \frac{1}{M} \cdot \square \cdot f(x). \end{aligned}$$

Further,

$$\begin{aligned}
\Pr\left(U \leq \square \cdot \frac{f(Y)}{Mg(Y)}\right) &= \Pr(\text{accept}) \\
&= \sum_x \Pr(\text{accept}|Y=x)\Pr(Y=x) \\
&= \sum_x \cdot \square \cdot \frac{f(x)}{Mg(x)} \cdot \square \cdot g(x) \qquad \text{(using (3.7))} \\
&= \frac{1}{M} \cdot \square \cdot \sum_x f(x) \\
&= \frac{1}{M} \cdot \square \qquad \text{(since } f \text{ is a pdf).}
\end{aligned}$$

Hence $\Pr(X = x) = f(x)$, which needed to be shown.

Exercise 3.12. Alice wants to generate random integer numbers from the set $\{0, 1\}$. With probability 0.8 the number 0 should be drawn, while 1 should be drawn with probability 0.2. She implemented the following rejection method in Matlab.

```
nTrials=10^4;

reject=0;
for i=1:nTrials
  y=randi(2)-1; %draws 0 and 1 uniformly at random
  u=rand(1);    %draws a real number in (0,1) uniformly at random
  while ((y==0) && (u>0.8)) || ((y==1) && (u>0.2))
      reject=reject+1; u=rand(1); %reject
  end;
  x(i)=y; %accept
end;

x %is the vector holding the nTrials generated numbers
```

She is not sure whether her program works correctly.

(a) What is the probability that the number 0 is drawn by her implementation?

(i) 0.8, (ii) 0.5, (iii) 0.2?

(b) Give a brief explanation of your answer given in (a).

4 Markov Chain Monte Carlo (MCMC) methods

Markov Chain Monte Carlo (MCMC) methods are particularly useful when it is practically not feasible to directly sample from the desired distribution. This is often the case, for instance, for combinatorially defined sets where the sample space is so large preventing that the so-called partition function (defined below) can be computed. Also, MCMC methods find applications in cases where it is rather straightforward to compare probabilities of pairs of events.

The general idea behind MCMC is to construct a Markov chain whose stationary distribution is the probability distribution we want to simulate. We then generate samples of the distribution by running the Markov chain. In *Markov chains*[19] one allows only probabilistic dependence on the past through the previous state. This produces already a great diversity of behaviors.

Historically, the idea for MCMC sampling goes back to 1953, when N. Metropolis[20] et al. published the paper "Equations of state calculations by fast computing machines" [8]. The authors were trying to solve problems in physics that arise due to the random kinetic motion of atoms and molecules. We will discuss this later in more detail. The algorithm introduced in this paper, nowadays called **Metropolis** algorithm, has been cited as among the top ten algorithms[21] having the greatest influence on the development of science and engineering. MCMC, whether via Metropolis or modern variations, is now also very important in statistics and machine learning.

Let's start by introducing the background from Markov chain theory that we need for our purposes.

4.1 Markov Chains

We deal exclusively in this chapter with discrete-time homogeneous Markov chains on a finite state space $\Omega = \{\omega_1, \ldots, \omega_M\}$. We will therefore define Markov chains only for this case. Many of the definitions extend to countable state spaces with only minor complication. A more comprehensive treatment (and proofs) can be found, e.g., in [9, 10].

Definition 4.1. *A sequence $\{X_t\}_{t\in\mathbb{N}_0}$ of random variables is a **Markov chain (MC)**, with state space Ω, if*

$$\Pr(X_{t+1} = y \mid X_t = x_t,\ X_{t-1} = x_{t-1}, \ldots,\ X_0 = x_0) = \Pr(X_1 = y \mid X_0 = x_t) \tag{4.1}$$

for all $t \in \mathbb{N}_0$ and all $y, x_t, x_{t-1}, \ldots, x_0 \in \Omega$.

Equation (4.1) encapsulates the **Markovian property** whereby the history of the MC prior to time t is forgotten.

We may write

$$P(x, y) := \Pr(X_{t+1} = y \mid X_t = x),$$

where P is the **transition matrix** of the MC. The transition matrix P describes single-step transition probabilities; the t-step transition probabilities P^t are given inductively by

$$P^t(x, y) := \begin{cases} I(x, y) & : \quad t = 0, \\ \sum_{y'\in\Omega} P^{t-1}(x, y')P(y', y) & : \quad t > 0, \end{cases}$$

where I denotes the identity matrix $I(x, y) := \delta_{xy}$. Thus,

$$P^t(x, y) = \Pr(X_t = y \mid X_0 = x).$$

Also note that the row sums of P satisfy $\sum_{y\in\Omega} P(x, y) = \sum_{y\in\Omega} \Pr(X_1 = y|X_0 = x) = 1$, because given that we are in state x, the next state must be one of the possible states from Ω.

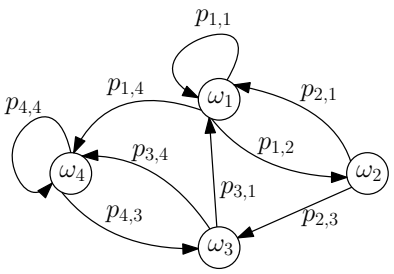
MCs can be represented by directed graphs. In this representation, the states are represented by the vertices of the graph. A directed edge extends from one vertex, x say, to another y, if a transition from x to y is possible in one iteration. In this case the weight $P(x, y)$ is associated to that edge. The set $N(x)$ of all states that can be reached from x in one iteration is the so-called **neighborhood of x.**

[19] They are named after the Russian mathematician Andrey Andreyevich Markov (1856–1922) who began to study such processes in 1907.

[20] Nicholas Constantine Metropolis (June 11, 1915 – October 17, 1999) was a Greek-American physicist.

[21] See, e.g., http://www.uta.edu/faculty/rcli/TopTen/topten.pdf.

Example 4.1. *The following is a graph representation of a four-state Markov chain with transition matrix P.*



$$P = \begin{array}{c} \begin{matrix} \omega_1 & \omega_2 & \omega_3 & \omega_4 \end{matrix} \\ \begin{pmatrix} p_{1,1} & p_{1,2} & 0 & p_{1,4} \\ p_{2,1} & 0 & p_{2,3} & 0 \\ p_{3,1} & 0 & 0 & p_{3,4} \\ 0 & 0 & p_{4,3} & p_{4,4} \end{pmatrix} \end{array} \begin{matrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{matrix}$$

The neighborhood of, for instance, ω_1 is $N(\omega_1) = \{\omega_1, \omega_2, \omega_4\}$.

An MC is **irreducible** if, for all $x, y \in \Omega$, there exists a $t \in \mathbb{N}_0$ such that $P^t(x, y) > 0$. In other words, *irreducibility* is the property that regardless of the present state we can reach any other state in finite time. (Equivalently, in the graph representation any vertex can be reached by a directed path from any other vertex.)

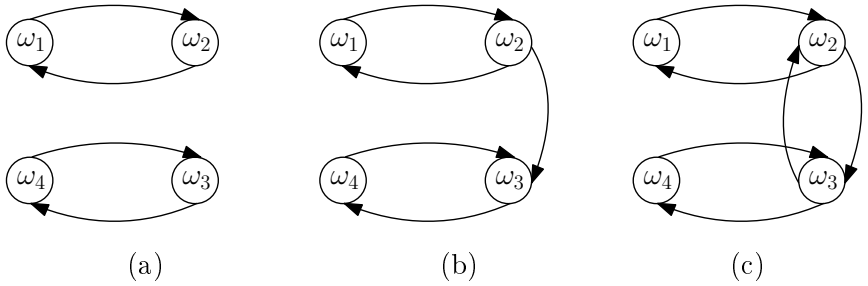


Figure 19: Examples of three Markov chains. Only (c) is irreducible, (a) and (b) are not.

Each state x in a Markov Chain has a period $\gcd\{t : P^t(x, x) > 0\}$. An MC is **aperiodic** if all states have period 1. Otherwise, it is called **periodic**. To show aperiodicity, we remark that in the case of an irreducible MC, it suffices to verify that the period of just one state $x \in \Omega$ is 1.

Figure 20: Examples of two Markov chains. (a) is periodic with period 2, (b) is aperiodic.

A **stationary distribution** of an MC with transition matrix P is a pdf $\pi : \Omega \to [0, 1]$ satisfying

$$\pi(y) = \sum_{x \in \Omega} \pi(x) P(x, y) \qquad \text{or, equivalently,} \qquad \pi = \pi P, \tag{4.2}$$

where in the last equation we write $\pi = (\pi_{\omega_1}, \ldots, \pi_{\omega_M})$ as row vector. Equation (4.2) is known as **global balance equation**.

Clearly, if X_0 is distributed as the stationary distribution π then so is X_1 (and hence so is X_t for all $t \in \mathbb{N}_0$).

It can be shown that a finite MC always has at least one stationary distribution. It is often possible to determine the stationary distributions by solving the global balance equation keeping in mind that $\pi = (\pi_{\omega_1}, \ldots, \pi_{\omega_M})$ needs to be a non-negative vector satisfying additionally the **normalizing condition**

$$\sum_{i=1}^{M} \pi_{\omega_i} = 1. \tag{4.3}$$

The stationary distribution, however, might not be unique (see Example 4.4). Uniqueness is guaranteed if the Markov chain is irreducible and aperiodic.

Theorem 4.1. *An irreducible aperiodic MC has a unique stationary distribution π; moreover the MC tends to π in the sense that $P^t(x,y) \to \pi(y)$, as $t \to \infty$, for all $x \in \Omega$.*

Informally speaking, an irreducible aperiodic MC eventually 'forgets' its starting state.

Example 4.2. *Consider the task of finding a stationary distribution of the Markov chain with transition matrix*

$$P = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 2/3 & 1/3 \\ p & 1-p & 0 \end{pmatrix},$$

for some given $p \in (0,1)$.

Let $\pi = (\pi_1, \pi_2, \pi_3)$. Equation $\pi P = \pi$ is equivalent to

$$(p\pi_3, \pi_1 + \frac{2}{3}\pi_2 + (1-p)\pi_3, \frac{1}{3}\pi_2) = (\pi_1, \pi_2, \pi_3).$$

Solving this we find $\pi_1^* = p\pi_3^*$, $\pi_2^* = 3\pi_3^*$ and π_3^* can be chosen arbitrarily. However, since π needs to be a pdf, we need to have $\pi_1 + \pi_2 + \pi_3 = 1$ (and $\pi_1, \pi_2, \pi_3 \geq 0$). This gives the (unique) solution

$$(\pi_1^*, \pi_2^*, \pi_3^*) = \frac{1}{p+4}(p, 3, 1).$$

(It is easily checked that we made no computational mistakes; just verify $(\pi_1^*, \pi_2^*, \pi_3^*)P = (\pi_1^*, \pi_2^*, \pi_3^*)$.)

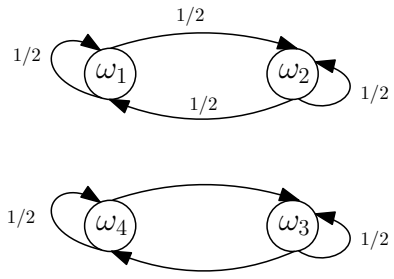
Example 4.3. *Let us consider the question whether Theorem 4.1 also holds if the MC is periodic.*

Consider the Markov chain in Fig. 20(a) with transition probabilities $p_{1,2} = p_{3,2} = 1$ and $p_{2,1} = p_{2,3} = 1/2$. We have

$$P^{2t+1} = \begin{pmatrix} 0 & 1 & 0 \\ 1/2 & 0 & 1/2 \\ 0 & 1 & 0 \end{pmatrix}, \qquad P^{2t+2} = \begin{pmatrix} 1/2 & 0 & 1/2 \\ 0 & 1 & 0 \\ 1/2 & 0 & 1/2 \end{pmatrix},$$

for all $t \geq 0$. Hence we do not have convergence $P^t(x,y) \to \pi(y)$ as $t \to \infty$ for all $x \in \Omega$ in the sense of Theorem 4.1. The stationary distribution is, by the way, $\pi = (1/4, 1/2, 1/4)$.

Example 4.4. *Consider the following MC.*



$$P = \begin{array}{c} \\ \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{array} \begin{array}{c} \begin{array}{cccc} \omega_1 & \omega_2 & \omega_3 & \omega_4 \end{array} \\ \begin{pmatrix} 1/2 & 1/2 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 1/2 & 1/2 \end{pmatrix} \end{array}$$

It is easily verified that both $\pi = (1/2, 1/2, 0, 0)$ and $\pi = (0, 0, 1/2, 1/2)$ are stationary distributions of this Markov chain. Does this contradict Theorem 4.1?

The answer is No. *The MC is clearly not irreducible (there is for instance no path from ω_1 to ω_3 in the graph representation of the MC).*

In many situations one has an idea about what the stationary distribution might be. Instead of verifying the global balance condition it can be easier to verify so-called **detailed balance**, which is (4.4) in the following theorem. (An MC for which detailed balance holds is said to be *time reversible*. The Markov chains constructed by the sampling methods discussed later are all time reversible.)

Theorem 4.2. *Suppose P is the transition matrix of an MC. If the function $\pi' : \Omega \to [0,1]$ satisfies*

$$\pi'(x)P(x,y) = \pi'(y)P(y,x), \qquad \text{for all } x, y \in \Omega, \tag{4.4}$$

and

$$\sum_{x\in\Omega} \pi'(x) = 1,$$

then π' is a stationary distribution of the MC. If the MC is irreducible and aperiodic then π' is the unique stationary distribution.

Proof. We verify that π' is a stationary distribution (see (4.2)). Suppose, X_0 is distributed as π' then

$$\sum_{x\in\Omega} \pi'(x)P(x,y) = \sum_{x\in\Omega} \pi'(y)P(y,x) = \pi'(y).$$

Note that the last equality follows from the law of total probability. □

Let us consider two sampling problems for types of objects that are rather central in discrete mathematics. The first is about sampling independent sets in a graph, the second is about sampling of matchings in a graph. Independent sets and matchings appear in many contexts. For instance, independent sets can model conflicts between objects. The vertices might correspond to workers and an edge might indicate that the corresponding pair of workers cannot work in the same shift (because they don't like each other or they need to share the same equipment, etc.). Matchings are often used to pair objects; for instance, assigning students to courses, jobs to machines, etc.

We remark that there many computational questions arising in this context. Some of these questions are settled, some remain subjects of ongoing research. (It is known, for instance, that both problems of counting the number of independent sets in a graph and counting the number of matchings in a graph is an intractable problem[22]. Still it is possible to sample efficiently, that means in polynomial time in n, matchings in graphs, while such effient sampling of independent sets is intractable. Note that the sampling procedures that we will discuss have a-priorily an exponential running time. More on this fascinating topic can be found in [11].)

Example 4.5. *Let $G = (V, E)$ be a graph with vertices in $V = \{v_1, \ldots, v_n\}$ and edges in $E = \{e_1, \ldots, e_m\}$. A set $I \subseteq V$ is an* independent set *if no two vertices of I are joined by an edge in E (see Fig. 21(a)).*

The following Markov chain randomly selects independent sets in $\mathcal{I}_G :=$ {all independent sets in G}.

Let $I \in \mathcal{I}_G$ be given (e.g., $I = \emptyset$).

(a) Select $v \in V$ uniformly at random;

(b) Flip a fair coin;

*(c) **If** 'heads' and no neighbor of v is in I, add v to I (i.e., set $I \leftarrow I \cup \{v\}$), **otherwise**, remove v from I (i.e., set $I \leftarrow I \setminus \{v\}$).*

It can be shown (Exercise 22) that the Markov chain is irreducible, aperiodic and has the uniform distribution on $\mathcal{I}_G$ as stationary distribution. Hence, we can use this Markov chain to (uniformly) generate random independent sets in G.

Figure 21: A graph with vertices $v_1, \dots, v_4$ and edges $\{v_1, v_2\}$, $\{v_1, v_3\}$, $\{v_2, v_3\}$, $\{v_3, v_4\}$. (a) Independent set $I = \{v_1, v_4\}$ (gray nodes), (b) Matching $M = \{\{v_1, v_2\}, \{v_3, v_4\}\}$ (red edges). I is also a maximum independent set; M is also a maximum matching in G.

Example 4.6. *Let $G = (V, E)$ be a graph with vertices in $V = \{v_1, \dots, v_n\}$ and edges in $E = \{e_1, \dots, e_m\}$. A set $M \subseteq E$ is a* matching *if the edges of M are pairwise vertex disjoint (see Fig. 21(b)).*

The following Markov chain randomly selects matchings in $\mathcal{M}_G :=$ {all matchings in G}.

Let $M \in \mathcal{M}_G$ be given (e.g., $M = \emptyset$).

(a) Select $e \in E$ uniformly at random;

(b) Flip a fair coin[23]*;*

(c) ***If*** *'heads' then do not change M (i.e., set $M \leftarrow M$),*
otherwise *do the following: Set*

$$M' := \begin{cases} M' = M \cup \{e\} & : \ e \notin M, \\ M' = M \setminus \{e\} & : \ \text{otherwise}, \end{cases}$$

and set $M \leftarrow M'$ if M' is a matching, otherwise set $M \leftarrow M$.

We claim that the Markov chain is irreducible, aperiodic and has the uniform distribution on $\mathcal{M}_G$ as stationary distribution. Hence, we can use this Markov chain to (uniformly) generate random matchings in G.

The state space of the Markov chain is $\Omega = \mathcal{M}_G$. This is a finite set containing at most 2^m elements. Clearly the chain is homogeneous (there is only dependence on the previous state).

(a) Aperiodicity: *The Markov chain might remain in the same state (see step (a) and step (c)), hence all states have period 1, and the chain is therefore aperiodic.*

(b) Irreducibility: *The Markov chain is irreducible, since it is possible to reach the empty matching from any state by removing edges (and reach any state from the empty matching by adding edges).*

(c) Uniform distribution: *We verify the detailed balance condition for $\pi = \frac{1}{|\mathcal{M}_G|}(1, \dots, 1)$, where $|\mathcal{M}_G|$ denotes the size/cardinality of the set $\mathcal{M}_G$. Note that as $|\mathcal{M}_G|$ can be rather large it is often not feasible to compute this number explicitly.*

For $M_1, M_2 \in \mathcal{M}$ with $M_2 = M_1 \cup \{e\}$ for some $e \in E \setminus M_1$ we have

$$\pi(M_1)\Pr(X_1 = M_2 \mid X_0 = M_1) = \frac{1}{m} \cdot \frac{1}{2} = \pi(M_2)\Pr(X_1 = M_1 \mid X_0 = M_2).$$

Therefore, by Theorem 4.2, the uniform distribution π is a stationary distribution.

[22] Unless the famous conjecture $\mathbb{P} \neq \mathbb{NP}$ would surprisingly fail to hold; see also https://www.claymath.org/sites/default/files/pvsnp.pdf.

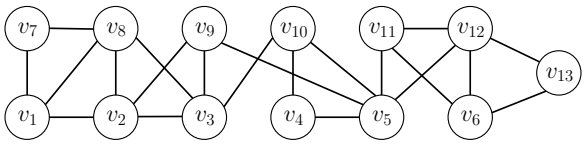
[23] To flip a fair coin means to draw a random number taking on only two values, say 0 (for heads) and 1 (for tails), both of which should occur with probability 1/2.

The matlab implementation might look like the one shown in Fig. 22.

```
function [M]=randommatchingsample(n,edges)
  %n is the number of vertices of the graph
  %ith row of edges cotains the numbers of the two vertices of that edge

  nTrials=10^3; %number of iteration for running the Markov chain
  m=size(edges,1);
  M=zeros(1,m); %initialize M
  saturated=zeros(1,n);

  for sim=1:nTrials %Implementation of the algorithm from Example 4.6. in the lecture notes
    newe=randi(m,1);
    coin=randi(2,1);
    if (coin==1)
        if (M(newe)==0)
            if (sum(saturated(edges(newe,:)))==0)
                M(newe)=1; saturated(edges(newe,:))=[1,1];
            end;
        else
            M(newe)=0; saturated(edges(newe,:))=[0,0];
        end;
  end;
end
```

Figure 22: Sampling matchings in a graph; see Example 4.6.

Example 4.7. *Let's continue with sampling matchings in a graph (Example 4.6). In fact, we now want to focus in the problem of* counting *matchings in the following graph.*



By enumeration (a method, which, for larger n, becomes quickly intractable) we obtain the following number s_i of matchings of size $i = 0, \ldots, 21$.

i	0	1	2	3	4	5	6	7-21
s_i	1	21	158	521	749	403	61	0

The matlab code for enumerating all possible solutions might look like the one shown in Fig. 23.

Now, suppose we sample matchings in G (using the method from Example 4.7). If we do this for sufficiently many trials we should get a good picture about the distribution of the matchings. Note that this is still an exponential method! But depending on the situation (whether or not the Markov chain is so-called rapidly mixing*) it may provide good approximations in reasonable time.*

Fig. 24 shows the outcome of a random sampling of $\mathrm{nTrials} = 10^4$ *matchings (drawn after evolving the Markov chain for 10^3 steps); depicted are the relative frequencies $n_i/$`nTrials` of the events that M is a matching of size i, with i being depicted on the x-axis.*

Now, observe that we can obtain estimates $\hat{s}_i$ of the number of matchings of size i from this data. We know that $s_0 = 1$ (there is one matching of size 0, the empty set). Therefore $\hat{s}_0 = 1$. As $n_1/n_0 \approx s_1/s_0$ we can set $\hat{s}_1 := \hat{s}_0 n_1/n_0$ and, recursively, $\hat{s}_i := \hat{s}_{i-1} n_i/n_{i-1}$. The estimates that we obtain for the present data are depicted in Table 4.

4.2 Metropolis-Hastings

We turn now to a slightly different question. *How can we modify the transition probabilities in a given Markov chain to ensure that the stationary distribution is equal to a prescribed distribution π?*

```
edges=[1,7;1,8;2,8;2,3;3,8;3,10;4,10;5,9;5,10;5,12;6,11;
       6,12;7,8;3,9;4,5;2,9;5,11;11,12;1,2;12,13;6,13];
m=size(edges,1); n=max(max(edges));
matchingsofsize=zeros(1,m+1);
nTrials=10^4;

%Check all 2^m possibilities for M, and count which of them are matchings
%This determines the correct total number of matchings in G.
%This can only work for small m, since it takes exponential time (in m)
for i=0:2^m-1
  curE=de2bi(i,m);
  saturated=zeros(1,n);
  for j=1:m
      if (curE(j)==1)
          saturated(edges(j,1))=saturated(edges(j,1))+1;
          saturated(edges(j,2))=saturated(edges(j,2))+1;
      end;
  end;
  if (max(saturated)<2)
      matchingsofsize(sum(curE)+1)=matchingsofsize(sum(curE)+1)+1;
  end;
end;

matchingsofsize
```

Figure 23: Matlab code for enumerating all matchings in the graph of Example 4.7.

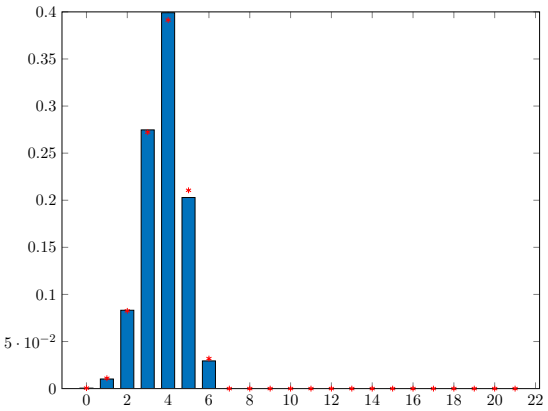


Figure 24: Histogram of the relative frequencies of matchings of different sizes based on `nTrials` $= 10^4$ samples drawn after evolving the Markov chain from Example 4.7 for 10^3 steps. Values s_i/`nTrials` depicted as red points.

The method that we introduce here for solving this question is the so-called *Metropolis-Hastings* algorithm. It is named after Metropolis et al. (1953), which was a major breakthrough in Monte Carlo methods and Hastings[24] (1970), which was a significant generalization[25].

The transitions in Metropolis-Hastings work as follows.

- Start with some point x_0, whether deterministic or randomly sampled. For $t \geq 0$, given that $X_t = x$, generate a random proposal Y from a distribution $\Pr(Y = y \mid X = x) = P(x, y)$ (note that this P is the transition matrix of our original Markov chain).

[24] Wilfred Keith Hastings (July 21, 1930 – May 13, 2016)

[25] A very readable account on the history of the Metropolis-Hastings algorithm can be found at https://www.jstor.org/stable/30037292.

i	0	1	2	3	4	5	6	7-21
s_i	1	21	158	521	749	403	61	0
n_i	5	102	832	2747	3991	2029	294	0
$\hat{s}_i = \hat{s}_{i-1} n_i / n_{i-1}$	1.0	20.4	166.4	549.4	798.2	405.8	58.8	0

Table 4: Estimating the number of matchings of size i (Example 4.7).

- New to what we did previously, accept or reject the proposal y. With probability $A(x,y)$ accept it and set $X_{t+1} = Y$. With probability $1 - A(x,y)$ reject the proposal and then $X_{t+1} = X_t$.

This may now be considered to be a new Markov chain. We denote the new transition probabilities by $P'(x,y)$, $x, y \in \Omega$. Note that

$$P'(x,y) = \Pr(Y = y | X = x) = P(x,y)A(x,y). \tag{4.5}$$

Given a desired stationary distribution π and a proposal mechanism via matrix Q, we want to design the acceptance probabilities $A(x,y)$ such that π is a stationary distribution of the new Markov chain. In view of Theorem 4.2 it suffices to construct the acceptance probabilities $A(x,y)$ in such a way that detailed balance is satisfied.

The **Metropolis-Hastings acceptance probability** is given by

$$A(x,y) := \begin{cases} \min\left(1, \frac{\pi(y)P(y,x)}{\pi(x)P(x,y)}\right) & \text{if } \pi(x)P(x,y) > 0, \\ 1 & \text{otherwise,} \end{cases} \qquad x, y \in \Omega. \tag{4.6}$$

We remark that the Metropolis-Hastings (MH) algorithm simulates samples from a probability distribution by making use of the full joint density function and (independent) proposal distributions for each of the variables of interest. In summary, the HR algorithm is as follows.

Metropolis-Hastings algorithm

Input: Probability distribution function π (may be unnormalized), transition matrix P.
Output: Random sample x from the Markov chain.

(a) Initialize x (deterministic or randomly).

(b) Draw proposal y from $\Pr(Y \mid X = x) = P(x, \cdot)$.

(c) Compute acceptance probability $A(x,y)$ according to (4.6).

(d) Draw u from $U(0,1)$.

(e) **If** $u \leq A(x,y)$ then 'accept proposal' (i.e., set $x \leftarrow y$)
otherwise 'reject' (i.e., set $x \leftarrow x$).

(f) Go to step (b).

Note that the condition $u \leq A(x,y)$ realizes the desired probability $A(x,y)$ of accepting the proposal y. (You can see this by considering cdf inversion for a discrete random variable taking on only two values, 'accept' and 'not accept,' where 'accept' should occur with probability $A(x,y)$.) Further notice that the ratio inside the minimum in (4.6) has a factor $\pi(y)/\pi(x)$. Other things being equal, this implies that moves to higher probability states y are favored. The second factor, $P(y,x)/P(x,y)$, implies, if other things are equal, that we hesitate to move to y if it would be hard to get back to x.

What should be also noted is that the factor Z for an unnormalized distribution π_u with $\pi = \pi_u/Z$ cancels out in (4.6). Hence the same acceptance rule can be applied if the partition function Z is unknown.

Theorem 4.3. *The Markov chain generated by the Metropolis-Hastings algorithm has π as stationary distribution.*

Proof. We want to prove detailed balance. We distinguish the following three cases for $x, y \in \Omega$.

(i) Suppose, $\pi(x)P(x,y) = \pi(y)P(y,x)$. Then, $A(x,y) = A(y,x) = 1$, implying

$$\pi(x)P(x,y)A(x,y) = \pi(y)P(y,x)A(y,x),$$

hence $\pi(x)P'(x,y) = \pi(y)P'(y,x)$, showing detailed balance for this case.

(ii) Suppose $\pi(x)P(x,y) > \pi(y)P(y,x)$. In this case

$$A(x,y) = \frac{\pi(y)P(y,x)}{\pi(x)P(x,y)} \qquad \text{and} \qquad A(y,x) = 1.$$

Hence,

$$\pi(x)P'(x,y) = \pi(x)P(x,y)A(x,y) = \pi(x)P(x,y) \cdot \frac{\pi(y)P(y,x)}{\pi(x)P(x,y)}$$

$$= \pi(y)P(y,x)A(y,x) = \pi(y)P'(y,x).$$

(iii) Suppose $\pi(x)P(x,y) < \pi(y)P(y,x)$. In this case

$$A(x,y) = 1 \qquad \text{and} \qquad A(y,x) = \frac{\pi(x)P(x,y)}{\pi(y)P(y,x)}.$$

Hence

$$\pi(y)P'(y,x) = \pi(y)P(y,x)A(y,x) = \pi(y)P(y,x) \cdot \frac{\pi(x)P(x,y)}{\pi(y)P(y,x)}$$

$$= \pi(x)P(x,y)A(x,y) = \pi(x)P'(x,y).$$

□

Since the algorithm allows for rejection, the Markov chain of the MH algorithm is clearly aperiodic. What is left to check to ensure convergence to the required target distribution is in each specific case only irreducibility.

Example 4.8. *Let's consider Example 4.5 again.*

We have shown (Exercise 22) that the MC in that example has the uniform distribution as stationary distribution. Suppose now that we want a stationary distribution where each independent set I has a probability proportional to $\lambda^{|I|}$. (When $\lambda = 1$ this is the uniform distribution; when $\lambda > 1$, larger independent sets have a larger probability than smaller independent sets; and when $\lambda < 1$ then smaller independent sets have a larger probability than larger independent sets.)

Let's change the algorithm from Example 22 such that we obtain the Metropolis-Hastings variants which samples from $\frac{1}{Z}\lambda^{|I|}$. Here is the modified algorithm.

Let $I \in \mathcal{I}_G$ be given (e.g., $I = \emptyset$).

(a) Select $v \in V$ uniformly at random;

(b) Flip a fair coin;

(c) ***If*** *'heads' and no neighbor of v is in I, then propose $I' \leftarrow I \cup \{v\}$,* ***otherwise****, propose $I' \leftarrow I \setminus \{v\}$.*

(d) Draw u from $U(0,1)$.

(e) ***If*** *$u \leq A(x,y) = \min\left(1, \lambda^{|I'|}/\lambda^{|I|}\right)$ then accept proposal (i.e., set $I \leftarrow I'$),* ***otherwise*** *reject proposal (i.e., set $I \leftarrow I$).*

Fig. 25 shows some output for the choices of $\lambda = 4$ and $\lambda = 0.1$.

Let's consider the output for $\lambda = 4$. We actually have in this particular case $\tilde{n}_0 = 202$ sampled sets of size 0, $\tilde{n}_1 = 3384$ sets of size 1, and $\tilde{n}_2 = 6414$ sets of size 2 (and no sets of larger size).

Can we estimate from this the actual number n_i of independent sets of size i, $i = 1, \dots, 4$, in G?

Well, we can first of all observe that we know how many independent sets there are of size 0 and size 1; there is one independent set of size 0 and there are $n = 4$ independent sets of size 1.

Theoretically we should have

$$\frac{\tilde{n}_1}{\tilde{n}_0} = \frac{\frac{\lambda^1 n_1}{Z} \cdot \mathrm{nTrials}}{\frac{\lambda^0 n_0}{Z} \cdot \mathrm{nTrials}} = \frac{4 \cdot 4}{1 \cdot 1} = 16.$$

And, in fact, for these samples we have $\tilde{n}_1/\tilde{n}_0 = 3384/202 \approx 16.7525$. We can use this argument to estimate any of the n_i.

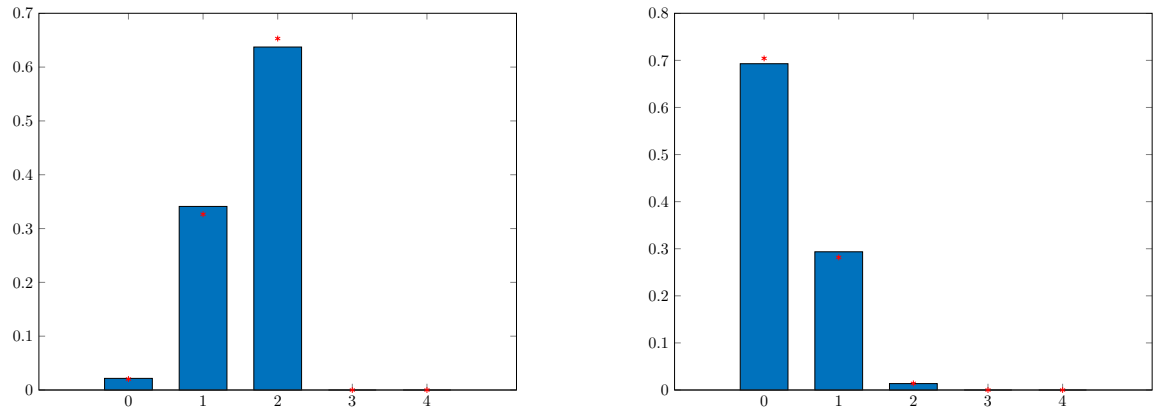
We should have

$$\frac{\tilde{n}_2}{\tilde{n}_1} = \frac{\frac{\lambda^2 n_2}{Z} \cdot \mathrm{nTrials}}{\frac{\lambda^1 n_1}{Z} \cdot \mathrm{nTrials}} = \frac{4 n_2}{n_1}.$$

Solving this for n_2 we obtain,

$$n_2 = \frac{\tilde{n}_2}{\tilde{n}_1} \cdot \frac{n_1}{4} = \frac{6414}{3384} \cdot \frac{4}{4} \approx 1.8954,$$

which is actually not far from the true value of $n_2 = 2$.



(a) (b)

Figure 25: Sampling independent sets (Example 4.8). Histogram for (a) $\lambda = 4$ and (b) $\lambda = 0.1$ (the red markers show the correct value of $\frac{1}{Z}\lambda^{|I|}$) based on `nTrials` $= 10^4$ samples drawn after evolving the Markov chain for 20 steps.

4.3 Other samplers

Many other samplers have been developed in the past. We just list a few of them, which can be frequently found in the literature. (We omit the cases $\pi(x)P(x,y) = 0$ in the statement of the acceptance probabilities; there are conditions that can ensure that $\pi(x)P(x,y) > 0$.)

Barker's sampler The following is an acceptance probability due to Barker[26] (1965)

$$A(x,y) := \begin{cases} \frac{\pi(y)P(y,x)}{\pi(x)P(x,y)+\pi(y)P(y,x)} & \text{if } \pi(x)P(x,y) + \pi(y)P(y,x) \neq 0, \\ 1 & \text{otherwise.} \end{cases} \tag{4.7}$$

[26] A. A. Barker, a mathematical physicist working at that time at the University of Adelaide.

It is easily verified that $A(x, y)$ satisfies detailed balance and that $A(x, y) \leq A_{MH}(x, y)$ for any $x, y \in \Omega$, with A_{MH} denoting the Metropolis-Hastings acceptance probability from (4.6). See Exercise 25.

Original Metropolis sampler The original Metropolis algorithm is the special case of Metropolis-Hastings where $P(x, y) = P(y, x)$. In this case,

$$A(x, y) := \min\left(1, \frac{\pi(y)}{\pi(x)}\right). \tag{4.8}$$

Independence sampler Perhaps the simplest proposal mechanism is to take independent and identically distributed proposals from some distribution P that does not even depend on the present x. Then $P(x, y)$ is simply $P(y)$ for a pdf P. The Metropolis-Hastings proposal for this so-called independence sampler simplifies to

$$A(x, y) := \min\left(1, \frac{\pi(y)P(x)}{\pi(x)P(y)}\right), \qquad x, y \in \Omega. \tag{4.9}$$

Gibbs sampler The Gibbs sampler[27] originates from the field of image processing; see [12]. It is a special case of Metropolis-Hastings sampling where the proposal is always accepted (i.e., $A(x, y) = 1$ for all $x, y \in \Omega$.) It is designed to work for distributions where each state x is in fact a vector $x = (\xi_1, \ldots, \xi_d)$. In image processing, for instance, ξ_i might represent the color of the ith pixel.

The main idea of Gibbs sampling is that one only uses *univariate* conditional distributions (the distribution where all of the random variables but one are assigned fixed values). It is often far easier to sample from such conditional distributions than from the joint distribution.

The Gibbs sampler (more precisely, the *random scan Gibbs sampler*) proceeds as follows. Given a (current) state x then a (uniformly) random coordinate k is selected, and the new state is

$$y = (\xi_1, \ldots, \xi_{k-1}, \xi, \xi_{k+1}, \ldots, \xi_d),$$

with the kth coordinate of y being ξ determined from the pdf for ξ, given the other coordinates.

4.4 Simulated annealing

Simulated annealing[28] is an algorithmic approach for finding the maximum of a function that has typically many local maxima. The idea is that when we initially start sampling the space Ω, we will accept a reasonable probability of a down-hill move in order to explore the entire space. As the process proceeds, we decrease the probability of such down-hill moves. The analogy (and hence the term) is the annealing of a crystal as temperature decreases (initially there is a lot of movement, which then over time gradually decreases).

Simulated annealing is very closely related to Metropolis sampling, differing only in that the probability $A(x, y)$ of a move is given by

$$A(x, y) = \min\left(1, \left(\frac{\pi(y)}{\pi(x)}\right)^{T(t)}\right)$$

where the function $T(t)$ is called the **annealing schedule** (for $T = 1$ we have the Metropolis algorithm). The particular value $T(t)$ for any given $t \in \mathbb{N}_0$ is typically referred to as **temperature**.

A typical choice for T over $T_{\max}$ time steps is

$$T(t) := T_0 \left(\frac{T_{\mathrm{f}}}{T_0}\right)^{t/T_{\max}},$$

where T_0 is a given 'starting temperature' cooling down to a 'final temperature' T_{f}.

[27] Named after the physicist Josiah Willard Gibbs (February 11, 1839 – April 28, 1903), in reference to an analogy between the sampling algorithm and statistical physics.

[28] Simulated annealing was developed in 1983 by Kirkpatrick et al.; see S. Kirkpatrick, C. Gelatt, M. Vecchi, Optimization by simulated annealing, Science 220 (4598) (1983) 671–680, https://science.sciencemag.org/content/220/4598/671.

4.5 An example of a Gibbs sampler

Example 4.9. *(See also Exercise 26).*
Consider a 2D image consisting of an $n \times n$ grid of black and white pixels. Let X_j, $j = 1, \ldots, n^2$, denote the indicator of the jth pixel being white (i.e., $X_j = 1$ if the jth pixel is white, and $X_j = 0$ otherwise). Viewing the pixels as vertices in a graph, the set $N(i)$ of neighbors of the ith pixel are the pixels immediately above, below, to the left, and to the right (except for boundary cases).

A commonly used model[29] *for the pdf π of $X = (X_1, \ldots, X_{n^2})$ is*

$$\pi(x) = \frac{1}{Z} e^{\frac{\beta}{2} \sum_{i=1}^{n^2} \sum_{j \in N(i)} \delta_{\xi_i, \xi_j}},$$

where $x = (\xi_1, \ldots \xi_{n^2})$, $\beta > 0$, and

$$\delta_{\xi_i, \xi_j} = \begin{cases} 1 & \text{if } \xi_i = \xi_j, \\ 0 & \text{otherwise.} \end{cases}$$

With this π neighboring pixels prefer to have the same color. The normalizing constant Z of π is a sum over all 2^{n^2} possible configurations, so it may be very difficult to compute. This motivates the use of MCMC to simulate from the model.

Let us now develop a Gibbs sampler for sampling from π.

Denote the vector of random variables by $X = (X_1, \ldots, X_{n^2})$; a sample is denoted by $x = (\xi_1, \ldots, \xi_{n^2})$. Further, let $X_{-k} = (X_1, \ldots, X_{k-1}, X_{k+1}, \ldots, X_{n^2})$ and $x_{-k} = (\xi_1, \ldots, \xi_{k-1}, \xi_{k+1}, \ldots, \xi_{n^2})$.

Note that π can be viewed as joint pdf of the variables X and X_{-k}, hence

$$\pi(\xi_1, \ldots, \xi_{k-1}, \xi_k, \xi_{k+1}, \ldots, \xi_{n^2}) = \pi_{X_k, X_{-k}}(\xi_k, (\xi_2, \ldots, \xi_{n^2})) = \pi_{X_k \mid X_{-k} = (\xi_2, \ldots, \xi_{n^2})}(\xi_k) \pi_{X_{-k}}(x_{-k}),$$

for $\xi_k \in \{0, 1\}$. Assuming $\pi_{X_{-k}}(x_{-k}) \neq 0$ we therefore have

$$\pi_{X_k \mid X_{-k} = (\xi_2, \ldots, \xi_{n^2})}(\xi_k) = \alpha \cdot \pi(\xi_1, \ldots, \xi_{k-1}, \xi_k, \xi_{k+1}, \ldots, \xi_{n^2}) = \frac{\alpha}{Z} e^{\frac{\beta}{2} \sum_{i=1}^{n^2} \sum_{j \in N(i)} \delta_{\xi_i, \xi_j}},$$

with $\alpha := 1/\pi_{X_{-k}}(x_{-k}) > 0$. Now, notice that only a few terms in the sum $\sum_{i=1}^{n^2} \sum_{j \in N(i)} \delta_{\xi_i, \xi_j}$ depend on ξ_k. In fact, the only terms depending on ξ_k are $\sum_{j \in N(k)} \delta_{\xi_k, \xi_j} + \sum_{i \in N(k)} \delta_{\xi_i, \xi_k}$, hence we can write

$$\pi_{X_k \mid X_{-k} = (\xi_2, \ldots, \xi_{n^2})}(\xi_k) = \frac{\alpha}{Z} e^{\beta R/2} \cdot e^{\beta \sum_{j \in N(k)} \delta_{\xi_k, \xi_j}},$$

where $R \geq 0$ is a constant.

We do not need to worry to much about the factor $\frac{\alpha}{Z} e^{\beta R/2}$. In fact, we will see that we can compute it! But before we do this, note that

$$e^{\beta \sum_{j \in N(k)} \delta_{\xi_k, \xi_j}} = e^{\beta n_{\xi_k}},$$

with n_{ξ_k} denoting the number of neighbors of the kth pixel having color ξ_k.

Now let us consider the constant $C := \frac{\alpha}{Z} e^{\beta R/2}$. We first note that $\pi_{X_k \mid X_{-k} = (\xi_2, \ldots, \xi_{n^2})}(\xi_k)$ is a pdf and that ξ_k can only have the two possible values $\xi_k = 0$ and $\xi_k = 1$. Hence

$$\pi_{X_k \mid X_{-k} = (\xi_2, \ldots, \xi_{n^2})}(0) + \pi_{X_k \mid X_{-k} = (\xi_2, \ldots, \xi_{n^2})}(1) = C e^{\beta n_0} + C e^{\beta n_1} = 1,$$

[29] This model is called **Ising model**, named after the physicist Ernst Ising (May 10, 1900 - May 11, 1998). This model, easy to define but with amazingly rich behavior, serves as a mathematical model of ferromagnetism in statistical mechanics. See also, e.g., http://personal.rhul.ac.uk/uhap/027/ph4211/PH4211_files/brush67.pdf.

consequently

$$C = \frac{1}{e^{\beta n_0} + e^{\beta n_1}}.$$

Putting things together we have

$$\pi_{X_k \mid X_{-k}=(\xi_2,\ldots,\xi_{n^2})}(\xi_k) = Ce^{\beta n_{\xi_k}} = \frac{e^{\beta n_{\xi_k}}}{e^{\beta n_0} + e^{\beta n_1}}, \qquad \xi_k \in \{0,1\}.$$

In summary, we obtain the following Gibbs sampler. Typical results are shown in Fig. 26. (It is interesting to note that the Gibbs sampler is here, in comparison to the Metropolis-Hastings sampler, the method with the faster running time; see also Exercise 26 and Fig. 27).

(i) Initialize $x = (\xi_1, \ldots, \xi_{n^2})$ (deterministically or randomly).

(ii) Draw k and u from $U(\{1, \ldots, n^2\})$ and $U(0,1)$, respectively.

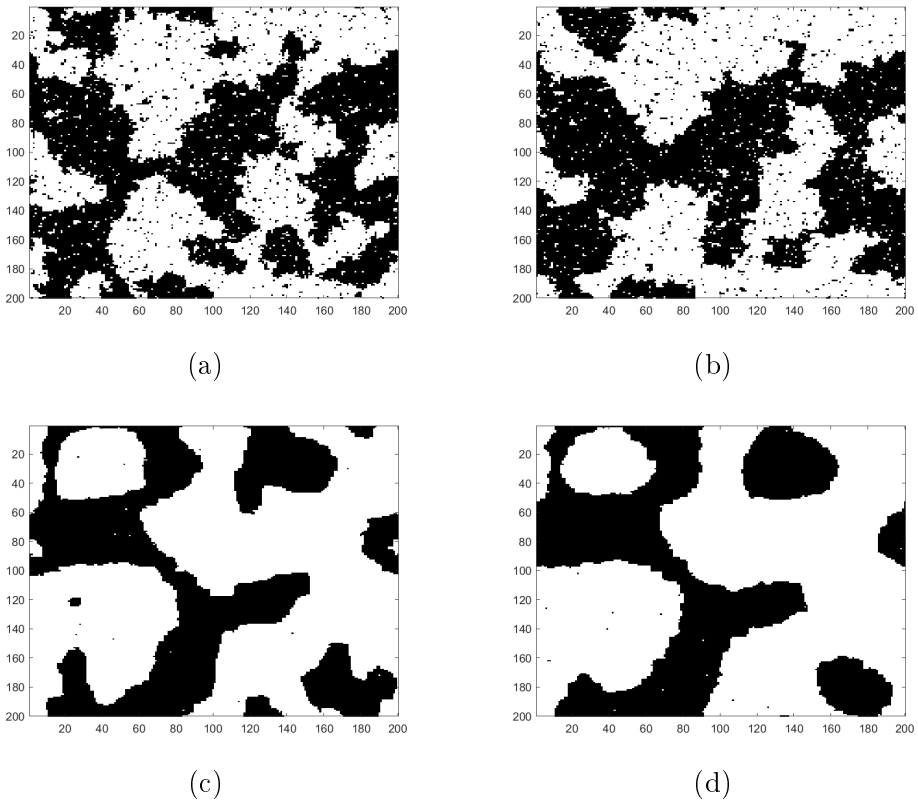
(iii) For $j \in \{0,1\}$ compute n_j, the number of neighbors of the kth pixel having color j.

(iv) ***If***

$$u \leq \frac{\exp(\beta n_1)}{\exp(\beta n_0) + \exp(\beta n_1)}$$

then set $\xi_k = 1$ ***otherwise*** *set $\xi_k = 0$.*

(v) Got to step (ii).



(a) (b) (c) (d)

Figure 26: Typical outputs for the Gibbs sampler on the 200×200 torus, randomly initialized. (a) $\beta = 1$ output after 10^7 iterations, (b) output after the next 10^7 iterations. (c) and (d) the same as for (a) and (b) but with $\beta = 2$. (Computation times for generating any of these images were around 25s.)

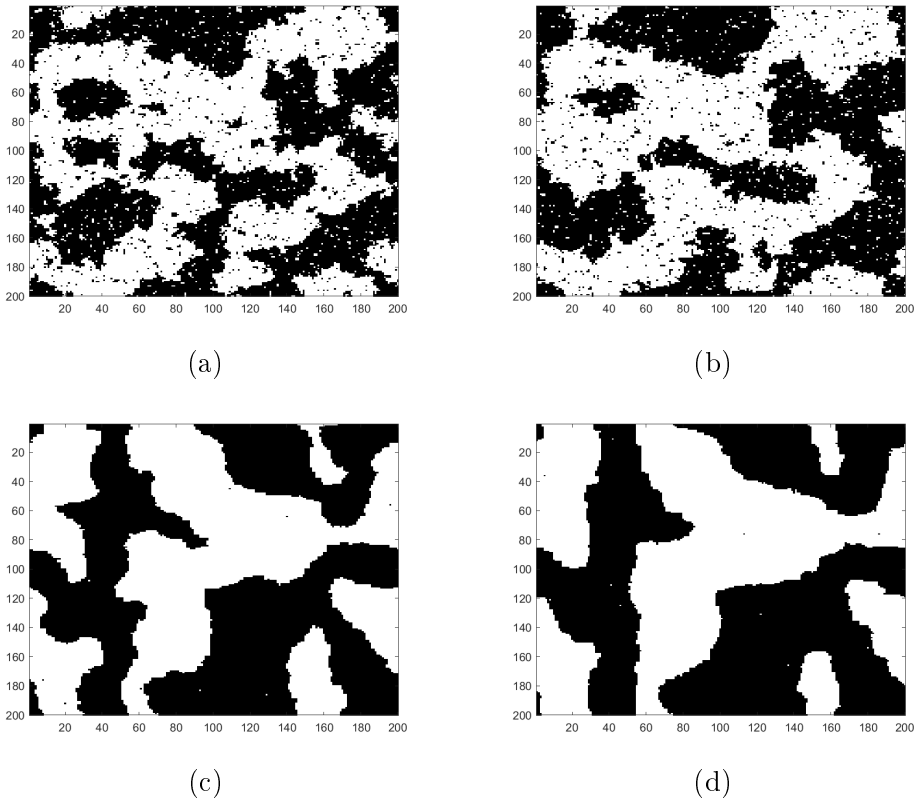
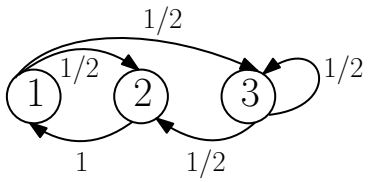


Figure 27: Typical outputs for the Metropolis sampler (see Exercise 26) on the 200×200 torus, randomly initialized. (a) $\beta = 1$ output after 10^7 iterations, (b) output after the next 10^7 iterations. (c) and (d) the same as for (a) and (b) but with $\beta = 2$. (Computation times for generating any of these images were around 35s.)

4.6 Exercises

Exercise 4.1. A homogeneous Markov chain $\{X_t\}_{t\geq 0}$ with state space $\Omega = \{1, 2, 3\}$ has the following transition graph.



(a) Determine its transition matrix.

(b) Is the Markov chain irreducible? Is it aperiodic?

(c) Determine the stationary distribution.

Exercise 4.2. A homogeneous Markov chain $\{X_t\}_{t\geq 0}$ with state space $\Omega = \{1, 2\}$ has the following transition matrix

$$P = \begin{array}{c} \\ 1 \\ 2, \end{array} \begin{array}{c} \begin{array}{cc} 1 & 2 \end{array} \\ \begin{pmatrix} 1-\alpha & \alpha \\ \beta & 1-\beta \end{pmatrix} \end{array}$$

where $\alpha, \beta \in [0, 1]$.

(a) Draw its transition graph.

(b) Give the value of $\Pr(X_1 = 2 \mid X_0 = 1)$.

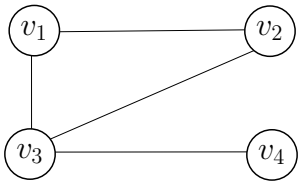
(c) Give the value of $\Pr(X_2 = 2 \mid X_0 = 1)$.

(d) State the global balance equation and the normalization condition.

(e) Determine all stationary distributions. Explain why there is only one stationary distribution when not both α and β are equal to 0.

Exercise 4.3. Show that the Markov chain from Example 4.5. (sampling of independent sets) is irreducible, aperiodic and has the uniform distribution on $\mathcal{I}_G$ as stationary distribution.

Exercise 4.4. Consider the following graph $G = (V, E)$.



(a) List all independent sets of G.

(b) Give a matlab implementation of the algorithm from Example 4.5. for sampling independent sets of G. (Your implementation need not be general, it suffices if it works for this particular G.)

(c) Draw `nTrials` $= 10^4$ random samples from $\mathcal{I}_G$ using your implementation from (b). Plot a histogram of the size distribution of the drawn independent sets (the size of $I \in \mathcal{I}_G$ is the number of vertices in I). What do you observe?

Exercise 4.5. Prove or disprove the following claim:

If the transition matrix P of a Markov chain is symmetric (i.e., $P(x,y) = P(y,x)$ for all $x, y \in \Omega$) then the uniform distribution on Ω is a stationary distribution.

Exercise 4.6. Consider Barker's sampler from the lectures. Show the following:

(a) The Markov chain generated by Barker's sampler has π as stationary distribution.

(b) With $A(x,y)$ and $A_{MH}(x,y)$ denoting the acceptance probability of Barker's and the Metropolis-Hastings sampler, respectively, we have

$$A(x,y) \leq A_{MH}(x,y), \quad \text{for all } x, y \in \Omega.$$

Exercise 4.7. Consider a $2D$ image consisting of an $n \times n$ grid of black and white pixels. Let X_j, $j = 1, \ldots, n^2$, denote the indicator of the jth pixel being white (i.e., $X_j = 1$ if the jth pixel is white, and $X_j = 0$ otherwise). Viewing the pixels as vertices in a graph, the set $N(i)$ of neighbors of the ith pixel are the pixels immediately above, below, to the left, and to the right (except for boundary cases).

A commonly used model for the pdf π of $X = (X_1, \ldots, X_{n^2})$ is

$$\pi(x) = \frac{1}{Z} e^{\frac{\beta}{2} \sum_{i=1}^{n^2} \sum_{j \in N(i)} \delta_{x_i, x_j}},$$

where $\beta > 0$ and

$$\delta_{x_i, x_j} = \begin{cases} 1 & \text{if } x_i = x_j, \\ 0 & \text{otherwise.} \end{cases}$$

With this π neighboring pixels prefer to have the same color. The normalizing constant Z of π is a sum over all 2^{n^2} possible configurations, so it may be very difficult to compute. This motivates the use of MCMC to simulate from the model.

(a) Provide a Metropolis-Hastings algorithm for sampling from π, based on a proposal of selecting uniformly a random pixel k and changing its color from its present color c_old to a new color c_new.

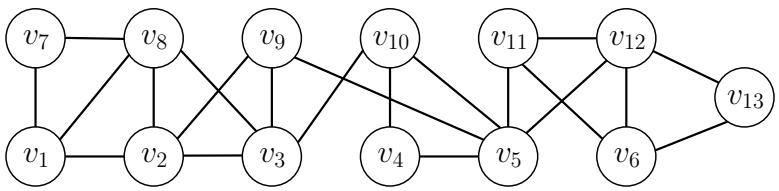
(b) Show that the ratio $\pi(y)/\pi(x)$ in (a) can be expressed as $\exp\left(\beta n_{c\_old} - \beta n_{c\_new}\right)$, where n_{c_old} denotes the number of neighbors of k with color c_old and n_{c_new} denotes the number of neighbors of k with color c_new.

Exercise 4.8. Prove the following claim.

> The (random scan) Gibbs sampler is a special case of the Metropolis-Hastings algorithm where proposals are always accepted. In particular, the stationary distribution of this sampler is as desired.

(Hint: Give the proof for the case that the Gibbs sampler selects the first coordinate. Recall the fact that the joint pdf p of two random variables X and Y satisfies $p_{X,Y}(x,y) = p_{X|Y=y}(y) p_X(x)$ for all $(x,y) \in \Omega$.)

Exercise 4.9. Use simulated annealing to find a maximum matching (a matching of largest size) in the graph from Example 4.7 in the lectures (depicted again below). Show the output of a particular run of your algorithm.



5 Learning theory and methods

Learning theory can be considered as the field devoted to studying the design and analysis of *machine learning* algorithms. The term *machine learning* seems to have been first coined by Arthur Lee Samuel[30], a pioneer in the *artificial intelligence* field, in 1959. The following, a paraphrase of his quote *'Programming computers to learn from experience should eventually eliminate the need for much of this detailed programming effort,'* nicely captures the essence and is therefore cited in many machine learning texts.

> **'Machine learning** is the field of study that gives computers the ability to learn without being explicitly programmed.'
>
> (Arthur L. Samuel, 1959)

5.1 The process of learning

Typically, one distinguishes between the following three main types of learning algorithms.

- **Supervised learning:** A supervised learning algorithm learns from labeled *training data* and, based on this, tries to predict outcomes for unforeseen data. An example for this would be predicting mortalities of COVID-19 based on training data from the past.
- **Unsupervised learning:** In these methods there is *no label* attached to the data, and the task is to identify patterns and/or model the data. A example for this would be the compression of information.
- **Reinforcement learning:** This method falls between the above two methods as there is some form of feedback available (known as *reward signal*) for each predictive step, but there is no label. An example for this would be training an agent to play video games. The reward signal can be the player's score.

Examples of machine learning problems include:

- **Classification**: Classify data into one or more categories (*classes*). For example, identifying in computerized tomography (CT) images whether a patient has a tumor or not.
- **Clustering**: Group a set of data points into clusters, such that points within a cluster have some properties in common. An example is in image segmentation, where the goal is to break up the image into meaningful or perceptually similar regions. (For another example see, e.g., Project 3.)
- **Prediction**: Based on historical data, build models and use them to forecast future values. For example, predicting temperature rises due to global warming based on data from the past.

Examples of important applications where machine learning algorithms are deployed include:

- Optical character recognition (OCR)[31],
- Text or document classification, spam detection[32],
- Speech recognition[33],
- Face recognition[34],
- Fraud detection[35]

[30] (December 5, 1901 – July 29, 1990)
[31] http://human.ait.kyushu-u.ac.jp/publications/PRL2008-Malon.pdf
[32] https://arxiv.org/pdf/1606.01042.pdf
[33] https://www.youtube.com/watch?v=RBgfLvAOrss
[34] https://www.mathworks.com/discovery/face-recognition.html
[35] https://www.fico.com/blogs/5-keys-using-ai-and-machine-learning-fraud-detection

- Language translation[36],
- Games like chess and Go[37],
- Autonomous driving[38],
- Medical diagnosis (decisions about Caesarian sectioning[39] or tumor removal[40]),
- Recommendation systems, search engines[41],
- Representations of polycrystals in materials science[42].

Before we go into further details let us consider two initial examples.

Example 5.1. *The last application mentioned above is the representations of polycrystals in materials science. Fig. 28 is from the paper that can be found at* https://www-m9.ma.tum.de/foswiki/pub/Allgemeines/AndreasAlpersPublications/H1.pdf. *Shown in black in these two images are so-called grain boundaries. These are boundaries of small crystals that make up the whole material, here an aluminum sample. The red and blue boundaries, respectively are the boundaries that one obtains by some specific clustering method (which we call* generalized balance power diagrams*). One observers a relatively good fit, which on the one hand indicates that nature seems to try to find a similar clustering. On the other hand, for storing the clusters one needs only to store a few parameters per grain and not the whole image. Thus, one has a much sparser representation of the data.*

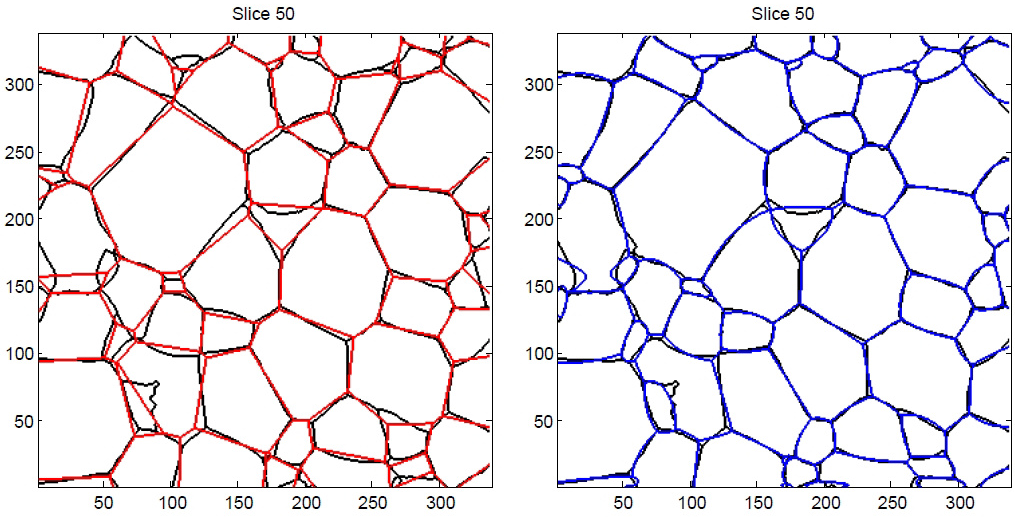


Figure 28

Example 5.2. *Consider the current COVID-19 outbreak. As other pandemics it is expected to have exponential growth. That means that the number $x(t)$ of infected persons at time t follows a function*

$$x(t) = x_0 b^t, \tag{5.1}$$

where x_0 is the number of cases at the beginning, and b is the number of people infected by each infected person.

[36] https://www.youtube.com/watch?v=AIpXjFwVdIE
[37] https://ai.googleblog.com/2016/01/alphago-mastering-ancient-game-of-go.html
[38] https://iopscience.iop.org/article/10.1088/1757-899X/662/4/042006/pdf
[39] https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8519731
[40] https://www.nature.com/articles/s41598-019-48738-5
[41] https://www.seroundtable.com/google-explains-machine-learning-search-28697.html
[42] https://www-m9.ma.tum.de/foswiki/pub/Allgemeines/AndreasAlpersPublications/H1.pdf

The first two UK cases appear on January 31st, 2020, which for us is $t = 0$. Let us, for the sake of exposition, consider the available data (source: Public Health England[43]) up until March 18th, 2020 (i.e., up to $t = 48$):

t	0	9	10	13	24	28	29	30	31	32	33	34	35	36
$x(t)$	2	3	4	8	9	13	19	23	35	40	51	85	114	160

t	37	38	39	40	41	42	43	44	45	46	47	48
$x(t)$	206	271	321	373	456	590	797	1061	1391	1543	1950	2626

Taking logarithms in (5.1) we obtain

$$\ln(x(t)) = \ln(x_0) + \ln(b)t,$$

which is a linear function in t. Hence, we can apply linear regression.

Using the matlab command

```
[r,m,b] = regression(t,log(xt),'one')
```

we obtain

$$\ln(x(t)) = -0.6848 + 0.1618t$$

and hence (taking exponential values)

$$x(t) = 0.5042 + 1.1756^t.$$

A plot is shown in Fig. 29.

One might ask the question when are or have been 1 million persons infected, i.e., for which t_1 do we have $10^6 = 0.5042 + 1.1756^t$? Solving this equation we see that this would have happened for $t = 86$, which is April 25th, 2020.[44]
As an exercise you might want to repeat these calculations for the data available for China, Germany, or any other country of your choice.

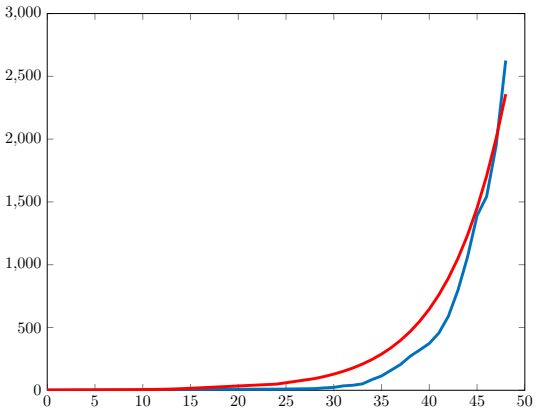


Figure 29: Number $x(t)$ of COVID-19 infected people for $t = 0$ (January 31st, 2020) to $t = 48$ (March 18th, 2020) shown in blue. In red the predicted numbers obtained by linear regression.

[43] https://www.gov.uk/government/publications/covid-19-track-coronavirus-cases

[44] As we know it came somewhat differently as different testing policies and national lockdown measures were imposed. The number of 1 million infected UK persons was surpassed on October 31st, 2020.

5.2 Supervised learning

In supervised learning the training data comes in pairs of inputs (x_i, y_i), where $x_i \in \mathcal{R}^d$ is the input instance and $y_i \in \mathcal{C}$ its label. The space $\mathcal{R}^d$ is the feature space — in our case we always consider $\mathcal{R}^d = \mathbb{R}^d$. The space $\mathcal{C}$ is the label space. Several examples are shown in Table 5.

Task	Label space	Example
Binary classification	$\mathcal{C} = \{0, 1\}$ or $\mathcal{C} = \{+, -\}$	A CT image contains a tumor or not
Multi-class classification	$\mathcal{C} = \{1, \ldots, k\}$ with $k > 2$	Speech and character recognition
Regression	$\mathcal{C} = \mathbb{R}$	Prediction of temperature rises

Table 5: Examples of label spaces.

We focus in the following on **classification problems**. In such problems the data points (x_i, y_i) are drawn from some (unknown) distribution. The goal is to learn a function $h : \mathcal{R}^d \to \mathcal{C}$ such that for a new pair (x, y), we have $h(x) = y$ (or $h(x) \approx y$) with high probability.

Examples of supervised learning techniques are *support vector machines, naïve Bayes classifiers, decision trees,* and *neural networks*. We will discuss the basics in the following sections. Our first section is, however, devoted to a theoretical aspect, namely that of measuring the capacity (complexity, expressive power, richness, or flexibility) of a space of functions that can be learned by a classification algorithm.

5.2.1 Vapnik–Chervonenkis dimension

The **Vapnik–Chervonenkis (VC) dimension** was originally introduced by V. Vapnik[45] and A. Chervonenkis[46] in 1971 (see [13]). Roughly speaking, the VC-dimension of a function (i.e. hypothesis) class is the maximum number of data points for which, no matter how we label them (by 0/1), there is always a hypothesis in the class which perfectly explains the labeling. Among others, the VC-dimension of a hypothesis space therefore describes its ability to correctly represent the training data. (Another use of the VC-dimension is, for instance, that it can be used to upper bound the estimation error of a given classifier).

Let us introduce the VC-dimension formally[47]. Each hypothesis $h : \mathcal{R}^d \to \{0, 1\}$ can naturally be viewed as a subset of $\mathcal{R}^d$, where $h = \{\xi : h(\xi) = 1\}$. Therefore, we can refer to h as a binary function and as a subset interchangeably. For any finite subset $S \subseteq \mathcal{R}^d$, let $\Pi_{\mathcal{H}}(S) = \{h \cap S : h \in \mathcal{H}\}$. This is called the *projection* of $\mathcal{H}$ onto S.

Definition 5.1. *The set S is **shattered** by the function class $\mathcal{H}$ if $\Pi_{\mathcal{H}}(S)$ contains all the subsets of S, i.e., $|\Pi_{\mathcal{H}}(S)| = 2^{|S|}$.*

In other words, the set S is shattered by $\mathcal{H}$ if, no matter how we assign 0/1-labels to points in S, there is always a hypothesis in $\mathcal{H}$ that 'explains' the labeling perfectly.

Definition 5.2. *The **VC-dimension** $\mathrm{VCD}(\mathcal{H})$ of a function class (or hypothesis class) $\mathcal{H}$ is the maximum size of a subset of $\mathcal{R}^d$ shattered by $\mathcal{H}$.*

In other words, if $\mathrm{VCD}(\mathcal{H}) = d$ then $\mathcal{H}$ *can* shatter some d points but it *cannot* shatter any $d + 1$ points. Let us consider two examples and the important class $\mathcal{H}$ consisting of halfspaces in $\mathbb{R}^2$.

Example 5.3. *Consider learning (positive) rays on the real line. The hypothesis class $\mathcal{H}$ consists of all rays of the form $\{x \in \mathbb{R} : x > a\}$, for some $a \in \mathbb{R}$. A ray (hypothesis) $\{x \in \mathbb{R} : x > a\} \in \mathcal{H}$ is interpreted as a classifier that identifies a point $p \in \mathbb{R}$ as being in class S if $p > a$ and identifies x as not being in class S if $p \leq a$.*

[45] Vladimir N. Vapnik (born 6 December 1936).

[46] Alexey Chervonenkis (7 September 1938 - 22 September 2014).

[47] Note that the VC-dimension is defined for spaces of binary functions (functions to $\{0, 1\}$). Generalizations for spaces of non-binary functions have later been suggested in the literature.

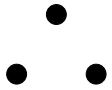
(a) *Can a set containing a single point in* $\mathbb{R}$ *be shattered by* $\mathcal{H}$?
Well, the answer is obviously yes (since we can have a ray that contains that point and we can have a ray that does not contain this point).

(b) *We claim that no sets of two points in* $\mathbb{R}$ *can be shattered by* $\mathcal{H}$.
Suppose, we have a set S *of two points* $p_1 < p_2 \in \mathbb{R}$. *Any ray containing* p_1 *also contains* p_2. *We can therefore never obtain the set* $\{p_1\}$ *as projection of some ray onto* S.

(c) *Can we now say something about the VC-dimension* $\mathcal{H}$?
Based on (a) we have $\mathrm{VCD}(\mathcal{H}) \geq 1$, *and based on (b) we have* $\mathrm{VCD}(\mathcal{H}) \leq 1$. *Hence,* $\mathrm{VCD}(\mathcal{H}) = 1$.

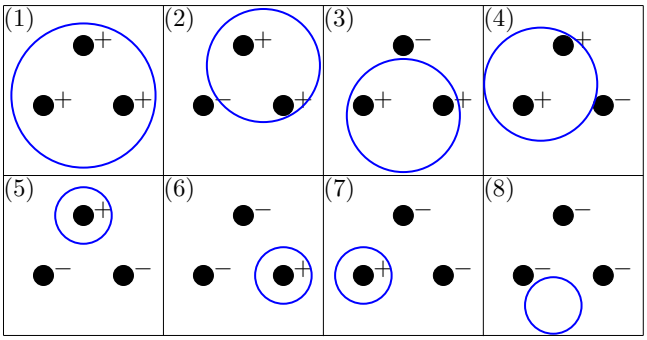
Example 5.4. *Consider learning closed balls in* $\mathbb{R}^2$. *The hypothesis class* $\mathcal{H}$ *consists of all balls* $\mathbb{B}(p,r) = \{(x_1,x_2) \in \mathbb{R}^2 \ : \ (x_1-p_1)^2 + (x_2-p_2)^2 \leq r^2\}$, *with* $p = (p_1,p_2) \in \mathbb{R}^2$ *and* $r > 0$. *A ball (hypothesis)* $\mathbb{B}(p,r) \in \mathcal{H}$ *is interpreted as a classifier that identifies a point* x *as being in class* S *if* $x \in \mathbb{B}(p,r)$ *and identifies* x *as not being in class* S *if* $x \notin \mathbb{B}(p,r)$.

(a) *Let us show that this set*



of three points can be shattered by $\mathcal{H}$.

First, we observe that there are the following eight combinations for subsets S *of the set of three points (the label '+' indicates that the respective point should be included in* S, *while '−' indicates that it should not be included in* S). *In each case a ball is sketched that contains the respective* S. *Hence the set of three points can be scattered by* $\mathcal{H}$.



(b) *Note that the result from (a) implies, by the definition of VC-dimension, that* $\mathrm{VCD}(\mathcal{H}) \geq 3$.

(c) *For showing* $\mathrm{VCD}(\mathcal{H}) = 3$ *we would need to show that no four points can be shattered by* $\mathcal{H}$.

We do not show this here. We just remark that no set of three points on a line can be shattered by $\mathcal{H}$ *(because balls are convex, hence if* S *contains the points* x_1 *and* x_3 *is also needs to contain any point* x_2 *lying on the line segment* $\overline{x_1x_3}$*).*

Theorem 5.1. *For the hypothesis class* $\mathcal{H}$ *consisting of halfplanes in* $\mathbb{R}^2$ *it holds that* $\mathrm{VCD}(\mathcal{H}) = 3$.

Proof. Let us first prove that $\mathrm{VCD}(\mathcal{H}) \geq 3$ by providing a specific set S of three points for which we show that we can obtain every subset of S as a projection of some halfplane onto S.

Consider three points (not all lying on a line). There are eight possible labelings, and for each we can find a halfplane containing only the positively labeled points; see Fig. 30. Hence, $\mathrm{VCD}(\mathcal{H}) \geq 3$.

To see that no set of four points can be shattered, we consider three cases. In the first case (shown in Fig. 31(a)), all four points lie on the convex hull defined by the four points. In this case, if we label one 'diagonal' pair positive and the other 'diagonal' pair negative as shown in Fig. 31(a), no halfspace satisfies this labeling. In the second case (shown in Fig. 31(b)), three of the four points define the convex hull of the four points, and if we label the interior point negative and the hull points positive,

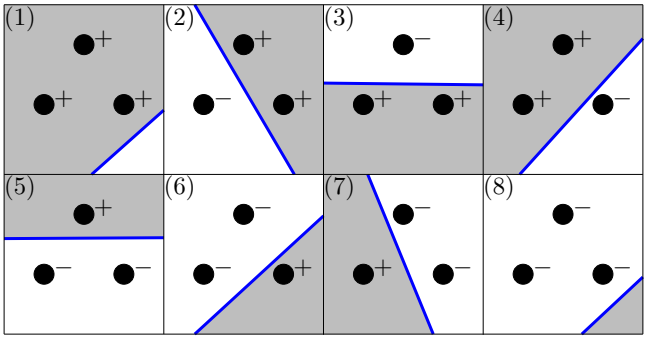


Figure 30: The eight possible labelings and a corresponding halfplane (gray shaded area bounded by the blue line) that contains only the positively labeled points.

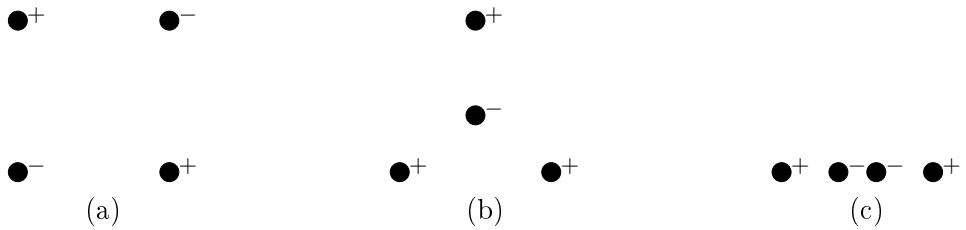


Figure 31: Examples of four points and their labelings. No halfplanes exist that satisfy these labelings.

again no halfspace can satisfy the labeling. In the third case (shown in Fig. 31(c)), two of the four points define the convex hull of the four points, and if we label the interior point negative and the hull points positive, again no halfspace can satisfy the labeling. □

In general it can be shown that $\mathrm{VCD}(\mathcal{H}) = d + 1$ for halfplanes in $\mathbb{R}^d$. Proving this is outside the scope of this text (but a proof can be given in a rather elegant and elementary manner relying on the so-called *Radon's theorem*).

5.2.2 Support vector machines

Support vector machines (SVMs) are supervised learning algorithms mainly used for classification. They find a hyperplane (or sets of hyperplanes) in a high- or infinite dimensional space, which can then be used for classification (or related tasks).

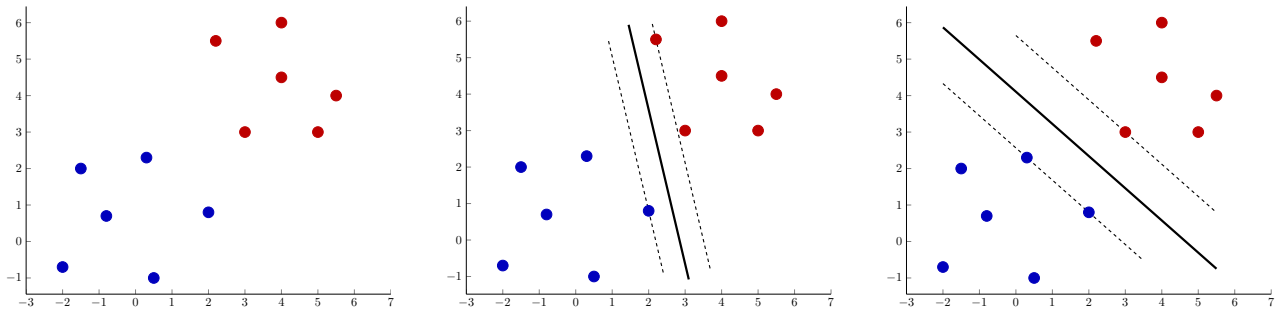
Originally proposed by Vapnik (and published by Vapnik and Lerner[48]) in 1963 for linear classification problems, we remark that the original SVM concept has been generalized over the years in numerous ways. It is, for instance, possible to use SVMs for non-linear classification (which is accomplished by mapping the data in a non-linear way to a higher dimensional space).

We focus on the original SVM concept. The **binary SVM problem** is the following.

Binary SVM problem:
Given training data $x_1, \ldots, x_m \in \mathbb{R}^d$ with labels $y_1, \ldots, y_m \in \{\pm 1\}$. Find a separating hyperplane that has *maximum margin*, i.e., the maximum distance between data points of both classes.

In classification problems there are usually many separating hyperplanes (see Fig. 32(b) and (c)). Why could it be a good idea to find a separating hyperplane that maximizes the margin? The intuition behind this is that points near the decision boundary are often misclassified (there is an almost 50% chance that the classifier decides either way). So, insisting on a large margin can potentially minimize misclassification. This can be a good idea if nothing else is known about the data. (For a more model dependent choice of decision boundaries, see the later section on naïve Bayes classifiers.)

[48] freely available at http://web.cs.iastate.edu/~cs573x/vapnik-portraits1963.pdf.



(a) (b) (c)

Figure 32: Binary SVM problem. (a) training data with two labels (red and blue), (b) a separating hyperplane (solid black, dashed lines indicating the margin), (c) the optimal separating hyperplane (solid black) maximizing the margin (indicated by the dashed lines).

We can specify a hyperplane by two parameters, $w \in \mathbb{R}^d$ and $\beta \in \mathbb{R}$. The hyperplane $H_{w,\beta}$ defined by these two parameters is then

$$H_{w,\beta} = \{x \in \mathbb{R}^d \ : \ w^T x + \beta = 0\}.$$

See Fig. 33. The points $x_i \in \mathbb{R}^d$ with $w^T x_i + \beta \geq 0$ lie on one side of the hyperplane, the points $x_i \in \mathbb{R}^d$ with $w^T x_i + \beta \leq 0$ lie on the other side of the hyperplane (points satisfying the inequality with equality are lying on both sides).

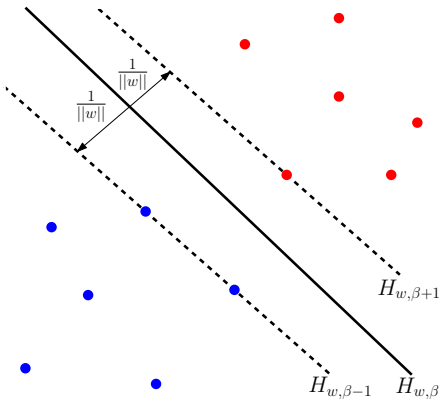


Figure 33: Maximum margin separarting hyperplanes.

Consider now the parallel hyperplanes $H_{w,\beta+1}$ and $H_{w,\beta-1}$. What we want to have is that $H_{w,\beta+1}$ and $H_{w,\beta-1}$ are separating and their distance (which can be shown to be $2/||w||$; see also Fig. 33) should be maximal[49]. This can be formulated as

$$\max_{w,\beta} \frac{2}{||w||}$$

$$\begin{aligned} \text{subject to} \quad & w^T x_i \geq \beta + 1, \quad \text{for all } i \text{ with } y_i = +1, \\ & w^T x_i \leq \beta - 1, \quad \text{for all } i \text{ with } y_i = -1, \end{aligned}$$

which has the same optimal w and β as the optimization problem

$$\min_{w,\beta} \frac{1}{2}||w||^2 \qquad \text{subject to} \qquad y_i(w^T x_i - \beta) \geq 1 \quad \text{for all } 1 \leq i \leq m.$$

[49] There is nothing special in considering $\beta + 1$ and $\beta - 1$ in the definition of $H_{w,\beta+1}$ and $H_{w,\beta-1}$. Instead one could have also considered $H_{w,\beta+\varepsilon}$ and $H_{w,\beta-\varepsilon}$ for any of your favorite choice of $\varepsilon \neq 0$, because $H_{w,\beta+1} = H_{w',\beta'+\varepsilon}$ and $H_{w,\beta-1} = H_{w',\beta'-\varepsilon}$ with $w' = \varepsilon w$ and $\beta' = \varepsilon\beta$.

In the latter form, the binary SVM problem is a well-studied optimization problem. It has a strictly convex objective function and (affinely) linear constraints. Due to the strict convexity it can be shown that the minimum is unique, and this minimum can be found by variety of well-established algorithms (in matlab, you can use the command `quadprog`).

Fig. 32(c) shows the maximum margin separating hyperplane for the data shown in Fig. 32(a).

Having found the maximum margin separating hyperplane, any new data $x \in \mathbb{R}^d$ will subsequently be classified according to this hyperplane, i.e., by

$$y = \begin{cases} +1 & \text{if } w^T x \geq \beta, \\ -1 & \text{otherwise.} \end{cases}$$

5.2.3 Naïve Bayes classifier

Sometimes we have some prior knowledge about the data and we want or need to include this. In **Bayesian learning** – and we discuss here the special case of naïve Bayes classifiers – prior knowledge is provided by asserting (a) a prior probability for each class labeling and (b) a probability distribution over observed data for each possible labeling.

Naïve Bayes is a statistical classification technique based on Bayes Theorem (see Section 1.3). A naïve Bayes classifier assumes that any of the features $\xi_1, \ldots, \xi_d$ of a feature vector $x = (\xi_1, \ldots, \xi_d)^T$ are independent given the label y. This is usually an oversimplification (hence the term *naïve*), but it simplifies the computations considerably.

Bayes theorem and the independence give

$$\begin{aligned} \Pr(y = y^* \mid \xi_1 = \xi_1^*, \ldots, \xi_d = \xi_d^*) &= \frac{\Pr(\xi_1 = \xi_1^*, \ldots, \xi_d = \xi_d^* \mid y = y^*)\Pr(y = y^*)}{\Pr(\xi_1 = \xi_1^*, \ldots, \xi_d = \xi_d^*)} \\ &= \frac{\Pr(\xi_1 = \xi_1^* \mid y = y^*) \cdot \ldots \cdot \Pr(\xi_d = \xi_d^* \mid y = y^*)\Pr(y = y^*)}{\Pr(\xi_1 = \xi_1^*, \ldots, \xi_d = \xi_d^*).} \end{aligned} \tag{5.2}$$

The terms in this expression are typically referred to as:

- $\Pr(y = y^* \mid \xi_1 = \xi_1^*, \ldots, \xi_d = \xi_d^*)$ is the so-called *posterior probability* of the label y^* given the feature vector $x^* = (\xi_1^*, \ldots, \xi_d^*)^T$.
- $\Pr(y = y^*)$ is the *prior probability* of the label y^*.
- $\Pr(\xi_i = \xi_i^* \mid y = y^*)$ is the *likelihood*, which is the probability of the feature ξ_i^* given the label y^*.
- $\Pr(\xi_1 = \xi_1^*, \ldots, \xi_d = \xi_d^*)$ is the *prior probability* of the feature vector x^*.

Now the idea is to find the label that maximizes the posterior probability, i.e., the probability of the observed features.

Naïve Bayes:
Given a vector $x^* = (\xi_1^*, \ldots, \xi_d^*)^T$ of features, the label $\hat{y}$ that will be assigned to x^* will be the $y^* \in L$ that maximizes $\Pr(y = y^* \mid \xi_1 = \xi_1^*, \ldots, \xi_d = \xi_d^*)$ in (5.2).

What we need for naïve Bayes is therefore the likelihood of each feature and the prior probability of each label. The prior probability of the feature vector is not needed, since for all labels it will be the same constant factor.

Unless nothing else is known, one usually assumes that the prior probability of each label is uniformly distributed. For the likelihood one often assumes that the ξ_i feature associated with each label $y^* \in L$ is distributed according to a normal distribution (of course, this assumes that the ξ_i feature is continuous). In other words,

$$\Pr(\xi_i = \xi_i^* \mid y = y^*) = \frac{1}{\sigma_{y^*}\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\xi_i^* - \mu_{y^*}}{\sigma_{y^*}}\right)^2},$$

where μ_{y^*} and $\sigma^2_{y^*}$ denote the mean and, respectively, variance of the ξ_i feature in the training data that is associated to label y^*. Such Bayes classifier is often referred to as **Gaussian naïve Bayes classifier.**

Different than in the SVM case discussed previously, decision boundaries for Bayes classifiers need not be linear (they are generally quadratic).

Example 5.5. *Suppose, we have the training data as considered in Section 5.2.2; Table 6(a) gives the numerical values.*

ξ_1	ξ_2	label
-1.5	2	-1
-0.8	0.7	-1
0.5	-1	-1
-2	-0.7	-1
2	0.8	-1
0.3	2.3	-1
3	3	+1
4	6	+1
5	3	+1
4	4.5	+1
5.5	4	+1
2.2	5.5	+1

(a)

feature	label	mean	variance
ξ_1	-1	-0.25	2.17
ξ_1	+1	3.95	1.50
ξ_2	-1	0.68	1.82
ξ_2	+1	4.33	1.57

(b)

Table 6: (a) Training data (as in Fig. 32), (b) mean and variance of the training data.

Using Gaussian naïve Bayes we want to classify $x^* = (\xi_1^*, \xi_2^*)^T = (1, 4)^T$.

First we compute the mean and variance of the training data separately for each label. The results are shown in Table 6(b).
For $y^* = -1$ *we obtain*

$$\Pr(y = -1) = 6/12 = 0.5,$$
$$\Pr(\xi_1 = \xi_1^* \mid y = -1) = \frac{1}{2.17\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{1+2.5}{2.17}\right)^2} \approx 0.0500,$$
$$\Pr(\xi_2 = \xi_2^* \mid y = -1) = \frac{1}{1.82\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{4-0.68}{1.82}\right)^2} \approx 0.0415.$$

Hence

$$\Pr(y = -1 \mid \xi_1 = \xi_1^*, \xi_2 = \xi_2^*) = 0.0010/\Pr(\xi_1 = \xi_1^*, \xi_2 = \xi_2^*).$$

For $y^* = +1$ *we obtain*

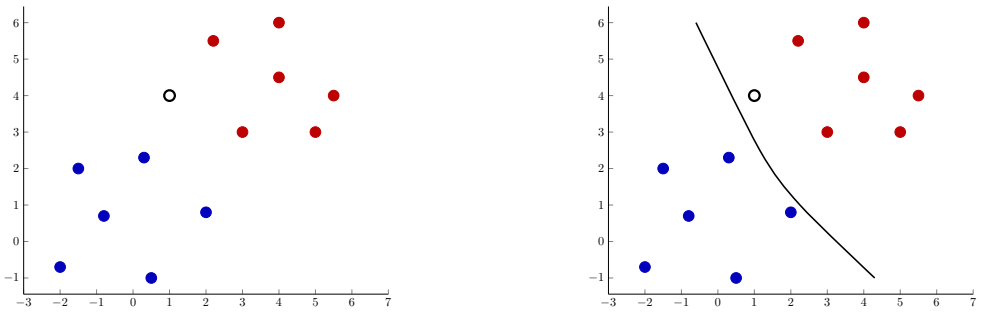
$$\Pr(y = +1) = 6/12 = 0.5,$$
$$\Pr(\xi_1 = \xi_1^* \mid y = +1) = \frac{1}{1.5\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{1-3.95}{1.5}\right)^2} \approx 0.0385,$$
$$\Pr(\xi_2 = \xi_2^* \mid y = +1) = \frac{1}{1.57\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{4-4.33}{1.57}\right)^2} \approx 0.2486.$$

Hence

$$\Pr(y = +1 \mid \xi_1 = \xi_1^*, \xi_2 = \xi_2^*) = 0.0048/\Pr(\xi_1 = \xi_1^*, \xi_2 = \xi_2^*).$$

As $\Pr(y = +1 \mid \xi_1 = \xi_1^*, \xi_2 = \xi_2^*) > \Pr(y = -1 \mid \xi_1 = \xi_1^*, \xi_2 = \xi_2^*)$ *we therefore assign* x^* *to class* $\hat{y} = +1$.

Fig. 34 shows the classification and the respective (non-linear, see Exercise 5.6) decision boundary.



(a) (b)

Figure 34: Gaussian naïve Bayes. Classifying the point $(1, 4)$ (shown as black circle). The decision boundary is shown in black.

5.2.4 Decision trees

A rather simple classification technique is the so-called **decision tree classifier**. Basically, the idea is to ask a series of carefully crafted questions. The series of questions and their possible answers can be organized into a hierarchical structure called a decision tree.

Example 5.6. *Figure 35 shows an example of the decision tree for a mammal classification problem adapted from [14], where the task is to decide whether a newly discovered species is a mammal or a non-mammal.*

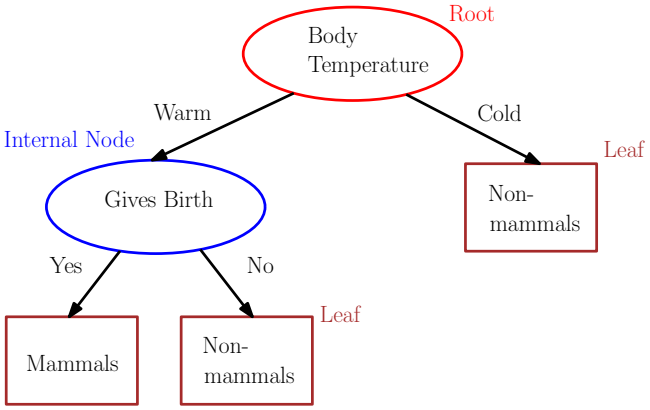


Figure 35: A decision tree for the mammal classification problem (adapted from [14]).

The first question is whether the species is cold- or warm-blooded. If it is cold-blooded, then it is definitely not a mammal. Otherwise, we ask whether is gives birth (as oppose to laying eggs). If the answer is yes it is a mammal otherwise it is a non-mammal.

So, suppose we want to classify the flamingo species based on the data shown in Fig. 36. Following the path through the tree (see dashed lines) we would classify flamingos as non-mammals.

Decision trees have three types of nodes: (a) a root note, which has no incoming arcs, (b) internal nodes, each of which has exactly one incoming arc and two or more outgoing arcs, and (c) leaf nodes, which have no outgoing arcs. The root and internal nodes represent the questions, the leaf nodes represent the corresponding labels (note that the same label can be present at different leaves).

So, how does one construct (learn) such a tree? Of course, we want to construct the tree from trainind data. The goal should be to find a decision tree that classifies all the training data correctly. It is not difficult to see that there are typically many different decision trees that achieve this goal. Thus, we might ask for an *optimal* decision tree — optimal in the sense that it minimizes the expected number of tests required to identify the unlabeled object. Finding such an optimal decision tree is,

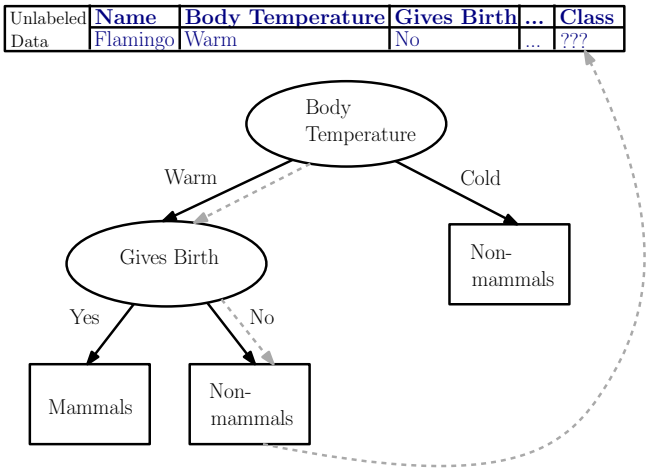


Figure 36: Classifying an unlabeled vertebrate (adapted from [14]).

however, an $\mathbb{NP}$-hard problem[50]. Therefore one usually tries to construct near-optimal decision trees using some heuristics.

Learning of a decision tree:
The general idea is to choose a feature on which to split the data. This will be the root node. Then in each of the child nodes one repeats the process (splits the data on a chosen feature).

Example 5.7. *Let us consider learning the decision tree for classifying species into mammals and non-mammals based on the data shown in Table 7.*

Vertebrate Name	Body Temperature	Gives Birth	Aerial Creature	Has Legs	Hiber-nates	Class Label
python	cold-blooded	no	no	no	yes	non-mammal
salmon	cold-blooded	no	no	no	no	non-mammal
eel	cold-blooded	no	no	no	no	non-mammal
frog	cold-blooded	no	no	yes	yes	non-mammal
komodo dragon	cold-blooded	no	no	yes	no	non-mammal
leopard shark	cold-blooded	yes	no	no	no	non-mammal
turtle	cold-blooded	no	no	yes	no	non-mammal
human	warm-blooded	yes	no	yes	no	mammal
whale	warm-blooded	yes	no	no	no	mammal
bat	warm-blooded	yes	yes	yes	yes	mammal
pigeon	warm-blooded	no	yes	yes	no	non-mammal
cat	warm-blooded	yes	no	yes	no	mammal
penguin	warm-blooded	no	no	yes	no	non-mammal
porcupine	warm-blooded	yes	no	yes	yes	mammal

Table 7: Training data (adapted from [14]) for a vertebrate classification problem.

Which of the features 'body temperature,' 'gives birth,' 'aerial creature,' 'has legs,' and 'hibernates,' should we choose as root node? The answer is that there are different rules around, each constituting a different algorithm. However, we do not go into the details here any further.

[50] See the short 1976 paper of Hyafil and Rivest, freely available at https://people.csail.mit.edu/rivest/HyafilRivest-ConstructingOptimalBinaryDecisionTreesIsNPComplete.pdf.

Let us assume that we choose 'body temperature' as root node. Then we have the two classes

$$C_1 = \{\text{python, salmon, eel, frog, komodo dragon, leopard shark, turtle}\},$$
$$C_2 = \{\text{human, whale, bat, pigeon, cat, penguin, porcupine}\}.$$

All elements of C_1 are non-mammals, hence we we do not split on this node any further. It becomes a leaf.

We need to split C_2 further. If we split on feature 'Gives birth' we get the two classes

$$C_3 = \{\text{human, whale, bat, cat, porcupine}\},$$
$$C_4 = \{\text{pigeon, penguin}\},$$

each of which contain only members of one of the class labels (either mammals or non-mammals). Hence we do not need to split any further. The two nodes become leaf nodes, and we have completed constructing the decision tree. The result is shown in Fig. 35.

Could we have chosen a different feature to split, say, class C_2? In principle, yes. But the resulting decision tree would have contained more levels. Splitting on 'aerial creature' would have resulted in two classes both of which containing both mammals and non-mammals; splitting on 'has legs' would have resulted in a class containing both mammals and non-mammals; splitting on 'hibernates' would have resulted in a class containing both mammals and non-mammals; in all cases one would have needed to split further.

5.2.5 Neural networks

A major class of supervised learning techniques uses *neural networks* (also called *artificial neural networks*). We will discuss them briefly in the later Section 5.5. Learning via neural networks that contain large numbers of hidden layers constitutes the field of so-called *deep learning*.

5.3 Unsupervised learning

In this section we discuss *unsupervised learning* (sometimes paraphrased as 'learning without a teacher'). The data consists of a set $X = \{x_1, \ldots, x_m\}$ of m observations, each being a random d-dimensional vector having a joint pdf $P(X)$. The goal is to directly infer the properties of this pdf without any help of a 'supervisor' that could provide correct answers for each observation.

5.3.1 Clustering

In **clustering** the goal is to divide the observations into groups (*clusters*) so that the pairwise dissimilarities between those assigned to the same cluster tend to be smaller than those in different clusters.

There are a number of different clustering algorithms in the literature (and the development is still ongoing). One of the most popular clustering methods is the so-called ***k*-means algorithm**[51]. The algorithm looks for a fixed number k (hence the name k-means) of clusters in a given data set.

Let us start by stating the algorithm. Then we consider examples followed by a brief discussion of theoretical aspects.

[51] The idea behind this clustering method seems to trace back to H. Steinhaus (January 14, 1887 - February 25, 1972). The standard algorithm was first proposed by Stuart Lloyd of Bell Labs in 1957, though it was not published as a journal article until 1982.

Given $k \in \mathbb{N}$, a set of *data points* $X = \{x_1, \dots, x_m\} \subseteq \mathbb{R}^d$, and *sites* $S = \{s_1, \dots, s_k\} \subseteq \mathbb{R}^d$ the ***k-means algorithm*** proceeds as follows.

(a) Partition X into clusters $C_1, \dots, C_k$ by assigning $x_j \in X$ to the cluster C_i of a closest site $s_i \in S$.

(b) Update each site s_i as the *centroid* of cluster C_i
(if $C_i = \emptyset$, choose $s_i = x_l$ for a random $l \leq n$ with $x_l \neq s_j$ for all $j \leq k$).

(c) If a stopping criterion is met `stop` and return the current assignment and sites, otherwise goto step (a).

Clearly, we need to specify several points in the above algorithm:

- What do we mean by '*closest*' site? Here, unless stated otherwise, we measure distances in the Euclidean norm[52], i.e., we say that x_j is *closest* to cluster C_i if

$$||x_j - s_i|| \leq ||x_j - s_l|| \qquad \text{for all } l \neq i.$$

- What do we mean by '*centroid of cluster* C_i?' Suppose the r data points $x_{i_1}, \dots, x_{i_r}$ are assigned to cluster C_i. Then the centroid of C_i is defined as

$$\frac{1}{r} \sum_{l=1}^{r} x_{i_l}.$$

- What is a possible *stopping criterion*? We use the following: From one to the next iteration the (average) *sum of squared error (SSE)*

$$\frac{1}{m} \sum_{i=1}^{k} \sum_{x_j \in C_i} ||x_j - s_i||^2 \qquad \text{does not decrease.} \tag{5.3}$$

(We note that, in practice, one also often stops when a maximum number of iterations `MaxIt` is reached or the centroids are not changing anymore.)

Example 5.8. *Consider the set*

$$X = \{(-10, 1), (-8, 3), (-6, 2), (3, -1), (5, -3), (6, -2), (9, 3), (10, 7), (11, 5), (-5, -3)\}$$

of 10 data points as shown in Fig. 37(a); the sites $(-7, 0)$, $(-5, 0)$, $(-2, 6)$, $(0, 0)$ *are depicted as 'x.' Fig. (b)-(d) show the results of k-means for* $k = 3$ *after Iteration 1,2, and 3. The algorithm has converged after 3 iterations, resulting in the sites* $(-8, 2)$, $(-5, -3)$, $(10, 5)$, $(14/3, -2)$.

We now want to prove that k-means is converging in a finite number of steps. To this end, we first need to prove the following lemma.

Lemma 5.1. *Let* $x_1, \dots, x_m \in \mathbb{R}^d$. *The sum of squared distances of the* x_i *to a point* $p \in \mathbb{R}^d$ *is minimized when* p *is the centroid, i.e., if* $p = \frac{1}{m} \sum_{i=1}^{m} x_i$.

Proof. Let $c = \frac{1}{m} \sum_{i=1}^{m} x_i$. Then,

$$\sum_{i=1}^{m} ||x_i - p||^2 = \sum_{i=1}^{m} ||x_i - c + c - p||^2 = \sum_{i=1}^{m} (x_i - c + c - p)^T (x_i - c + c - p)$$

$$= \sum_{i=1}^{m} ||x_i - c||^2 + 2(c - p)^T \sum_{i=1}^{m} (x_i - c) + m||c - p||^2,$$

[52] Recall, the Euclidean distance of a vector $v = (v_1, \dots, v_d)^T \in \mathbb{R}^d$ is $||(v_1, \dots, v_d)|| = \sqrt{v_1^2 + \dots + v_d^2}$.

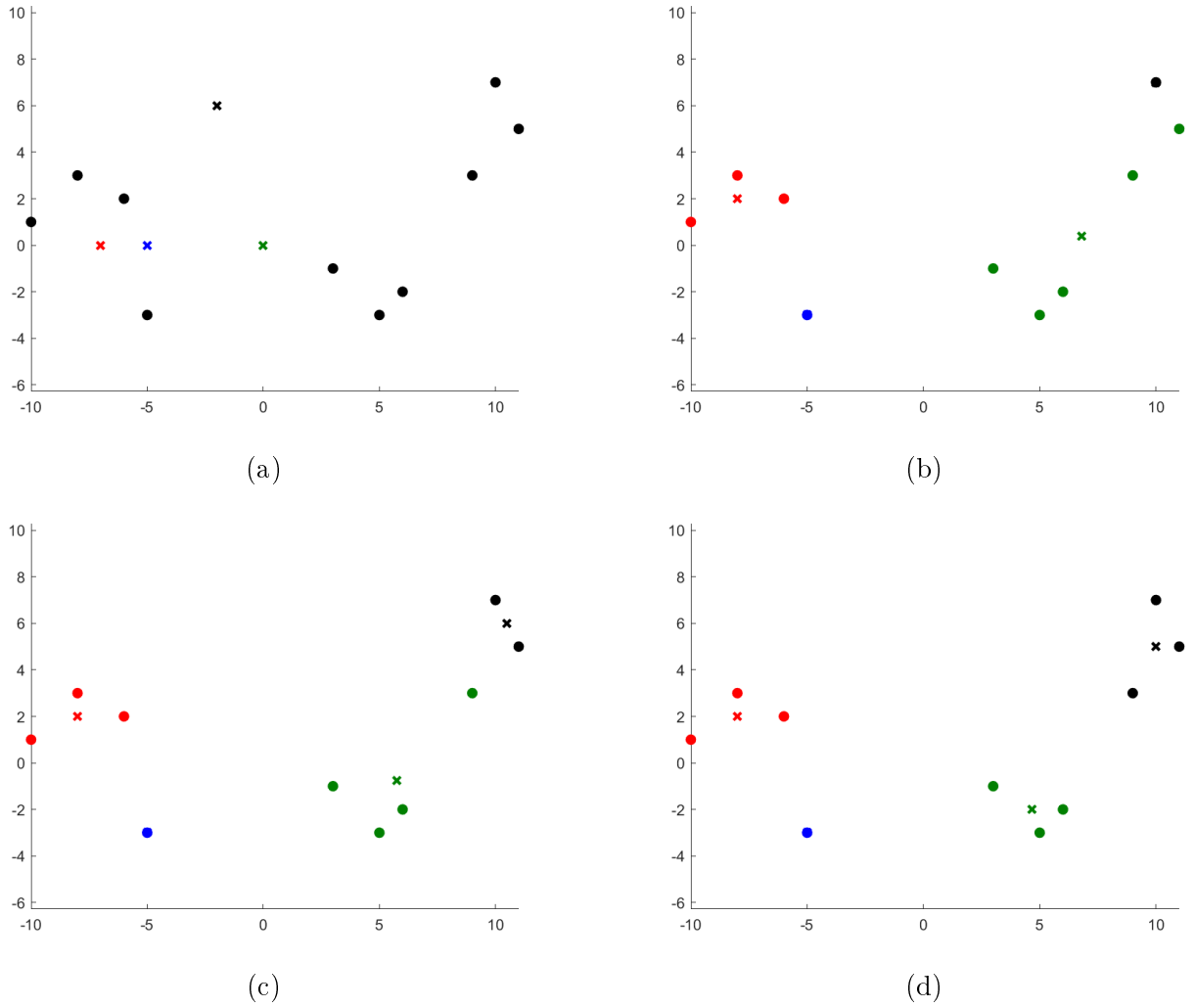


Figure 37: Example of k-means for $k = 4$. (a) Initial data, (b) Iteration 1, (c) Iteration 2, (d) Iteration 3.

where $(x_i - c + c - p)^T$ denotes the transposition of the column vector $x_i - c + c - p$.

Since

$$\sum_{i=1}^{m}(x_i - c) = \sum_{i=1}^{m} x_i - mc = \sum_{i=1}^{m} x_i - \sum_{i=1}^{m} x_i = 0$$

we therefore have

$$\sum_{i=1}^{m} ||x_i - p||^2 = \sum_{i=1}^{m} ||x_i - c||^2 + m||c - p||^2 \overset{(*)}{\geq} \sum_{i=1}^{m} ||x_i - c||^2$$

with equality in $(*)$ if, and only if, $c = p$. □

Theorem 5.2. *For any data set X, set of sites S, and any $k \in \mathbb{N}$ the k-means algorithm decreases (from one iteration to the next) the SSE.*

Proof. Let $X = \{x_1, \ldots, x_m\}$, $S = \{s_1, \ldots, s_k\}$, and let $C_1, \ldots, C_k$ denote the current clusters. Now, let $C'_1, \ldots, C'_k$ denote the clusters that we compute from these data in step (a), and let $c'_1, \ldots, c'_k$ denote

the corresponding cluster centroids. Then,

$$\frac{1}{m}\sum_{i=1}^{k}\sum_{x_j\in C_i}||x_j-s_i||^2 \geq \frac{1}{m}\sum_{i=1}^{k}\sum_{x_j\in C_i'}||x_j-s_i||^2 \quad \text{(since any } x_j \text{ is assigned to the closest } s_i\text{)}$$

$$\geq \frac{1}{m}\sum_{i=1}^{k}\sum_{x_j\in C_i'}||x_j-c_i'||^2 \quad \text{(by Lemma 5.1)}$$

□

Theorem 5.3. *For any data set X, set of sites S, and any $k \in \mathbb{N}$ the k-means algorithm converges in a finite number of steps.*

Proof. There are at most k^m ways to partition m points into k clusters (for every of the m points we have k possibilities to assign it to clusters). This is a finite (though large) number. For each such partition there are the centroids determined by step (b). Hence, we have at most k^m different values for the SSEs.

By Theorem 5.2 the SSE decreases in each iteration (if the SSE remains the same then the stopping criterion is fulfilled, hence we have convergence). The SSE, however, cannot continue to strictly decrease indefinitely since have already mentioned above that there are at most k^m values for it. Therefore, the stopping criterion is fulfilled after a finite number of steps. □

Although we have proved convergence, there are several unfortunate issues. First, the convergence is only to a local minimum (in practice, one therefore needs to re-run the algorithm several times and record the found minimum). Second, the speed of convergences can depend much on the dimension d, the number of points n, and the set of sites S. Is is beneficial to provide as input good approximations of the cluster centroids, but in general one encounters computationally difficult problems.

It is for instance, known[53] that k-means is $\mathbb{NP}$-hard when the dimension d is part of the input and $k = 2$. On the other hand, already for $d = 2$ k-means is $\mathbb{NP}$-hard if the number k is part of the input[54]. But there is a polynomial-time algorithm for planar k-means if k is fixed[55].

Example 5.9. *The k-means algorithm can be used for image compression.*

(a) (b) (c)

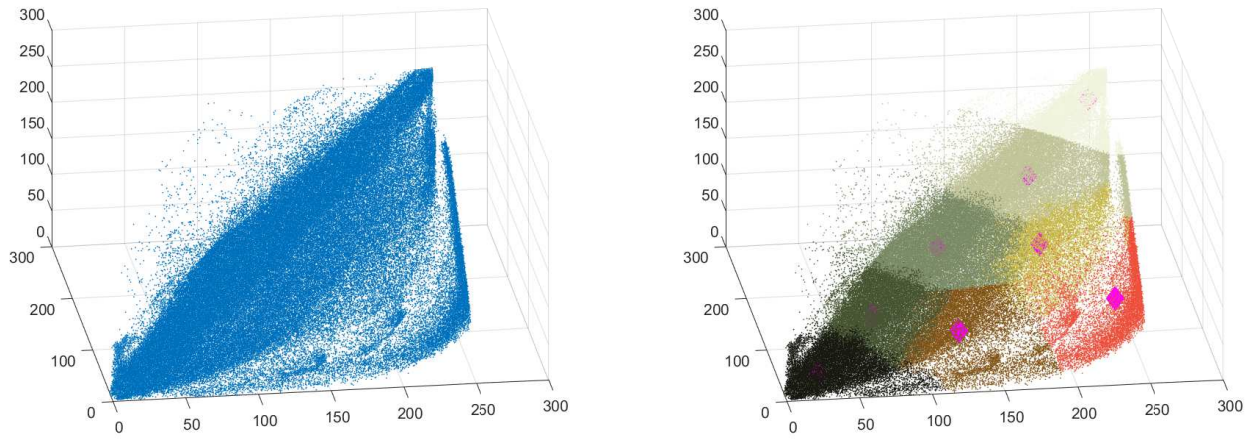
Figure 38: Compression of a 640×380 color image. (a) original image (`image_india.png`) containing $195,324 > 2^{17}$ colors, (b) compressed to $32 = 2^5$ colors, (c) compressed to $8 = 2^3$ colors.

Let us look at the compression rate. Often, color images are stored by providing for each pixel an RGB vector (this is a 3-dimensional vector containing the red/green/blue value in the respective components). Typically, each entry in the RGB vector is an integer between 0 and 225, hence each entry needs 1 Byte of storage. For color images not requiring the full color range, it is often beneficial to store for each pixel the color label together with a lookup table, which, for each color label, contains the corresponding RGB vector. The following table compares the sizes of the images shown in Fig. 38.
Just to get an idea about the computation times, we remark that with matlab's built-in `kmeans` *procedure, the computation time for producing (b) and (c) in Fig. 38 was* 10.0 *and* 4.2 *seconds, respectively.*

[53] https://www.cc.gatech.edu/~vempala/papers/dfkvv.pdf
[54] https://www.sciencedirect.com/science/article/pii/S0304397510003269
[55] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.361.4195&rep=rep1&type=pdf

Image	Size (Bytes)	Reduction Factor
(a)	$640 \cdot 380 \cdot 3 = 729,600$	1.0
(b)	$640 \cdot 380 \cdot 5/8 + 32 \cdot 3 = 152,096$	4.8
(c)	$640 \cdot 380 \cdot 3/8 + 8 \cdot 3 = 91,224$	8.0

Table 8: Compression rates for the images shown in Fig. 38.



(a) (b)

Figure 39: k-means clustering of the RGB vectors. (a) the 729,600 data points (RGB vectors) of `image_india.png`, (b) k-means clustering with k=8 giving the image in Fig. 38(c).

Let us state several remarks.

- The k-means algorithm is certainly one of the most popular clustering method. There are several variants of this method, e.g., *k-means++* (chooses the initial seeds in a special way), *fuzzy k-means* (a form of clustering in which each data point can belong to more than one cluster), and *k-medians clustering* (minimizing the SSE (5.3) where instead of the Euclidean norm the 1-norm is taken).

- Sometimes the number k of clusters is given by the application (e.g. we might know the number of crystals in a polycrystalline material). Then, it can be straightforward to apply k-means. In other cases, however, it might not be clear what the most appropriate k is. It needs to be learned from the data as well. A possible approach for doing this is, for instance, *G-means*[56], which is based on a statistical test for the hypothesis that a subset of data follows a Gaussian distribution. The algorithm runs k-means with increasing k in a hierarchical fashion until the test accepts the hypothesis that the data assigned to each k-means center are Gaussian.

- In step (a) of the algorithm we assign all points that are closest to s_i to the ith cluster. Doing this for all points in $\mathbb{R}^d$ one would obtain

$$P_i = \{x \in \mathbb{R}^d : ||x - s_i||^2 \leq ||x - s_j||^2\} \qquad \text{for all } j \neq i.$$

  This set P_i is called *Voronoi cell*, and the collection $P_1, \ldots, P_k$ is a so-called **Voronoi diagram**[57]. Voronoi diagrams appear in many different contexts as, for instance, in robotics, imaging, physics, biology, ecology, etc. The philosopher René Descartes[58] thought that the solar system can be viewed as a Voronoi diagram built of vortices centered at fixed stars (his Voronoi diagram's sites).

[56] See https://papers.nips.cc/paper/2526-learning-the-k-in-k-means.pdf
[57] Georgy Feodosevich Voronoy (28 April 1868 - 20 November 1908)
[58] (31 March 1596 - 11 February 1650)

Remarkably, the 'decision boundaries' in k-means are linear in the following sense.

Theorem 5.4. *The shared boundary of any two neighboring Voronoi cells P_1 and P_2 is linear.*

Proof. Note that for the shared boundary we have

$$\begin{aligned} & ||x - s_1||^2 = ||x - s_2||^2 \\ \Leftrightarrow \quad & (x - s_1)^T(x - s_1) = (x - s_2)^T(x - s_2) \\ \Leftrightarrow \quad & ||x||^2 - 2s_1^T x + ||s_1||^2 = ||x||^2 - 2s_2^T x + ||s_2||^2 \\ \Leftrightarrow \quad & 2(s_2 - s_1)^T x = ||s_2||^2 - ||s_1||^2. \end{aligned}$$

Hence, the boundary is a subset of

$$B = \{x \in \mathbb{R}^d : ||x - s_1||^2 = ||x - s_2||^2\} = \{x \in \mathbb{R}^d : (s_2 - s_1)^T x = (||s_2||^2 - ||s_1||^2)/2\},$$

the latter of which is clearly an affine hyperplane in $\mathbb{R}^d$ (s_1 and s_2 are fixed). □

If one chooses other norms, then one would obtain other types of *tessellations*[59], for instance, *power diagrams*, *Laguerre tessellations*, or *generalized balanced power diagrams*. An illustration of the tessellation for Fig. 37(d) is shown in Fig. 40.

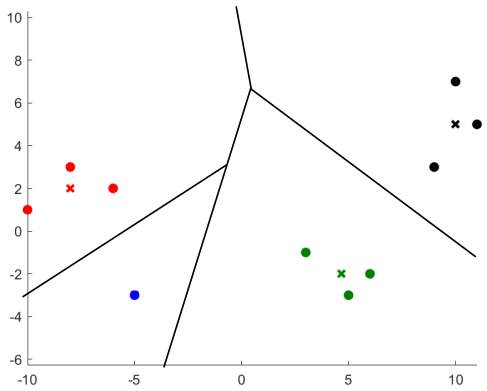


Figure 40: Tessellation for Fig. 37(d).

5.3.2 Learning on huge feature spaces

A large dimension d of the feature space can be problematic. On the one hand, the computational cost of the learning algorithms can be enormous. But even data visualization becomes difficult for $d > 2$ (and surely for $d > 3$). In addition, many unintuitive phenomena occur in higher dimensions. For instance, a random point in a unit square has about 0.4% chance to be located less than 0.001 from the boundary. But for a 10,000 dimensional hypercube there is 99.99% probability for a point to lie near the boundary. (To prove this just compare the volumes of the hypercube and the boundary strips.) The average distance between two random points in a unit square is around 0.52 where as for a 1,000,000 dimensional hypercube, it is about 408 (nicely explained proofs of these facts can be found at several websites[60]).

For large dimensions one therefore might want to resort to so-called **dimensionality reduction techniques**, which reduce the dimensionality of the data. Of course, such techniques are not always applicable as all the data can be important.

An important class of dimensionality reduction techniques projects the data into lower dimensional subspaces. We will discuss the technique called **principle component analysis (PCA)**[61]. It is one

[59] These are dissections of the space $\mathbb{R}^d$.

[60] See, e.g., https://math.stackexchange.com/questions/2985662/average-distance-between-points-on-a-hypercube

[61] Invented in 1901 by Karl Pearson, an English mathematician and biostatistician (27 March 1857 - 27 April 1936).

of the oldest and most widely used method for dimension reduction. The projection employed in this method is a linear projection (there exist also methods that utilize non-linear projections).

The PCA technique applies generally if one can assume that there is some redundancy in the data. Redundancy means that some of components/coordinates of the x_i are correlated with one another, possibly because they are measuring the same construct (for instance, height above sea level and temperature as the temperature decreases for larger heights, or acceleration and force on a particle which are related by Newton's second law of motion). Then the goal is to reduce the dimension d of each data point x_i by replacing $x_i \in \mathbb{R}^d$ by a smaller dimensional (artificial) data point $y_i \in \mathbb{R}^k$ ($k < d$) that will *account for most of the variance* in the observed data. We will aim for projecting the d-dimensional data points onto a subspace spanned by k directions called **principal components**.

First, we briefly recall several concepts.

- A matrix $M \in \mathbb{R}^{d\times d}$ is said to have an *eigenvalue* of λ if there is a d-dimensional vector $u \neq 0$ for which $Mu = \lambda u$. This u is then the *eigenvector* corresponding to λ.
- For the sake of exposition, let us assume in the following that the data is *standardized*, which means that the data has zero mean and standard deviation 1 in each dimension[62].
- Let $x_i = (\xi_1^{(i)}, \ldots, \xi_d^{(i)})^T$, $i = 1, \ldots, m$, denote the data points. The corresponding *sample covariance matrix* holds in its ith row and jth column the covariance between the two features i and j. Formally,

$$C = \begin{pmatrix} c_{1,1} & \cdots & c_{1,d} \\ \vdots & & \vdots \\ c_{1,1} & \cdots & c_{d,d} \end{pmatrix}, \qquad \text{with } c_{i,j} = \frac{1}{m-1}\sum_{\ell=1}^{m} \xi_i^{(\ell)}\xi_j^{(\ell)}.$$

  This, by the way, is a symmetric matrix. Oberserve that if we arrange the data points x_i into a matrix $X \in \mathbb{R}^{d\times m}$ whose ith column contains the data point x_i, then C can be expressed as

$$C = \frac{1}{m-1} X \cdot X^T = \frac{1}{m-1}\sum_{\ell=1}^{m} x_\ell x_\ell^T.$$

  Covariance is, in fact, a measure of how much two sample features vary together. (It is similar to variance, but where variance indicates how a single feature varies, covariance indicates how two features vary together.) A positive covariance between two features indicates that the features increase or decrease together, whereas a negative covariance indicates that the features vary in opposite directions.

Now, we are ready to derive the principal components. Note that there are several equivalent ways of deriving them. One way is by finding the projections that maximize the variance. The first principal component is the direction along which projections have largest variance. The second principal component is the direction which maximizes variance among all directions orthogonal to the first, and so on.

Another equivalent way is to look for projections with the smallest average mean-squared distance between the original points and their projections on to the principal components.

Variance maximization Let us first focus on maximizing the variance.

We assume that our standardized data is collected in a $d \times m$ matrix X, which contains the data point x_i in the ith column. We wish to find a direction u that captures as much as possible of the variance of our data points. Formally, this amounts to solving the optimization problem

$$\text{Find } u \in \mathbb{R}^d \text{ with } ||u|| = 1 \text{ so as to maximize } \operatorname{var}(u^T X). \tag{5.4}$$

[62] This is achieved by setting the transformed data to $\xi_i' = (\xi_i - \mu_i)/\sqrt{\sigma_i}$, where μ_i and σ_i is the mean and variance, respectively, of the samples ith coordinates. Standardization in matlab works by using the command `normalize(X')` where $X \in \mathbb{R}^{d\times m}$ is the matrix that has the data point x_i in its ith column.

Note that $u^T X$ denotes the projection of X (i.e, of all column vectors $x_1, \ldots, x_m$) to the subspace spanned by the single vector u. We require $||u|| = 1$ since if we would allow scaling we would obtain arbitrary large values of $\text{var}(u^T X)$, and it would therefore make no sense to speak of a maximal variance.

Theorem 5.5. *The solution to the optimization problem (5.4) is to set u to equal the first principal component (that is the eigenvector corresponding to the largest eigenvalue) of C.*

Proof. As we assume that the data $x_1, \ldots, x_m$ is standardized they have sample mean $\mu = 0$ and sample variance

$$\begin{aligned}\text{var}(u^T X) \overset{\text{def. of sample variance}}{=} &\frac{1}{m-1}\sum_{i=1}^{m}(u^T x_i - \mu)^2 = \frac{1}{m-1}\sum_{i=1}^{m}(u^T x_i)^2 \\ =&\frac{1}{m-1}\sum_{i=1}^{m}(u^T x_i)(x_i^T u) = \frac{1}{m-1}\sum_{i=1}^{m} u^T (x_i x_i^T) u \\ =&u^T C u.\end{aligned}$$

Now, our optimization problem reduces to finding $u \in \mathbb{R}^d$ with $||u|| = 1$ that maximizes $u^T C u$. To solve this, we use the following notation. With v_i we denote the eigenvector of C corresponding to the ith largest eigenvalue λ_i of C. Then,

Claim:

$$\max_{||u||=1} u^T C u = \max_{u \neq 0} \frac{u^T C u}{u^T u} = \lambda_1$$

and this maximum is realized for $u = v_1$.

Proof of claim: Notice first that, by the eigenvalue decomposition (for real symmetric matrices), we can write $C = QDQ^T$, where $Q \in \mathbb{R}^{m \times m}$ holds in the ith column the ith eigenvector v_i, and $D \in \mathbb{R}^{d \times d}$ is a diagonal matrix with $d_{i,i} = \lambda_i$ on the diagonal. (Note, Q is an orthogonal matrix, i.e., QQ^T is the identity matrix I.) Then,

$$\begin{aligned}\max_{u \neq 0} \frac{u^T C u}{u^T u} &= \max_{u \neq 0} \frac{u^T Q D Q^T u}{u^T Q Q^T u} \qquad (\text{since } QQ^T = I) \\ &= \max_{y \neq 0} \frac{y^T D y}{y^T y} \qquad (\text{by setting } y = Q^T u) \\ &= \max_{y \neq 0} \frac{\lambda_1 \eta_1^2 + \cdots + \lambda_d \eta_d^2}{\eta_1^2 + \cdots + \eta_d^2} \\ &\leq \max_{y \neq 0} \frac{\lambda_1 \eta_1^2 + \cdots + \lambda_1 \eta_d^2}{\eta_1^2 + \cdots + \eta_d^2} \\ &= \lambda_1,\end{aligned}$$

where equality is attained in the inequality when $y = (\eta_1, \ldots, \eta_d)^T = (1, 0, \ldots, 0)^T$, that is $u = Qy = v_1$, proving the claim.

□

Smallest average mean-squared distance Above we claimed that PCA can also equivalently be defined by looking for projections with the smallest average mean-squared distance between the original points and their projections on to the principal components.

Formally, this amounts to solving the optimization problem

$$\text{Find orthonormal } u_1, \ldots, u_d \in \mathbb{R}^d \text{ minimizing } \sum_{i=1}^{m} || \sum_{j=1}^{k} (u_j^T x_i) u_j - x_i ||^2 \text{ for every } k \leq d. \tag{5.5}$$

Theorem 5.6. *The solution to the optimization problem (5.5) is to set $u_1, \ldots, u_d$ to be the principal components $v_1, \ldots, v_d$ of C.*

Proof. Any x_i can be written as

$$x_i = \sum_{j=1}^{d} \alpha_{i,j} u_j,$$

where

$$\alpha_{i,j} = u_j^T x_i.$$

Now, suppose we approximate x_i just by the first k terms of that linear combination, i.e.,

$$x_i' = \sum_{j=1}^{k} \alpha_{i,j} u_j.$$

The approximation error should be small for all $x_1, \ldots, x_m$, hence we want that

$$\frac{1}{m-1} \sum_{i=1}^{m} ||x_i' - x_i||^2 = \frac{1}{m-1} \sum_{i=1}^{m} || \sum_{j=k+1}^{d} \alpha_{i,j} u_j||^2 = \frac{1}{m-1} \sum_{i=1}^{m} \sum_{j=k+1}^{d} \alpha_{i,j}^2$$

$$= \frac{1}{m-1} \sum_{i=1}^{m} \sum_{j=k+1}^{d} u_j^T x_i x_i^T u_j = \sum_{j=k+1}^{d} u_j C u_j$$

is small. If $k = d-1$ it is easy to see (you can do this as an exercise, analogously to the proof of the claim in the proof of Theorem 5.6) that we need $u_d = v_d$ since $v_d C v_d = \lambda_d$ is the smallest among all unit vectors. And consequently applying this argument we see that we need to remove subsequently the eigenvectors corresponding to the least $d-k$ eigenvalues. □

Summary and Examples In summary we obtain the following algorithm.

Principal component analysis (PCA):
Given $k \in \mathbb{N}$ and a set of standardized *data points* $X = \{x_1, \ldots, x_m\} \subseteq \mathbb{R}^d$.

(a) Construct the sample covariance matrix C.

(b) Decompose the covariance matrix into its eigenvectors and eigenvalues.

(c) Sort the eigenvalues in decreasing order.

(d) Select the first k eigenvectors corresponding to the k largest eigenvalues.

(e) Construct projection matrix W from these k eigenvectors.

(f) Transform the d-dimensional input dataset X into Y using the projection matrix W to obtain the new k-dimensional feature subspace.

A matlab implementation might be as follows.

```
%Matlab implementation of the PCA
%Input:  X : dxm matrix holding the d-dimensional standardized data points in the columns
%        k : the dimension into which we want to project
%Output: Y : kxm matrix holding the projected data points in the columns

C=X*X'              %C is the coveriance matrix (dimensions dxd)

[W,D] = eigs(C,k)   %Compute the k top eigenvalues and eigenvectors
                    %W contains top k eigenvectors as columns, it is a dxk matrix
                    %D contains top k eigenvalues on its diagonal, it is a kxk matrix

Y = W'*X
```

Note that sometimes, as in Fig. 41(b) one might view the projected points Y in the original space $\mathbb{R}^d$ (and not in $\mathbb{R}^k$). What one simply needs to do is to compute $W \cdot Y$, the columns of this matrix are the desired points in $\mathbb{R}^d$.

An example for a PCA for 50 data points in $\mathbb{R}^2$ projected onto the first principal component is shown in Fig. 41.

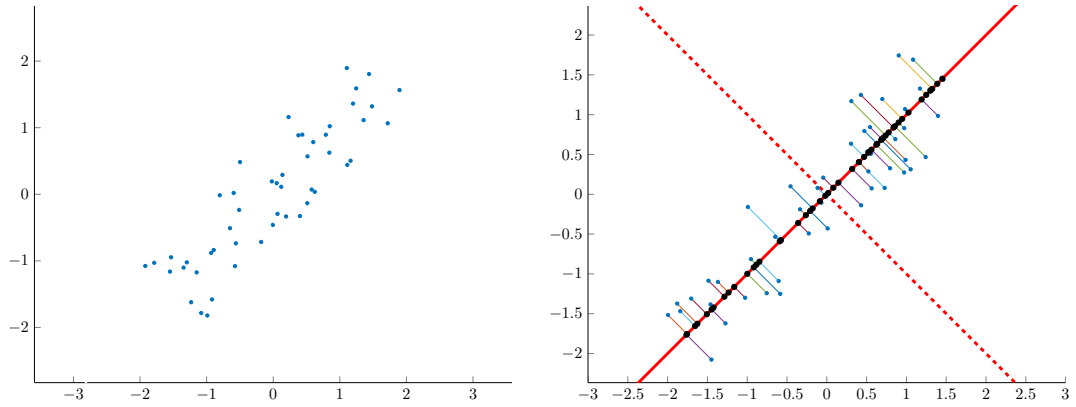


Figure 41: PCA in $\mathbb{R}^d$ for $k = 1$. (a) The 50 data points, (b) first principal component in red, second principal component in dashed red, projected points in the original space are shown in black on the red line.

5.4 Reinforcement learning and Markov decision processes

We have already mentioned that reinforcement learning as a technique falls between the two categories of supervised and unsupervised learning. Reinforcement learning allows an agent to learn how to behave in an environment, where the only feedback is the reward signal. The agent's goal is to perform actions that maximize future reward.

To make this precise we introduce the notion of Markov decision processes, which can be considered to provide a formal framework for reinforcement learning.

Definition 5.3. *A **Markov decision process (MDP)** is a tuple (Ω, A, T, R) in which*

- *Ω is the set of **states**,*
- *A is a finite set of **actions**,*
- *T is a **transition function** $T : \Omega \times A \times \Omega \to [0, 1]$, which, for $T(\omega, a, \omega')$, gives the probability that action a performed in state ω will lead to state ω',*
- *and R is a **reward function** defined as $R : \Omega \to \mathbb{R}$.*

A **policy** ρ is a function $\rho : \Omega \to A$, which tells the agent which action to perform in any given state. Application of a policy to an MDP proceeds as follows. First, a start state ω_0 is generated. Then, the policy ρ proposes the action $a_0 = \rho(\omega_0)$ and this action is performed. Based on the transition function T and reward function R, a transition is made to state ω_1 with probability $T(\omega_0, a_0, \omega_1)$, and a reward $R(\omega_1)$ is received. This process continues producing a sequence $\omega_0 a_0 \omega_1 a_1 \omega_2 a_2 \ldots$. (If the process is about to end in a final state, as for instance, in finite games, then one can consider to restart the process in a new initial state; this would always result in an infinite sequence.) Hence, given a fixed policy, MDPs are in fact Markov chains.

The main **goal** of learning in an MDP is to 'learn' (i.e., *find*) a policy that gathers rewards. If the agent was only concerned about the immediate reward, a simple optimality criterion would be to optimize $\mathbb{E}[r_t]$, where $r_t = R(\omega_t)$. There are, however, several ways of taking the future into account.

For instance, one could aim at optimizing

$$\mathbb{E}[\sum_{t=0}^{h} r_t] \qquad \text{or} \qquad \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t],$$

where h is a fixed number of times steps and $\gamma < 1$ a given parameter. The former function with the finite sum is used in so-called *finite horizon* approaches, the latter is used in so-called *discounted, infinite horizon approaches.*

A popular and mathematical rigorous method for finding an optimal policy for a an MDP (where really all the data, i.e., (Ω, A, T, R) is given) is so-called *dynamic programming.* This is a method, which is also used in other contexts such as for finding shortest paths in a graph. We discuss this in the next subsection. The interested reader may find [15] to be a good starting point to find out more about MPDs.

Example 5.10. *Consider the problem of finding an 'optimal strategy' in a game of tic-tac-toe*[63]*. We can model this as an MDP.*

A position of the board (i.e., the configuration of Xs and Os) can be considered as state *(states can be enumerated, hence Ω can be considered to be a set of natural numbers $\Omega = \{1, \dots, 3^9\}$). The* actions *correspond to the moves made by the agent (again these could be enumerated).*

What would the transition function tell us? $T(\omega, a, \omega')$ could, for instance, tell us with what probability the opponent would bring up the position ω' if we were in position ω performing move a. Such a model would be adequate if the agent should learn to play against a specific type of player (one, which does not necessarily play perfect). In practice such a transition matrix needs to be learned as well, and we will comment on this at the end of this example.

For the reward function, it would be most natural to assign to the end positions of the game where the agent wins a, say, positive value, to the losing positions a negative value on for the draw positions a zero value. The reward for the remaining states might be considered to be indifferent, one might assign also a zero value to them.

The goal of the agent should be to reach positive valued states, which means winning the game.

As mentioned, if all the data would be known one could apply dynamic programming and obtain an optimal policy/strategy. But suppose, T needs to be learned as well. Then it would be natural to play many times against the opponent and record the transitions as approximations of the probabilities. (Which, of course, assumes that the opponent does not change its style of playing after a period of time.) This can then also be combined with strategies that learn how 'valuable' a given position in the game actually is[64]*.*

5.4.1 Dynamic programming

We mentioned above that *dynamic programming* can be used to compute an optimal police for an MDP where all data is given. For the sake of illustration, suppose we are given the MPD shown in Fig. 42. We have 5 states, actions a, b, c, d, e, f, and 0. Instead of rewards, we consider costs (rewards and costs can be converted into each other by multiplication of -1). They are given as labels on the solid lines. Labels on the dashed lines give $\Pr(s|a)$, the probability of reaching state s given that action a is performed (note that in the notation of MDPs we have $T(w_1, a, s) = T(w_1, a, s) = \Pr(s|a)$ for all $w_1, w_2 \in \Omega$, and $R(s) = -\text{cost}(s)$ for all $s \in \Omega$). We are interested in computing an optimal policy for this MDP; optimal in the sense that wherever we start we want that the expected costs of reaching the state 0 are minimal. How can we do this?

[63] In this game, players alternate placing pieces (typically Xs for the first player and Os for the second) on a 3×3 board. The first player to get three pieces in a row (vertically, horizontally, or diagonally) is the winner.

[64] A good webpage for further studies on this theme, including sample code, can be found at https://www.codeproject.com/Articles/1400011/Reinforcement-Learning-A-Tic-Tac-Toe-Example.

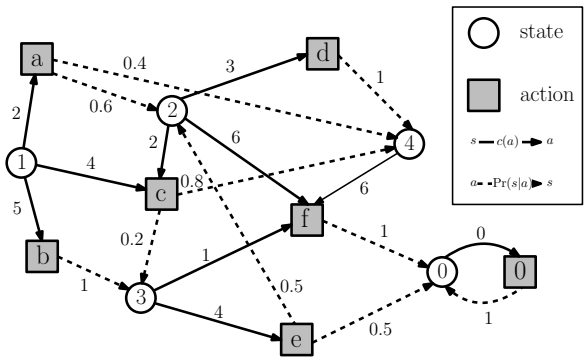


Figure 42: MDP with 5 states $0, \ldots, 4$ and actions a, b, c, d, e, f, and 0. Labels on solid lines give the costs, labels on dashed lines give $\Pr(s|a)$.

First of all, we can see that some of the arcs are irrelevant since no shortest path to 0 will lead over them. For instance, performing action b in state 1 leading to state 3 will always incur larger expected costs than performing action c in state 1 again leading to state 3. Hence, we can delete the arcs $(1, b)$ and $(b, 3)$. With this type of reasoning we obtain the graph shown in Fig. 43(a). Finally, we can replace each pair of solid and dashed arc by a single arc whose weight equals the product of the two deleted arcs (the expected cost incurred by traversing these two arcs). We obtain the graph shown in Fig. 43(b).

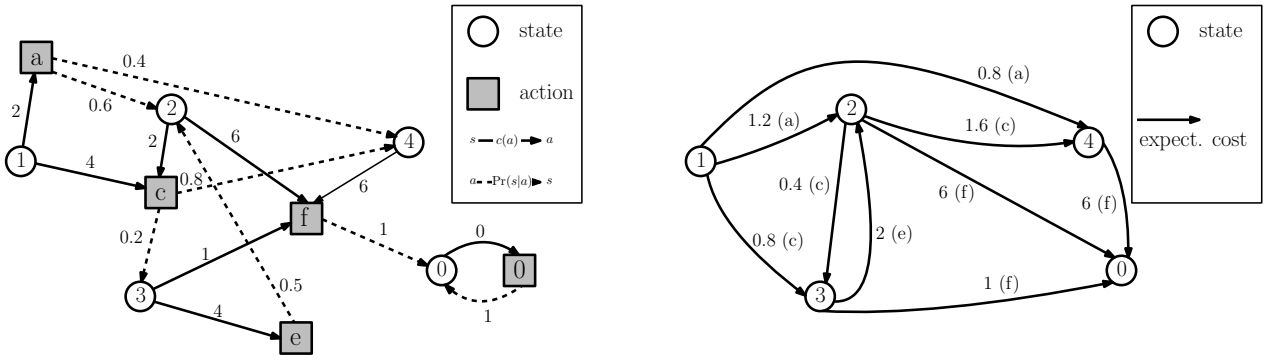


Figure 43: (a) Deleting arcs that are irrelevant for shortest paths to state 0, (b) replacing pairs of solid and dashed arcs by solid arcs indicating corresponding expected costs.

Now the task of finding an optimal policy (minimizing the expected costs to reach state 0) is reduced to the task of determining for each state the shortest path (with respect to the arc weights) leading to state 0.

Of course, we can try to solve this problem by complete enumeration (listing all possible paths and determining the shortest among them). This is, of course, not efficient, and there are indeed better methods for solving this task.

Solving this problem by **dynamic programming** gives a more efficient algorithm for solving this task. The basic idea of dynamic programming is to find a way to break the problem down into a manageable number of subproblems in such a way that we can use optimal solutions to the smaller subproblems to give us optimal solutions to the larger ones (unlike *divide-and-conquer* these subproblems are typically overlapping). The resulting algorithm for solving the so-called *single-sink shortest path problem* is the so-called **Bellman-Ford**[65] algorithm (it can be shown to work correctly if there are no negative-weight cycles).

The algorithm proceeds as follows.

[65] Named after R. Bellman (29 August, 1920 - 19 March 1984) and L. Ford (23 September 1927 - 26 February 2017).

Bellman-Ford:

1. For each state v find the length of the shortest path from v to 0 that uses at most ones arc, and write down ∞ if no such path exists;
2. Suppose, for all v we now know the length of the shortest path to 0 that uses $i-1$ or fewer arcs. This information can be used to solve for the shortest path that uses i or fewer arcs, because the shortest path from v to 0 that uses i or fewer arcs will first visit one neighbors w of v and then take the shortest path from w to 0 that uses $i-1$ or fewer arcs. As we have computed this length already, all we need to do is to take the minimum over all neighbors w of v.
3. With n denoting the total number of states, we can terminate after $i = n-1$ arcs have been considered as more arcs will always form a cycle (and this can clearly be no shortest path anymore).

For the example from Fig. 42 we obtain with Bellman-Ford the iterations shown in Fig. 44. The shortest path from state 1 to 0 is updated at iteration 2, its length is 1.8. The optimal policy from state 1 to reach state 0 has therefore expected cost of 1.8; it will perform action c to reach state 3 followed by performing action f to finally reach state 0.

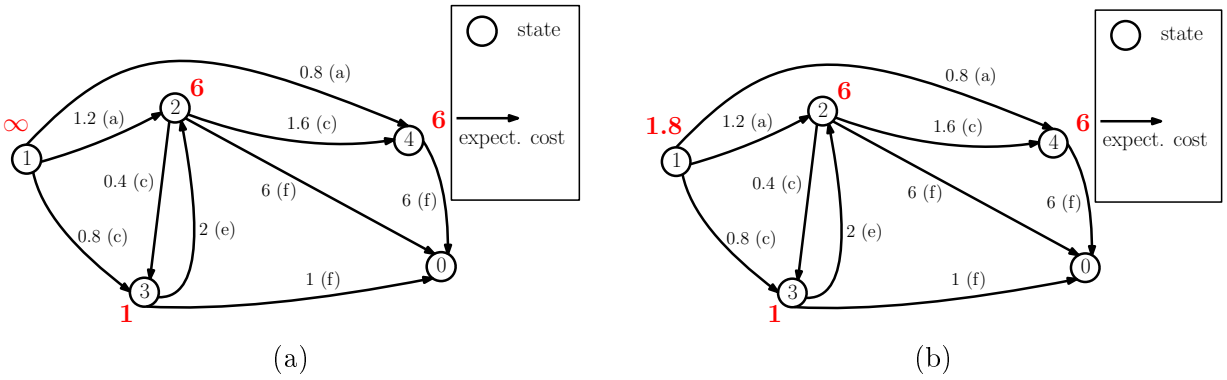


Figure 44: Bellman-Ford algorithm. (a) First iteration (length of shortest path to state 0 indicated in red), (b) Second (and final) iteration.

5.5 Neural networks: Hopfield networks, Boltzmann machines

In this section we will briefly discuss neural networks, and in particular Hopfield networks and Boltzmann machines. We will keep the exposition as short as possible. The interested reader can find more information in [16] and [17].

5.5.1 Neural networks

(Artificial) neural networks are usually used for supervised learning (there do exist examples for unsupervised and reinforcement learning, too). They try to model the apparently highly nonlinear infrastructure of brain networks. The historically first artificial neural network (called *perceptron*) was invented in 1958 by psychologist Frank Rosenblatt[66].

Neural networks are composed of layers of computational units called **nodes** (or **neurons**) with connections in different layers. Neurons are (typically nonlinear) parameterized functions of their input variables. In a very common model, called **McCulloch–Pitts neuron model**, the output s_i of a node i is given as

$$s_i = \psi(\sum_j w_{i,j} s_j + \theta_i),$$

[66] American psychologist (July 11, 1928 - July 11, 1971).

where $w_{i,j}$ is the **weight** of the connection between node i and j, θ_i is a given constant (called **threshold**) and ψ is the so-called **activation function**. Typical examples of activation functions include the identity function, the tanh function, and the step function

$$\psi(x) = \begin{cases} 1 & \text{if } x \geq 0, \\ -1 & \text{otherwise.} \end{cases}$$

Some of the nodes may be so-called **input nodes**, which means that their input is externally provided. Some of the nodes may be **output nodes**, which means that the user can read them out. Other types of nodes are **hidden nodes**, they are used internally solely for computations.

To specify a neural network ones needs to specify the **network architecture** (i.e., the nodes and connections), the **updating rules** (i.e., the functions evaluated by each node), and the **learning rules** (i.e., procedures to compute the weights from training data). We discuss this further below when we discuss Hopfield networks and Boltzmann machines.

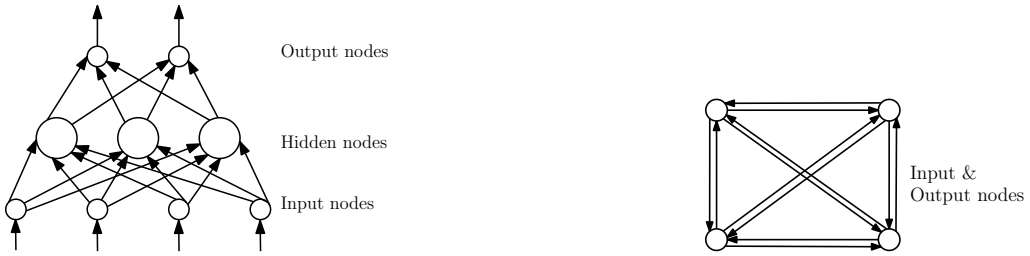


Figure 45: Examples of neural network architectures. (a) Network without any circle (*feedforward network*), (b) Network with cycles (*recurrent network*).

5.5.2 Hopfield networks

Hopfield networks played an important historical part in the development of neural networks. They were invented in 1982 [18] by the physicist John Hopfield[67], who demonstrated with these networks that a so-called *associative memory* can be formed via activity-dependent changes in the strength of connections between coactive neurons during training. An associative memory is able to retrieve a set of previously memorized patterns from their noisy versions. This can be considered as a form of noise reduction.

The architecture of the Hopfield network is a network of N nodes, which are fully interconnected with symmetric weights ($w_{i,j} = w_{j,i}$), without self-connections ($w_{i,i} = 0$ for all $i \in \{1, \ldots, N\}$), and all of whose nodes are both input and output nodes. For an example of an $N = 2 \cdot 2 = 4$ Hopfield network, see Fig. 45(b).

Since Hopfield networks contain loops (it is a so-called **recurrent network**) one should not view the inputs to the nodes as being just externally provided, instead one should view them as being outputs from other nodes that change with the time. Hopfield networks are dynamical systems whose state changes with the time. The **state of the neural network** is the set of the outputs of all nodes at a particular moment in time. The output s_i of node i can therefore also be referred to as **state of node i** (at a particular moment in time).

One of the most important contributions in the 1982 paper was the introduction of the idea of an **energy function** into neural network theory. For the Hopfield network the energy function H is

$$H = H(s_1, \ldots, s_N) = -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} w_{i,j} s_i s_j + \sum_{i=1}^{N} s_i \theta_i, \tag{5.6}$$

which gives the energy of the network as a function of the current states $s_1, \ldots, s_N$ (assuming that the weights and the thresholds $\theta_1, \ldots, \theta_N$ are given). We have seen such type of energy function before when we discussed in Example 4.9 the Ising model (see the exponent of e in the definition of π).

[67] (born July 15, 1933)

Updating a node in the Hopfield network[68] is performed using the rule

$$s_i = \begin{cases} +1 & \text{if } \sum_j w_{i,j} s_j \geq \theta_i, \\ -1 & \text{otherwise.} \end{cases}$$

Such updates might be performed either asynchronously (nodes are consecutively updated in a predefined order) or synchronously (all nodes are updated at the same time).

Remarkably, H decreases (i.e. decreases strictly or stays constant) as the system evolves according to its updating rule.

Theorem 5.7. *The energy function H of a Hopfield network decreases after any single updating step.*

Proof. Suppose node i changes its state from $s_i = \alpha$ to $s_i = \beta$ (we assume $\alpha, \beta \in \{\pm 1\}$; the case $\alpha, \beta \in \{0, 1\}$ follows analogously). We have

$$\begin{aligned} \Delta H &= H(s_1, \ldots, s_{i-1}, s_i = \beta, s_{i+1}, \ldots, s_N) - H(s_1, \ldots, s_{i-1}, s_i = \alpha_{i+1}, \ldots, s_N) \\ &= -\sum_j w_{i,j}\beta s_j + \beta\theta_i - \left(-\sum_j w_{i,j}\alpha s_j + \alpha\theta_i \right) \\ &= (\alpha - \beta)\left(\sum_j w_{i,j} s_j - \theta_i \right) \end{aligned}$$

Now if $\alpha = -1$ and $\beta = +1$ we have, since we were updating, $\sum_j w_{i,j} s_j - \theta_i \geq 0$, and thus $\Delta H = (\alpha - \beta)\left(\sum_j w_{i,j} s_j - \theta_i\right) \leq 0$. If $\alpha = +1$ and $\beta = -1$ we similarly have $\sum_j w_{i,j} s_j - \theta_i < 0$, and therefore $\Delta H = (\alpha - \beta)\left(\sum_j w_{i,j} s_j - \theta_i\right) < 0$. In both cases the energy function decreases (not necessarily strictly). □

The memorized patterns are the local minima of the energy function. Thus it is in principle possible to use Hopfield networks to solve optimization problems.

So, how does a Hopfield learn? The learning rule is rather simple. Suppose we have m patterns $\xi^{(\ell)} = (\xi_1^{(\ell)}, \ldots, \xi_N^{(\ell)})^T$ with $\xi_i^{(\ell)}$ being equal to the state s_i in the ℓth pattern, $\ell = 1, \ldots, m$. Then, the **learning rule** for weight $w_{i,j}$ with $i \neq j$ is

$$w_{i,j} = \frac{1}{m} \sum_{\ell=1}^{m} \xi_i^{(\ell)} \xi_j^{(\ell)}. \tag{5.7}$$

This is — up to a factor — the covariance between the states of node i and j (see definition of $c_{i,j}$ in Section 5.3.2). We can see that we obtain a large weight if most of the times the states for node i and j in the training set coincide. This is, in fact, a manifestation of a very intuitive learning rule from neuroscience, called *Hebbian learning rule*[69], which can be paraphrased as stating *'Neurons that fire together, wire together. Neurons that fire out of sync, fail to link.'*

Hopfield, in his original paper, demonstrated via simulations that about $0.15N$ relevant patterns can be stored in a Hopfield network consisting of N neurons. This is, by today's standards, not very impressive but it caught quite some attention in the 80's.

Example 5.11. *Let us consider the Hopfield network for $3 \cdot 3 = 9$ nodes, each can be in a $+1$ or a -1 state. The -1 state will be depicted in black, the $+1$ state in white. The nodes are numbered starting from the top left going to the right, row by row (see Fig. 46(a)). Further we assume that we set the thresholds θ_i to zero.*

[68] What we describe here is strictly speaking a discrete Hopfield network. There exist also continuous versions. They use a different activation function.

[69] Named after the Canadian psychologist Donald Hebb (July 22, 1904 - August 20, 1985) who introduced this rule in his MA thesis.

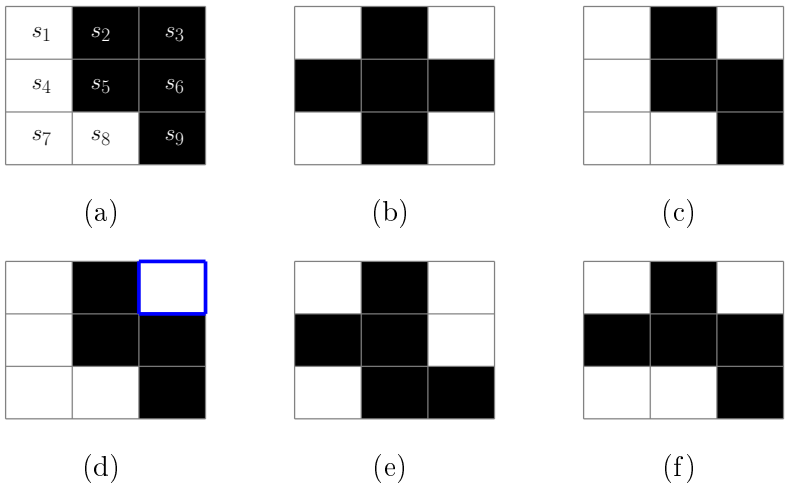


Figure 46: Hopfield network with $3 \cdot 3 = 9$ nodes. (a,b) learned patters, (c) first input pattern, (d) changed state, (e) second input pattern, (f) third input pattern.

Now, assume that we learn the two patters shown in Fig. 46(a) and (b). The corresponding weights $w_{i,j}$ (calculated according to (5.7)) are given by the following matrix (the entry in the ith row and jth column gives the weight $w_{i,j}$.

$$W = \begin{pmatrix} 0 & -1 & 0 & 0 & -1 & -1 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & -1 \\ -1 & 1 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ -1 & 1 & 0 & 0 & 1 & 0 & -1 & 0 & 0 \\ 1 & -1 & 0 & 0 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & -1 & 0 \end{pmatrix}$$

Now, suppose the input is the patterns as shown in Fig. 46(c). When we update all states, we find that only one single node will change its state, namely blue boxed node in Fig. 46(d). The corresponding input weights sum up to

$$\sum_{j=1}^{N} w_{i,j} s_j = -3.$$

Hence, the node will change into the state -1 (black). Then, no further updates will occur (the final energy, which will not decrease anymore, is here $H = -16$). The Hopfield does indeed recover the pattern Fig. 46(a), which makes sense from the point of noise removal.

When we take as input the pattern shown in Fig. 46(e), then we will indeed recover the patter in Fig. 46(b).

Starting, however, with the input pattern shown in Fig. 46(f) we will not recover any of the two learned patterns. The network gets stuck in a local minimum, which actually has energy $H = -8$.

As in the previous example, Hopfield networks can get stuck in local minima. This happens because there are usally several of them and the energy in each updating step decreases (or stay the same). This is where Boltzmann machines come into play.

5.5.3 Boltzmann machines

Boltzmann machines, named after Ludwig Boltzmann[70], are neural networks with the same architecture as Hopfields (additionally, it can contain hidden nodes but this is not important for the present discussion). The difference is that the updating rule is stochastic, which allows that network's energy does not get trapped in local minima. Boltzmann machines were first presented in 1985 [19]; one of the co-inventors, Terry Sejnowski[71], is a former PhD student of John Hopfield.

The Boltzmann machine's energy is defined in the same way as in (5.6) for Hopfields. **Updating** of the binary state of node i, however, proceeds as follows. First one computes the energy variation generated by a changes of its state from -1 to $+1$. According to (5.6) this is

$$\begin{aligned}\Delta H_i &= H(s_1,\dots,s_{i-1},s_i=+1,s_{i+1},\dots,s_N) - H(s_1,\dots,s_{i-1},s_i=-1,s_{i+1},\dots,s_N)\\ &= 2\theta_i - \sum_{j=1}^{N} w_{i,j}s_j.\end{aligned}$$

Then, node i turns (or remains) in state $+1$ given by the Metropolis-Hastings acceptance probability

$$\min\left\{1, \exp\left(-\frac{\Delta H_i}{T}\right)\right\},$$

where the (given) scalar T is referred to as the temperature of the system (see also *simulated annealing*).

The updating procedure for Boltzmann machines can therefore be understood as generating a Markov chain (it is, in fact, a form of Gibbs sampling). It is easily seen that this Markov chain is aperiodic and irreducible, and therefore, by Theorem 4.3, the stationary distribution is

$$\pi(s_1,\dots,s_N) = \frac{1}{Z}e^{-\frac{1}{T}H(s_1,\dots,s_N)}.$$

Distribution functions of this form are often referred to as **Boltzmann distributions**. (This is, in fact, why these machines are called 'Boltzmann machines.')

Boltzmann machines (in fact, restricted versions thereof) are nowadays considered to be powerful tools for recommender systems. For instance, all three approaches winning the Netflix Prize[72] (which sought to substantially improve the accuracy of predictions about how much someone is going to enjoy a movie based on their movie preferences) involved (restricted) Boltzmann machines[73].

With the interpretation of Boltzmann machines as Markov chains we have come, in some sense, full circle.

5.5.4 Solving optimization problems with neural networks

In previous chapters we have encountered several optimization problems (see, e.g., k-means, SVMs, simulated annealing). It might come as a surprise that neural networks, which we have previously used for supervised learning tasks such as classification, can also be used to solve optimization problems. In the following we illustrate this by showing how instances of the so-called *traveling salesman problem* can be solved by Hopfield networks (or Boltzmann machines).

The **traveling salesman problem (TSP)** is a very important problem in discrete mathematics and theoretical computer science[74]. The problem is as follows:

> *Given a list of n cities and the distances $d_{i,j}$ between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?*

[70] (February 20, 1844 - September 5, 1906), Austrian physicist and philosopher.

[71] (born 13 August 1947)

[72] https://www.netflixprize.com/

[73] Interestingly, it seems that Netflix never really implemented and employed any of these algorithms. See https://www.techdirt.com/articles/20120409/03412518422/why-netflix-never-implemented-algorithm-that-won-netflix-1-million-challenge.shtml.

[74] If you would like to play around with this problem, see the app https://www-m9.ma.tum.de/games/tsp-game/index_en.html.

The TSP is an $\mathbb{NP}$-hard problem, which means that it is provably in the class of the computationally hardest problems to solve (in fact, if you find an efficient algorithm for this, then you would win a US \$1 million prize of the Clay Mathematics Institute for solving the $\mathbb{P}$ vs. $\mathbb{NP}$ conjecture[75]). So, it is a notorious computationally hard problem to solve. Nevertheless, mathematicians have developed many novel mathematical techniques over the years[76], and this progress made it possible that a TSP instance involving 85,900 cities could be solved (which at the time of writing is the current word record[77]). Neural networks played no important role in this development. Discussing this neural network approach[78], however, serves some educational purposes as it gives an example of what general steps need to be taken in order to use a neural network to solve an optimization problem.

First we need some suitable encoding of the TSP instance. We do this by formulating the TSP as an unconstrained quadratic optimization problem involving only binary variables.

Let us start with the encoding. Any TSP route (also called tour) can be described as finite sequence $v_{i_1}, v_{i_2}, \dots v_{i_n}$ of cities (v_{i_1} being the first city followed by v_{i_2} on that tour). A bit more redundant, but better suited for encoding in a neural network, is the encoding of the tour by a permutation matrix $T \in \{0,1\}^{n\times n}$. Here, the entry $t_{i,i}$ in this matrix is 1 if, and only if, city v_i is the jth visited city on that tour. See Fig. 47 for an illustration.

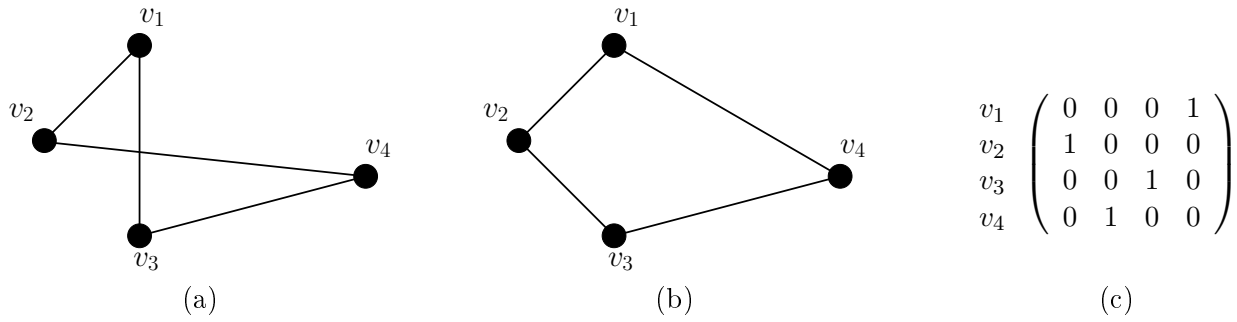


Figure 47: (a) A tour v_2, v_4, v_3, v_1, (b) a shortest tour (Euclidean distances), (c) Permutation matrix T representing the tour from (a).

What kind of constraints do we need to impose such that a given 0/1-matrix represents a tour? One possible way needs three types of constraints:

$$C_1(T) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\substack{k=1\\k\neq j}}^{N} t_{i,j}t_{i,k},$$

$$C_2(T) = \frac{1}{2}\sum_{j=1}^{N}\sum_{i=1}^{N}\sum_{\substack{k=1\\k\neq i}}^{N} t_{i,j}t_{k,j},$$

$$C_3(T) = \frac{1}{2}\left(\sum_{i=1}^{N}\sum_{j=1}^{N} t_{i,j} - n\right)^2.$$

If $C_1(T) = 0$ then in reach row there is at most one 1; if $C_2(T) = 0$ then in each column there is at most one 1; if $C_3(T) = 0$ then we have in total n 1-entries (together with $C_1(T)$ and $C_2(T)$ we therefore have exactly one 1 in each row and column). Note that for any choice of binary $t_{i,j}$ we have $C_1(T) \geq 0$, $C_2(T) \geq 0$, and $C_3(T) \geq 0$.

[75] https://www.claymath.org/millennium-problems/p-vs-np-problem

[76] See talk by William Cook; freely available on Youtube https://www.youtube.com/watch?v=q8nQTNvCrjE&t=35s.

[77] http://www.math.uwaterloo.ca/tsp/optimal/index.html

[78] Introduced by Hopfield and Tank in 1985; the paper is freely available at http://genomics.princeton.edu/tank/pdf-publications/neural%20computation%20of%20decisions%20in%20optimization,%20Hopfiedl%20&%20Tank.pdf.

The length of the tour represented by T is given by

$$F(T) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{\substack{k=1\\k\neq i}} d_{i,k}t_{i,j}\left(t_{k,j-1} + t_{k,j+1}\right)$$

where $t_{k,0} := t_{k,n}$ and $t_{k,n+1} = t_{k,1}$.

Hence, finding an optimal TSP tour is equivalent to solving the following unconstrained binary (quadratic) optimization problem

$$\min_{T\in\{0,1\}^{n\times n}} F(T) + \alpha_1 C_1(T) + \alpha_2 C_2(T) + \alpha_3 C_3(T), \tag{5.8}$$

where $\alpha_1, \alpha_2, \alpha_3 > 0$ need to be chosen suitably large (larger than any optimal tour; this would ensure that the minimum T^* of the problem in (5.8) would satisfy $C_1(T^*) = C_2(T^*) = C_3(T^*) = 0$).

Now, the optimization problem in (5.8) can be mapped to a Hopfield network (or Boltzmann machine) in the following way:

- Consider a Hopfield network with n^2 nodes $(i,j) \in \{1,\dots,n\}\times\{1,\dots,n\}$. The states $s_{i,j}$ should correspond to the variables $t_{i,j}$ from the optimization problem.
- The Hopfield network decreases the energy

$$H = H(s_{1,1},\dots,s_{n\times n}) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{\ell=1}^{n} w_{(i,j),(k,\ell)}s_{i,j}s_{k,\ell} + \sum_{i=1}^{n}\sum_{j=1}^{n} s_{i,j}\theta_{i,j};$$

  (see (5.6)). By comparing coefficient, set the $w_{(i,j),(k,\ell)}$ and $\theta_{i,j}$ to match the coefficients of $t_{i,j}t_{k,\ell}$ and, respectively, the constant term in $F(T) + \alpha_1 C_1(T) + \alpha_2 C_2(T) + \alpha_3 C_3(T)$ from (5.8).

Now, if we run the Hopfield network (for instance, with a randomly initialized binary input), update (for instance, asynchronously) by

$$s_{i,j} = \begin{cases} 1 & \text{if } \sum_k \sum_\ell w_{(i,j),(k,\ell)}s_{k,\ell} \geq \theta_{i,j},\\ 0 & \text{otherwise,}\end{cases}$$

and find a global minimum, then this minimum would be an optimal TSP tour.

5.6 Exercises

Exercise 5.1. Consider the task of clustering the set of data points $X = \{-2, 0, 6\} \subseteq \mathbb{R}$ into two clusters C_1 and C_2 using the *k-means algorithm* with $k = 2$.

(a) Start initially with the seeds $s_1 = -6$, and $s_2 = 3$, and show the first two iterations of k-means indicating at each iteration which points belong to each cluster and the coordinates of the two new cluster centers. In other words, fill in the tables below.

Iteration 1:

Data Point	**Cluster (C_1 or C_2)**
-2	
0	
6	

New cluster centers:

$s_1 =$	$s_2 =$

Iteration 2:

Data Point	**Cluster (C_1 or C_2)**
-2	
0	
6	

New cluster centers:

$s_1 =$	$s_2 =$

(b) TRUE or FALSE: No matter what the initial seeds s_1 and s_2 are, if a cluster becomes empty in iteration ℓ of k-means for the above three data points, it will remain empty in iteration $\ell + 1$. (Briefly explain your answer.)

Exercise 5.2. Given the set $X_\alpha := \{-6\alpha, 0, 2\alpha\} \subseteq \mathbb{R}$ of data points depending on a parameter $\alpha \in \mathbb{R}$ with $\alpha > 0$. Consider the task of clustering X_α into two clusters C_1 and C_2 using the *k-means algorithm* with $k = 2$.

(a) Start initially with the seeds $s_1 = -3\alpha$, and $s_2 = 6\alpha$, and show the first two iterations of k-means indicating at each iteration which points belong to each cluster and the coordinates of the two new cluster centers. In other words, fill in the tables below.

Iteration 1:

Data Point	**Cluster (C_1 or C_2)**
-6α	
0	
2α	

New cluster centers:

$s_1 =$	$s_2 =$

Iteration 2:

Data Point	**Cluster (C_1 or C_2)**
-6α	
0	
2α	

New cluster centers:

$s_1 =$	$s_2 =$

(b) TRUE or FALSE: There is an $\alpha > 0$ and initial seeds such that the k-means algorithm (for $k = 2$) clusters X_α into $C_1 = \{-6\alpha\}$, $C_2 = \{0, 2\alpha\}$ after iteration 1 while after iteration 2 the clustering $C_1 = \{-6\alpha, 0\}$, $C_2 = \{2\alpha\}$ is produced. (Briefly explain your answer.)

Exercise 5.3. Estimate, with help of a matlab implementation, the average (Euclidean) distance between two random points in a unit square.

Exercise 5.4. Let C denote the sample covariant matrix for a sample of $x_i = (\xi_1^{(i)}, \ldots, \xi_d^{(i)})^T$, $i = 1, \ldots, m$, data points, i.e.,

$$C = \begin{pmatrix} c_{1,1} & \cdots & c_{1,d} \\ \vdots & & \vdots \\ c_{1,1} & \cdots & c_{d,d} \end{pmatrix}, \qquad \text{with } c_{i,j} = \frac{1}{m-1} \sum_{\ell=1}^{n} \xi_i^{(\ell)} \xi_j^{(\ell)}, \qquad \text{for } i, j = 1, \ldots, d.$$

Verify that

$$C = \frac{1}{m-1} X \cdot X^T = \frac{1}{m-1} \sum_{\ell=1}^{m} x_\ell x_\ell^T,$$

where X is the $d \times n$ matrix holding x_ℓ in the ℓth column, $\ell = 1, \ldots, m$.

Exercise 5.5. Consider learning open intervals on the real line. The hypothesis class $\mathcal{H}$ consists of all intervals of the form (a, b), where $a < b$ and $a, b \in \mathbb{R}$. An open interval (hypothesis) $(a, b) \in \mathcal{H}$ is interpreted as a classifier that identifies a point x as being in class C if $a < x < b$, and identifies x as not being in class C if $x \leq a$ or $x \geq b$.

(a) Which sets containing two distinct points in $\mathbb{R}$ can be shattered by $\mathcal{H}$?

(b) Show that no set of three points in $\mathbb{R}$ can be shattered by $\mathcal{H}$.

(c) What is the VC-dimension of $\mathcal{H}$? Explain briefly how your answer follows from (a) and (b).

(d) Consider another hypothesis class $\mathcal{H}_2$. Each hypothesis in $\mathcal{H}_2$ is a union of two open intervals. $\mathcal{H}_2$ is the set of all such hypotheses (i.e., every union of two open intervals in $\mathbb{R}$). For example, $(2,3) \cup (5,8) \in \mathcal{H}_2$; that is the set of all points x such that $2 < x < 3$ or $5 < x < 8$.

What is the largest number of distinct points that $\mathcal{H}_2$ can shatter? Explain why no larger number can be shattered.

Exercise 5.6.

(a) Show that the decision boundary for the naïve Gaussian Bayes classifier is *quadratic*, i.e., show that the points of

$$\{(\xi_1^*, \ldots, \xi_d^*)^T \in \mathbb{R}^d : \Pr(y = -1 \mid \xi_1 = \xi_1^*, \ldots, \xi_d = \xi_d^*) = \Pr(y = +1 \mid \xi_1 = \xi_1^*, \ldots, \xi_d = \xi_d^*)\}$$

satisfy a quadratic equation in $\xi_1^*, \ldots, \xi_d^*$.

(b) Suppose the variances σ_{-1} and σ_{+1} of the likelihoods for both label classes are the same. Is the corresponding decision boundary then *linear*?

6 Epilogue

We have covered several fundamental concepts in the field of stochastic theory and methods in data science. Several of the concepts were only briefly discussed, and there is even more we haven't had time to discuss at all. Equipped with basic knowledge from this course, I hope that you are prepared and motivated to follow up and explore many of the new developments in the field.

What we did not cover at all, but which should be always considered, are the social and ethical consequences of any data collection, analysis, and on-this-based decisions/predictions/recommendations. It is always important to understand the underlying mathematics, the limitations, and shortcomings of the methods that one applies. A black-box can never explain the process that leads to the output. And who, in the end, wants to rely on a black-box when it comes to making, for instance, medical decisions?

What one also needs to consider is the reliability of the data. It does not make sense to apply sophisticated mathematical methods if, for whatever reason, the input data is plainly wrong. Incorrect or inconsistent data can lead to false conclusions. And even correct data, especially large data sets, may contain false relationships.

> "*Not everything that can be counted counts, and not everything that counts can be counted.*"
> (Albert Einstein, Physicist)

References

[1] P. Brémaud. *Discrete Probability Models and Methods*. Springer, Berlin, 2017.

[2] J. Haigh. *Probability Models*. Springer, Berlin, 2nd edition, 2013.

[3] M. Lefebvre. *Basic Probability Theory with Applications*. Springer, Berlin, 2009.

[4] A. Gut. *Probability: A Graduate Course*. Springer, New York, 2013.

[5] C. Lemieux. *Monte Carlo and Quasi-Monte Carlo Sampling*. Springer, Berlin, 2009.

[6] D. Knuth. *Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley Professional, Boston, 3rd edition, 1998.

[7] R. W. Shonkwiler and F. Mendivil. *Explorations in Monte Carlo Methods*. Springer, New York, 2009.

[8] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equations of state calculations by fast computing machines. *J. of Chem. Phys.*, 21:1097–1091, 1953. available at https://bayes.wustl.edu/Manual/EquationOfState.pdf.

[9] P. Brémaud. *Markov Chains: Gibbs Fields, Monte Carlo Simulation, and Queues*. Springer, Berlin, 1999.

[10] N. Privault. *Understanding Markov Chains*. Springer, Singapore, 2nd edition, 2018.

[11] Jerrum M. *Counting, sampling and integrating: algorithms and complexity*. Birkhäuser, Basel, 2003. see also: https://www.math.cmu.edu/ af1p/Teaching/MCC17/Papers/JerrumBook.

[12] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 6:721–741, 1984. available at https://ieeexplore.ieee.org/document/4767596.

[13] V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory Probab. Appl.*, 16:264–280, 1971.

[14] P.N. Tan, M. Steinbach, V. Kumar, and A. Karpatne. *Introduction to Data Mining*. Pearson, Harlow, 2nd edition, 2019.

[15] M. Wiering and M. von Otterlo, editors. *Reinforcement Learning*. Springer, Heidelberg, 2012.

[16] G. Dreyfus. *Neural Networks*. Springer, Berlin, 2nd edition, 2004.

[17] C. C. Aggarwal. *Neural Networks and Deep Learning*. Springer, New York, 2018.

[18] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc. Natl. Acad. Sci. U.S.A.*, 79:2554–2558, 1982. available at https://www.pnas.org/content/pnas/79/8/2554.full.pdf.

[19] D. H. Ackley, G. E. Hinton, and T. Sejnowski. A learning algorithm for Boltzmann machines. *Cogn. Sci.*, 9:147–169, 1985. available at https://onlinelibrary.wiley.com/doi/pdf/10.1207/s15516709cog0901_7.