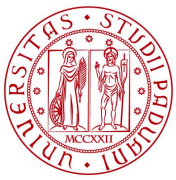

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

Corso di Laurea Triennale in Matematica

Markov Chain Monte Carlo methods for optimization problems

Relatore:
Prof. Marco Formentin

Laureanda: Greta Rachele Novello
Matricola: 2003945

Anno Accademico 2022/2023

data ufficiale di laurea: 22/09/2023

Contents

Introduction

In this thesis we will present one of the most powerful practical uses of the ergodic theory of Markov chains: Markov Chain Monte Carlo (MCMC). This generic problem occurs in diverse scientific applications, for instance Statistics, Computer Science, and Statistical Physics.

Chapter 1: The first chapter contains two basic parts, the first part focuses on discrete Markov chains, the main interest is in homogeneous and irreducible chains that will be classified as positive recurrent, recurrent null, or transient. The stationary distribution is also defined, outlining its existence and uniqueness for positive recurrent chains. The chapter concludes with the Ergodic theorem, which will give us an important result on convergence for homogeneous, irreducible and positive recurrent chain. The second part begins by defining the concept of periodicity, after we are going to present the classical Monte Carlo method by providing two examples with the Acceptance Rejection method and the Inverse Distribution method. Then the convergence of this method is defined using two fundamental theorems: The law of large numbers and The central limit theorem, continuing with two application examples of this last theorem. Following the methods of reducing variance are introduced and provide several techniques to speed up convergence. The last part focuses on the Monte Carlo Markov Chains method for value chains in a finite space and provides some examples including the famous Metropolis algorithm.

Chapter 2: in the second chapter we deal with a global search optimization algorithm: The Simulated Annealing algorithm. Describing the phases of this method and introducing the Neighborhood structure, finally providing a fundamental example for the understanding of this algorithm: the Metropolis Sampler, a Markov chain Monte Carlo method for obtaining a sequence of random samples from a probability distribution from which direct sampling is difficult, and The Travelling Salesman problem, which is a problem in combinatorial optimization, of the latter we will provide its implementation on python.

Much of the thesis material is taken from the books: *Markov Process and Applications* by Etienne Pardoux and *Markov Chains* by Pierre Bremaud.

Acknowledgements

I would like to thank my family especially my parents, Marialuisa Tognon and Giampiero Novello, to whom I dedicate this thesis, for allowing me to realize this dream and supporting me along this path.

I would also like to thank in a special way my boyfriend Leonardo Zen who believed in me from the first moment and was close to me in this path full of challenges, and thanks to all those people who have supported me in every single step up to this goal.

Chapter 1

Markov Chain Monte Carlo

1.1 Markov chain

1.1.1 Discrete Markov chain

A Markov chain is a stochastic process, but it differs from a general stochastic process in that a Markov chain must be *memory-less.* That is, (the probability of) future actions are not dependent upon the steps that led up to the present state. This is called the Markov property. While the theory of Markov chains is important precisely because so many *everyday* processes satisfy the Markov property, there are many common examples of stochastic features that do not satisfy the Markov property.

We wish to study Markov chains $\{X_n\}_{n\in\mathbb{N}}$ with values in a countable state space E.

Definition 1.1.1. *(Markov chain)*
The E-valued stochastic process $\{X_n; n \in \mathbb{N}\}$ is called a Markov chain whenever, for all $n \in \mathbb{N}$, the conditional law of X_{n+1} given $X_0, X_1, ..., X_n$ equals its conditional law given X_n that is, for all $i_0, i_1, ..., i_n, j \in E$,

$$\mathbf{P}(X_{n+1} = j | X_0 = i_0, ..., X_n = i_n) = \mathbf{P}(X_{n+1} = j | X_n = i_n).$$

A criterion,which allows us to verify that a given process is a Markov chain, is given by the following lemma:

Lemma 1.1.1. *Let E and F be two countable sets, and let f be a mapping from $\mathbb{N}\times E\times F$ into E. Let $X_0, Y_1, ...$ be mutually independent random variables, X_0 being E-valued, and the Y_n being F-valued. Let $\{X_n; n \geq 1\}$ be the E-valued process defined by*

$$X_{n+1} = f(n, X_n, Y_{n+1}) \qquad n \in \mathbb{N}.$$

Then $\{X_n; n \geq 1\}$ is a Markov chain.

Definition 1.1.2. *(Homogeneous Markov chain)*
Let $\{X_n\}_{n\geq 0}$ be a discrete-time stochastic process with countable state space E. If for all integers $n \geq 0$ and all states $i_0, ..., i_{n-1}, i, j$,

$$\mathbf{P}(X_{n+1} = j | X_n = i, X_{n-1} = i_{n-1}, ..., X_0 = i_0) = \mathbf{P}(X_{n+1} = j | X_n = i) \qquad (1.1.1.1)$$

whenever both sides are well-defined, this stochastic process is called a Markovian chain. It is called a homogeneous Markov chain (HMC) if in addition,the right-hand side of (1.1.1.1) *is independent of n.*

Definition 1.1.3. *(Transition matrix)*
The matrix $P = \{p_{i,j}\}_{i,j\in E}$, where

$$p_{i,j} = \mathbf{P}(X_{n+1} = j | X_n = i),$$

is called the transition matrix of the HMC. Since its entries are probabilities, and since a transition form any state i must be to some state, it follows that

$$p_{i,j} \geq 0, \qquad \sum_{k\in E} p_{i,k} = 1$$

for all states i, j.

As we will now see, the law of a Markov chain is entirely determined by the *initial law* $(\mu_x; x \in E)$, which is the law of X_0, and the transition matrix of the chain.

Proposition 1.1.1. *The distribution of a HMC only depends on its initial distribution and on the transition matrix.*

The matrix P^n is called the *n-step transition matrix* because its general term is

$$p_{i,j}(m) = \mathbf{P}(X_{n+m} = j | X_n = i),$$

Remark 1.1.1. *(Notation)*
Assuming that $A = \{X_{n+1} = j_1, .., X_{n+k} = j_k\}$, we use this notation $\mathbf{P}_i(A)$ to indicate $\mathbf{P}(A|X_0 = i)$, moreover if μ is a probability distribution on E, we define the probability of A given μ as a initial distribution $\mathbf{P}_\mu(A) = \sum_{i\in E} \mu(i)\mathbf{P}_i(A)$.

Now we will see more properties of Markov chains.

Definition 1.1.4. *(Communication)*
State j is said to be accessible from state i if there exists $M \geq 0$ such that $p_{i,j}(M) > 0$. In particular, a state i is always accessible from itself, since $p_{i,i}(0) = 1$. States i and j are said to communicate if i is accessible from j and j is accessible from i, and this is denoted by $i \longleftrightarrow j$.

The communication relation is an equivalence relation, and it generates a partition of the state space E into disjoint equivalence classes called *communication classes*.

Definition 1.1.5. *(Irreducibility)*
If there exists only one communication class, then the chain, its transition matrix are said to be irreducible.

Another important definition that will be useful for describing the *Markov Chain Monte Carlo methods* is described here:

Definition 1.1.6. *(Reversible Chain)*
One calls reversible a stationary Markov chain respect to the distribution π assumed positive if for all $i, j \in E$,

$$\pi(i)p_{i,j} = \pi(j)p_{j,i}.$$

1.1.2 Recurrent and Transient States

Consider a Markov chain its values in $E = \mathbb{N}$ there is a possibility that for any initial state $i \in \mathbb{N}$ the chain will never visit i after some finite random time. This is often an undesirable feature.

The good notion of a stability for an irreducible HMC is that of *positive recurrence*, when any given state is visited infinitely often and when, moreover, the average time between two successive visits to this state is finite. The principal problem is to find sufficient conditions guaranteeing stability. We begin with the *potential matrix* criterion and the *stationary distribution criterion.*
Let us first introduce an important notion: T_i, that denotes the *return* time to state i, i.e. $T_i = \inf\{n \geq 1; X_n = i\}$.

Definition 1.1.7. *(Recurrence and Transience)*
State $i \in E$ is called recurrent if

$$\mathbf{P}_i(T_i < \infty) = 1$$

and otherwise it is called transient.
A recurrent state $i \in E$ is called positive recurrent if

$$\mathbf{E}_i[T_i] < \infty$$

and the otherwise it is called null recurrent.

Now we introduce a particular matrix that will allow us to define an important criterion.

Definition 1.1.8. *(Potential matrix)*
The potential matrix $\mathbf{G}$ associated with the transition matrix P is defined by

$$\mathbf{G} = \sum_{n \geq 0} P^n$$

Its general term

$$g_{i,j} = \sum_{n=0}^{\infty} p_{i,j}(n) = \sum_{n=0}^{\infty} \mathbf{P}_i(X_n = j) = \sum_{n=0}^{\infty} \mathbf{E}_i[\mathbb{1}_{\{X_n=j\}}] = \mathbf{E}\left[\sum_{n=0}^{\infty} \mathbb{1}_{\{X_n=j\}}\right]$$

is the average number of visits to state j, given that the chain starts from state i.

Theorem 1.1.1. *(Potential Matrix Criterion)*
State $i \in E$ is recurrent if and only if

$$\sum_{n=0}^{\infty} p_{i,i}(n) = \infty.$$

A theoretical application of the potential matrix criterion is to the proof that recurrence is a (communication) class property.

Theorem 1.1.2. *(Recurrence is a Class Property)*
If i and j communicate, they are either both recurrent or both transient.

Proof. By definition, i and j communicate if and only if there exist integers M and N such that $p_{i,j}(M) > 0$, $p_{j,i}(N) > 0$. Going from i to j in M steps, then from j to j in n steps, then from j to i in N steps, is just one way of going from i back to i in $M+n+N$ steps.
Therefore, $p_{i,i}(M+n+N) \geq p_{i,j}(M)p_{j,j}(n)p_{j,i}(N)$. Similarly, $p_{j,j}(N+n+M) \geq p_{j,i}(N)p_{i,i}(n)p_{i,j}(M)$. Therefore, writing $\alpha = p_{i,j}(M)p_{j,i}(N)$ we have $p_{i,i}(M+n+N) \geq \alpha p_{j,j}(n)$ and $p_{j,j}(M+n+N) \geq \alpha p_{i,i}(n)$. This implies that the series $\sum_{n=0}^{\infty} p_{i,i}(n)$ and $\sum_{n=0}^{\infty} p_{j,j}(n)$ either both converge or both diverge. The potential matrix criterion concludes the proof.

□

An irreducible Markov chain has all its states of the same nature: transient, positive recurrent, or null recurrent so we should call it a transient chain, a positive recurrent chain, or a null recurrent chain and if one have to determine to which category it belongs, it suffices to study one state.

Definition 1.1.9. *An irreducible HMC is called positive recurrent if one state $i \in E$ is positive recurrent. In the same way null recurrent chain and transient chain are defined.*

1.1.3 Recurrence and Invariant Measures

The notion of invariant measure plays an important technical role in the recurrence theory of Markov chains.

Definition 1.1.10. *(Invariant Measure)*

A nontrivial vector $x = \{x_i\}_{i\in E}$ is called an invariant measure of the stochastic matrix $P = \{p_{i,j}\}_{i,j\in E}$ if for all $i \in E$,

$$x_i \in [0, \infty)$$

and

$$x_i = \sum_{j\in E} x_j p_{j,i}$$

(In abbreviated notation, $0 \leq x < \infty$ and $x^T P = x^T$.)

Definition 1.1.11. *(Stationary Distribution)*

A probability distribution π is called a stationary distribution of the transition matrix P if

$$\pi^T = \pi^T P \tag{1.1.3.1}$$

The *global balance equation* (1.1.3.1) says that for all states i,

$$\pi(i) = \sum_{j\in E} \pi(j) p_{j,i}$$

The following theorem allows us to delineate the invariant measure of the transition matrix P.

For the proof of this theorem we have to introduce the quantity

$$p'_{0,i}(n) = \mathbf{E}_0[\mathbb{1}_{\{X_n=i\}} \mathbb{1}_{\{n\leq T_0\}}] = \mathbf{P}(X_1 \neq 0, ..., X_{n-1} \neq 0, X_n = i).$$

This is the probability, starting from state 0, of visiting i at time n before returning to 0.

From the definition of x,

$$x_i = \sum_{n\geq 1} p'_{0,i}(n).$$

Theorem 1.1.3. *(Regenerative Form of Invariant Measure)*

Let P the transition matrix of an irreducible recurrent HMC $\{X_n\}_{n\geq 0}$. Let 0 be an arbitrary state and let T_0 be the return time to 0. Define for all $i \in E$

$$x_i = \mathbf{E}_0 \left[\sum_{n\geq 1} \mathbb{1}_{\{X_n=i\}} \mathbb{1}_{\{n\leq T_0\}} \right] \tag{1.1.3.2}$$

Then, for all $i \in E$,

$$x_i \in (0, \infty), \tag{1.1.3.3}$$

and x is an invariant measure of P.

Proof. We first prove

$$x_i = \sum_{j\in E} x_j p_{j,i}.$$

Observe that

$$p'_{0,i}(1) = p_{0,i}$$

and using first-step analysis, for all $n \geq 2$,

$$p'_{0,i}(n) = \sum_{j \neq 0} p'_{0,j}(n-1)p_{j,i}.$$

Summing up all the above equalities,and taking $x_i = \sum_{n \geq 1} p'_{0,i}(n)$ into account, we obtain

$$x_i = p_{0,i} + \sum_{j \neq 0} x_j p_{j,i},$$

since $x_0 = 1$ we have proved

$$x_i = \sum_{j \in E} x_j p_{j,i}$$

Next we show that $x_i > 0$ for all $i \in E$. Indeed, iterating

$$x_i = \sum_{j \in E} x_j p_{j,i}$$

we find $x^T = x^T P^n$, that is, since $x_0 = 1$,

$$x_i = \sum_{j \in E} x_j p_{j,i}(n) = p_{0,i}(n) + \sum_{j0} x_j p_{j,i}(n).$$

If x_i were null for some $i \in E$, $i \neq 0$, the latter equality would imply that $p_{0,i}(n) = 0$ for all $n \geq 0$, which means that 0 and i do not communicate, in contradiction to the irreducibility assumption.
It remains to show that $x_i < \infty$ for all $i \in E$. As before, we find that

$$1 = x_0 = \sum_{j \in E} x_j p_{j,0}(n)$$

for all $n \geq 1$, and therefore if $x_i = \infty$ for some i, necessarily $p_{i,0}(n) = 0$ for all $n \geq 1$, and this also contradicts irreducibility.

□

Another important result regarding invariant measures is the *Uniqueness of Invariant Measure theorem.*

Theorem 1.1.4. *(Uniqueness of Invariant Measure)*
The invariant measure of an irreducible recurrent stochastic matrix is unique up to a multiplicative factor.

Proof. In the proof of *Regenerative form of invariant measure theorem* we showed that for an invariant measure y of an irreducible chain $y_i > 0$ for all $i \in E$, and therefore, one can define, for all $i, j \in E$, the matrix Q by

$$q_{i,j} = \frac{y_i}{y_j} p_{i,j}.$$

It is a transition matrix,since $\sum_{i\in E} q_{j,i} = \frac{1}{y_j} \sum_{i\in E} y_i p_{i,j} = \frac{y_j}{y_j} = 1$.

The general term of Q^n is

$$q_{j,i}(n) = \frac{y_i}{y_j} p_{i,j}(n).$$

Indeed, supposing this true formula for n,

$$q_{j,i}(n+1) = \sum_{k\in E} q_{j,k} q_{k,i}(n) = \sum_{k\in E} \frac{y_k}{y_j} p_{k,j} \frac{y_i}{y_k} p_{i,k}(n)$$

$$= \frac{y_i}{y_j} \sum_{k\in E} p_{i,k}(n) p_{k,j} = \frac{y_i}{y_j} p_{i,j}(n+1),$$

and $q_{i,j}(n) = \frac{y_i}{y_j} p_{i,j}(n)$ follows, by induction, for all $n \geq 1$.

Clearly, Q is irreducible, since P is irreducible. Also, $p_{i,i}(n) = q_{i,i}(n)$, and therefore $\sum_{n\geq 0} q_{i,i}(n) = \sum_{n\geq 0} p_{i,i}(n)$ and this ensures that Q is recurrent by the potential matrix criterion. Call $g_{j,i}(n)$ the probability, relative to the chain governed by the transition matrix Q, of returning to state i for the first time at step n when starting from j. First-step analysis gives $g_{i,0}(n+1) = \sum_{j\neq 0} q_{i,j} g_{j,0}(n)$, that is, using

$$q_{i,j} = \frac{y_i}{y_j} p_{i,j}$$

$$y_i g_{i,0}(n+1) = \sum_{j\neq 0} (y_j g_{j,0}(n)) p_{j,i}.$$

Recall that $p'_{0,i}(n+1) = \sum_{j\neq 0} p'_{0,j}(n) p_{j,i}$, or equivalently,

$$y_0 p'_{0,i}(n+1) = \sum_{j\neq 0} (y_0 p'_{0,j}(n)) p_{j,i}.$$

We therefore see that the sequences $\{y_0 p'_{0,i}(n)\}$ and $\{y_i g_{i,0}(n)\}$ satisfy the same recurrence equation. their first terms ($n = 1$), respectively $y_0 p'_{0,i}(1) = y_0 p_{0,i}$ and $y_i g_{i,0}(1) = y_i q_{i,0}$, are equal in view of

$$q_{i,j} = \frac{y_i}{y_j} p_{i,j}.$$

Therefore, for all $n \geq 1$,

$$p'_{0,i}(n) = \frac{y_i}{y_0} g_{i,0}(n).$$

Summing up with respect to $n \geq 1$ and using $\sum_{n\geq 1} g_{i,0}(n) = 1$, we obtain the announced result $x_i = \frac{y_i}{y_0}$.

□

The equality

$$\sum_{i \in E} x_i = \mathbf{E}_0[T_0]$$

that we observed in the theorem *Regenerative Form of Invariant Measure* and the definition of positive recurrence give the following.

Theorem 1.1.5. *(Positive vs. Null Recurrence)*
An irreducible recurrent HMC is positive recurrent if and only if its invariant measure x satisfy

$$\sum_{i \in E} x_i < \infty.$$

1.1.4 Stationary Distribution Criterion

We now introduce the central notion of the stability theory of discrete-time HMCs.

Theorem 1.1.6. *(Stationary Distribution Criterion)*
An irreducible homogeneous Markov chain is positive recurrent if and only if there exists a stationary distribution. Moreover, the stationary distribution π is, when, it exists, unique, and $\pi > 0$.

Proof. The direct part follows from *Regenerative Form of Invariant Measure theorem* and *Positive vs.Null Recurrence theorem*. For the converse part, assume the existence of a stationary distribution π. Iterating $\pi^T = \pi^T P$, we obtain $\pi^T = \pi^T P^n$, that is, for all $i \in E$,

$$\pi(i) = \sum_{j \in E} \pi(j) p_{j,i}(n).$$

If the chain were transient,then, in view of the potential matrix criterion and the discussion following it, for all states i, j,

$$\lim_{n \uparrow \infty} p_{j,i}(n) = 0$$

and since $p_{j,i}(n)$ is bounded by 1 uniformly in j and n, by the dominated convergence theorem for series

$$\pi(i) = \lim_{n \uparrow \infty} \sum_{j \in E} \pi(j) p_{j,i}(n) = \sum_{j \in E} \pi(j) (\lim_{n \uparrow \infty} p_{j,i}(n)) = 0$$

This contradicts the assumption that π is a stationary distribution. The chain must therefore be recurrent and by Theorem *Positive vs. Null Recurrence* it is positive recurrence. The stationary distribution π of an irreducible positive recurrent chain is unique. Also recall that $\pi(i) > 0$ for all $i \in E$.

□

Theorem 1.1.7. *(Finite State Space and Positive Recurrence)*
An irreducible HMC with finite state space is positive recurrent.

Proof. We first show recurrence. If the chain were transient,then,from the potential matrix criterion and the observations following it,for all $i, j \in E$,

$$\sum_{n\geq 0} p_{i,j}(n) < \infty,$$

and therefore,since the state space is finite

$$\sum_{j\in E}\sum_{n\geq 0} p_{i,j}(n) = \sum_{n\geq 0} 1 = \infty.$$

a contradiction. Therefore, the chain is recurrent. By *Regenerative Form of Invariant Measure Theorem*, it has an invariant measure x.
Since E is finite, $\sum_{i\in E} x_i < \infty$, and therefore the chain is positive recurrent, by *Stationary Distribution Criterion Theorem.* □

1.1.5 Ergodic Theorem

This subsection is dedicated to the ergodic theorem for Markov chains. It gives conditions which guarantee that empirical averages of the type

$$\frac{1}{N}\sum_{k=1}^{N} f(X_k, ..., X_{k+1})$$

converge to probabilistic averages.

Proposition 1.1.2. *Let $\{X_n\}_{n\geq 0}$ be an irreducible recurrent HMC, and let x denote the canonical invariant measure associated with state $0 \in E$,*

$$x_i = \mathbf{E}_0\left[\sum_{n\geq 1} \mathbb{1}_{\{X_n=i\}}\mathbb{1}_{\{n\leq T_0\}}\right]$$

where T_0 is the return time to 0. Define for $n \geq 1$

$$v(n) = \sum_{k=1}^{n} \mathbb{1}_{\{X_k=0\}}.$$

Let f a function from E to $\mathbb{R}$ such that

$$\sum_{i\in E} |f(i)|x_i < \infty. \tag{1.1.5.1}$$

Then, for any initial distribution μ, $\mathbf{P}_\mu$-a.s,

$$\lim_{n\to\infty}\sum_{k=1}^{N} f(X_k) = \sum_{i\in E} f(i)x_i \tag{1.1.5.2}$$

Proof. Let $T_0 = \tau_1, \tau_2, ...$ be the successive return times to state 0, and define

$$U_p = \sum_{n=\tau_p+1}^{\tau_p-1} f(X_n).$$

In view of the regenerative cycle theorem, $\{U_p\}_{p\geq 1}$ is an i.i.d sequence. Moreover,assuming $f \geq 0$ and using the strong Markov property,

$$\mathbf{E}[U_1] = \mathbf{E}_0\left[\sum_{n=1}^{T_0} f(X_n)\right] =$$

$$= \mathbf{E}_0\left[\sum_{n01}^{T_0}\sum_{i\in E} f(i)\mathbb{1}_{\{X_n=i\}}\right] =$$

$$= \sum_{i\in E} f(i)\mathbf{E}_0\left[\sum_{n=1}^{T_0}\mathbb{1}_{\{X_n=i\}}\right] =$$

$$= \sum_{i\in E} f(i)x_i.$$

By hypothesis, this quantity is finite, and therefore the strong law of large numbers applies, to give

$$\lim_{n\uparrow\infty}\frac{1}{n}\sum_{p=1}^{n} U_p = \sum_{i\in E} f(i)x_i,$$

that is

$$\lim_{n\uparrow\infty}\frac{1}{n}\sum_{k=T_0+1}^{\tau_n-1} f(X_k) = \sum_{i\in E} f(i)x_i.$$

Observing that

$$\tau_{v(n)} \leq n < \tau_{v(n)+1},$$

we have

$$\frac{\sum_{k=1}^{\tau_{v(n)}} f(X_k)}{v(n)} \leq \frac{\sum_{k=1}^{n} f(X_k)}{v(n)} \leq \frac{\sum_{k=1}^{\tau_{v(n)-1}} f(X_i)}{v(n)}$$

Since the chain is recurrent, $\lim_{n\uparrow\infty} v(n) = \infty$, and therefore, from $\lim_{n\uparrow\infty}\frac{1}{n}\sum_{k=T_0+1}^{\tau_n-1} f(X_k) = \sum_{i\in E} f(i)x_i$ the extreme terms of the above chain of inequality tend to $\sum_{i\in E} f(i)x_i$ as n goes to ∞ and implies (1.1.5.2).
The case of a function f of arbitrary sign is obtained by considering (1.1.5.2) written separately for $f^+ = \max(0, f)$ and $f^- = \max(0, -f)$ and then taking the difference of the two equalities obtained this way. The difference is not an undetermined form $\infty - \infty$ due to hypothesis (1.1.5.1).

□

Theorem 1.1.8. *(Ergodic Theorem)*

Let $\{X_n\}_{n\geq 0}$ be an irreducible positive recurrent Markov chain with the stationary distribution π, and let f a function from E to $\mathbb{R}$ be such that

$$\sum_{i\in E} |f(i)|\pi(i) < \infty \tag{1.1.5.3}$$

Then for any initial distribution μ, $\mathbf{P}_\mu$-a.s.,

$$\lim_{n\to\infty} \frac{1}{N}\sum_{k=1}^{N} f(X_K) = \sum_{i\in E} f(X_k)\pi(i) \tag{1.1.5.4}$$

Proof. Apply Proposition 1.1.3 to $f = 1$. Condition (1.1.5.1) of that proposition is satisfied, since in the positive recurrent case $\sum_{i\in E} x_i < \infty$. Therefore, $\mathbf{P}_\mu - a.s$,

$$\lim_{N\uparrow\infty} \frac{N}{v(N)} = \sum_{j\in E} x_j.$$

Now f satisfying (1.1.5.3) also satisfies (1.1.5.1) of the proposition 1.1.3, since x and π are proportional, and therefore, $\mathbf{P}_\mu - a.s$

$$\lim_{N\uparrow\infty} \frac{1}{v(N)}\sum_{k=1}^{N} f(X_k) = \sum_{i\in E} f(i)x_i.$$

Combination of the above equalities gives,$\mathbf{P}_\mu - a.s$

$$\lim_{N\to\infty} \frac{1}{N}\sum_{k=1}^{N} f(X_k) = \lim_{N\to\infty} \frac{v(N)}{N}\frac{1}{v(N)}\sum_{k=1}^{N} f(X_k) = \frac{\sum_{i\in E} f(i)x_i}{\sum_{j\in E} x_j}$$

From which (1.1.5.4) follows, since π is obtained by normalization of x.

□

We can notice that the version of this theorem for Markov chains is a kind of strong law of large numbers, and it can be used in simulations to calculate quantities pf the type $\mathbf{E}_\pi[f(X_0)]$, when π is unknown.

1.2 Markov Chain Monte Carlo Simulation

In statistics, Markov Chain Monte Carlo (MCMC) methods comprise a class of algorithms for sampling from a probability distribution. By constructing a Markov chain that has the desired distribution as its equilibrium distribution, one can obtain a sample of the desired distribution by recording states from the chain.
The more steps that are included, the more closely the distribution of the sample matches the actual desired distribution. Various algorithms exist for constructing chains, including the Metropolis–Hastings algorithm.

The Markov chain property of MCMC is the idea that the random samples are generated by a special sequential process. Each random sample is used as a stepping stone to generate the next random sample (hence the chain). A special property of the chain is that, while each new sample depends on the one before it, new samples do not depend on any samples before the previous one (this is the "Markov" property).

1.2.1 Periodicity

Before introducing the Monte Carlo Markov Chain Method we need to present the meaning of *Aperiodic Markov Chain.*

Definition 1.2.1. *(Aperiodic)*
A state i is said to be aperiodic if there exists N such that

$$p_{i,i}(n) > 0, \forall n \geq N$$

Lemma 1.2.1. *If P is irreducible and there exists an aperiodic state i, then for all* $y, z \in E$ *there exists M such that* $p_{y,z}(n) > 0$*, for all* $n \geq M$*. In particular all states are aperiodic.*

Proof. From the irreducibility, there exist $r, s \in \mathbf{N}$ such that $p_{y,i}(r) > 0$, $p_{i,z}(s) > 0$. Moreover

$$p_{y,z}(r + s + n) \geq p_{y,i}(r)p_{i,i}(n)p_{i,z}(s)$$

if $n \geq N$. Hence we have the desired property with $M = N + r + s$. □

Definition 1.2.2. *(Aperiodic chain) An irreducible HMC is called aperiodic if one its state is aperiodic. Otherwise it is called periodic.*

Definition 1.2.3. *(Ergodic chain) An irreducible, positive recurrent and aperiodic HMC is called ergodic chain.*

(https://amslaurea.unibo.it/23629/1/Alessandro$_B orgia_t esi.pdf$)

1.2.2 Monte Carlo Method

Monte Carlo simulation, also known as the Monte Carlo method, is a mathematical technique that is used to estimate the possible results of an uncertain event. The Monte Carlo method was invented by John von Neumann and Stanislaw Ulam during World War II to improve decision-making under uncertain conditions.

We can now see in detail where the Monte Carlo Markov Chain method originated and what it consists of.
We want to evaluate the expectation $\mathbf{E}[\psi(\mathbf{Z})]$ of a random vector $\mathbf{Z}$ of dimension k with a probability density f(x) where ψ takes its values from $\mathbb{R}^k$ into $\mathbb{R}$ and $\mathbf{E}[\psi(\mathbf{Z})] < \infty$ the formula

$$\mathbf{E}[\psi(\mathbf{Z})] = \int_{\mathbb{R}^k} \psi(x) f(x) dx$$

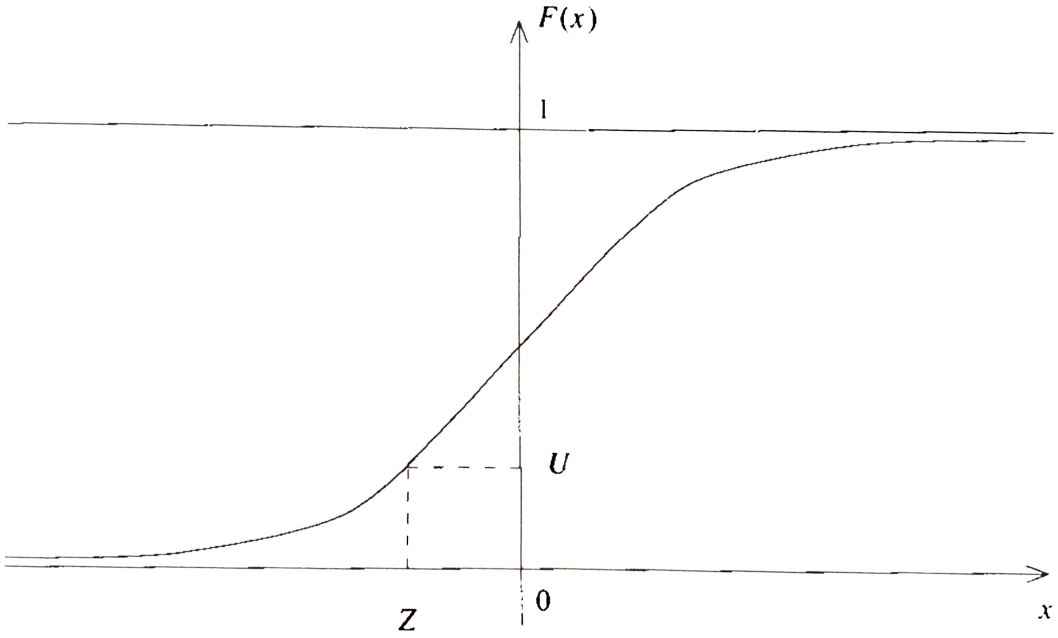

Figure 1.1: The inverse method

can be used if one is able to compute the integral analyitically.
A first way is numerical integration.
A second alternative is the method Monte Carlo, generating a sequence of i.i.d random vectors $\{Z_n\}_{n\geq 1}$ with the same distribution as Z and to invoke the strong law of large numbers

$$\mathbf{E}\left[\psi(\mathbf{Z})\right] = \left[\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} \psi(Z_i)\right]$$

to obtain an estimate based on a finite sample $(Z_1, ..., Z_n)$ namely $\frac{1}{n}\sum_{i=1}^{n}\psi(Z_i)$.

Remark 1.2.1. *In some applications, the Monte Carlo method is unavoidable, two of them are the method of the Inverse and Acceptance-Rejection.*

Definition 1.2.4. *(Inverse Distribution Method)*

It is a technique for generating a sample of random numbers distributed according to a given random distribution, known as its probability distribution function.

The Inverse distribution method uses a sequence of i.i.d random variables $\{U_n\}_{n\geq 1}$ *uniformly distributed on [0,1] And generates* $\{Z_n\}_{n\geq 1}$ *by*

$$Z_n = F^{-1}(U_n)$$

where $F^{-1}(u)$ *is the inverse function of* $u = F(x) = \int f(y)dy$.

Example 1.2.1. *The inverse function of* $F(x) = 1 - e^{\theta x}$ *is* $F^{-1} = -\frac{1}{\theta}\log(1-y)$*. Therefore, if* U *is uniformly distributed on* $[0,1]$*, then* $\frac{1}{\theta}\log(1-U)$ *is an exponential random variable with mean* θ^{-1}*. So is*

$$Z = \frac{1}{\theta}\log(U),$$

since U *and* $1-U$ *are identically distributed.*

In many cases it is not possible to apply the Inverse Distribution method because it is difficult to calculate the partition function or its inverse. A method alternative is that of Acceptance-Rejection.

Definition 1.2.5. *(Acceptance-Rejection Method)*

Rejection sampling is based on the fact that, to sample a random variable in one dimension, one can perform uniformly random sampling of the two-dimensional Cartesian graph and keep the samples in the region below the graph of its density function.
A random subset of the generated samples are rejected; the rest are accepted. The goal is for the accepted samples to be distributed as if they were from the target probability distribution.
Suppose one can generate a sequence of i.i.d random vectors Y_n *with the probability density g(x) satisfyng, for all x,*

$$\frac{f(x)}{g(x)} \leq c$$

for a finite costant c. Let U_n *be a sequence of i.i.d random variables uniformly distribuited on [0,1]. If we define* τ *to be the first index* $n \geq 1$ *for which*

$$U_n \leq \frac{f(Y_n)}{cg(Y_n)}$$

and let

$$Z = Y_\tau$$

so we accept it as a sample, then Z admits the probability density function $f(x)$ *and*

$$\mathbf{E}[\tau] = c.$$

Remark 1.2.2. *Both the inverse method and the acceptance-rejection method have disadvantages when Z is a discrete random variable with values in a finite space* $E = 1, 2, ..., r$*. We denote by* π *the distribution of* Z*.*
The inverse method is, in this case, always feasible: it generates a random variable U uniformly distributed on $[0,1]$ *and lets* $Z = i$ *if and only if* $\sum_{l=1}^{i-1}\pi(l) \leq U < \sum_{l=1}^{i}\pi(l)$*.*
The A-R method uses a distribution p on E such that $\frac{\pi(i)}{p(i)} \leq c < \infty$ *for all* $i \in E$*. the main problem is that* π *is often known only up to a normalizing factor.*

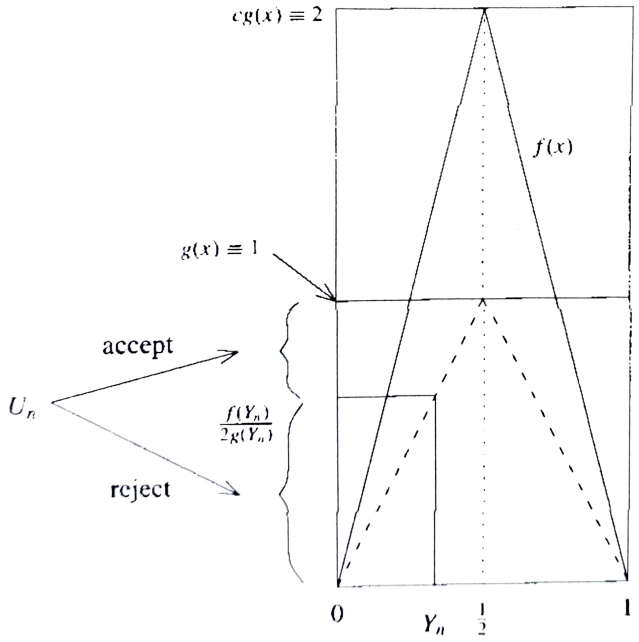

Figure 1.2: The acceptation-rejection method

1.2.3 Convergence Rates

In this section we deal with the problem of convergence and we will answer the two questions:
• When and why does this algorithm converge?
• Can we get a precise idea of the accuracy of this algorithm?
The answers to the two above questions are given by the two most fundamental theorems in the calculus of probability, namely *The law of large numbers*, which permits us to establish the convergence of the method, and *The central limit theorem*, which gives a precise indication of its rate of convergence.

Theorem 1.2.1. *(The law of large numbers)*
Let $(X_n, n \geq 1)$ *be a sequence of i.i.d. random variables, all having the law of* X*. If* $\mathbf{E}(|X|) < \infty$*, then, for* $\mathbf{P}$ *almost all* ω*.*

$$\mathbf{E}(x) = \lim_{n\to\infty} \frac{1}{n}(X_1 + ... + X_n)(\omega)$$

The evaluation of the method relies upon estimating the error

$$\xi_n = \mathbf{E}(X) - \frac{1}{n}(X_1 + ... + X_n)$$

The central limit theorem gives the asymptotic behaviour of the quantity ξ_n, which has a random nature. It says that the law of ϵ_n tends to look like a centred Gaussian law.

Theorem 1.2.2. *(The central limit theorem)*
Let $(X_n, n \geq 1)$ be a sequence of i.i.d. random variables,all having the laws of X. Assume that $\mathbf{E}(X^2) < \infty$. Let δ^2 denote the variance of X:

$$\delta^2 = \mathbf{E}(X^2) - \mathbf{E}(X)^2 = \mathbf{E}(X - \mathbf{E}(X))^2$$

then $\frac{\sqrt{n}}{\delta}\xi_n$ converges in law towards $\mathbf{Z} \simeq N(0,1)$
In other words, for all $a < b$,

$$\lim_{n\to\infty} \mathbf{P}\big(\frac{\delta}{\sqrt{n}}a \leq \epsilon_n \leq \frac{\delta}{\sqrt{n}}b\big) = \int_a^b e^{\frac{-x^2}{2}} \frac{dx}{\sqrt{2\pi}}$$

From this we can understand that if n is not too small the above probability can be replaced by its limit, then we may act as if ϵ_n were a centred Gaussian random variable with variance $\frac{\delta^2}{n}$.

Remark 1.2.3. *1. This outcome is powerful, because it gives us a rate of convergence which can be estimated with the help of the simulations that have already been realized. The fact that we have a reliable estimate of the error, without any further computation, is a real strength of the method.*

2. Anyway, the central limit theorem never gives a bound for the error, since the support of a Gaussian random variable is R. One way to describe the error in the Monte Carlo method is either by providing the standard deviation of ξ_n, which is equal to $\frac{\delta}{\sqrt{n}}$.
Or else, by providing a 95% confidence interval for the result.

We will now describe the use of the central limit theorem for analysing the rate of convergence of the Monte Carlo method, in two examples. This will allow us to see a limitation of the use of the Monte Carlo method.

Example 1.2.2. *(Good case)*
We want to compute $p = \mathbf{P}(Y \leq \lambda)$ where Y is a random variable with an arbitrary law. Define $X = \mathbb{1}_{\{Y\leq\lambda\}}$ then $\mathbf{E}(X) = p$ and $\sigma^2 = var(X) = p(1-p)$. After n independent draw $X_1, ..., X_n$ of X, we have

$$p_n = \frac{X_1 + ... + X_n}{n} \simeq p + \frac{\sigma}{\sqrt{n}Z}.$$

Since $p(1-p) \leq \frac{1}{4}$, if we want the standard deviation $\frac{\sigma}{\sqrt{n}}$ of the error to be order of 0.01, we should choose n of the order of 2500. If n=2500, the 0.95 confidence interval for p is then, according to the central limit theorem, $[p_n - 1.96 \times 0.01, p_n + 1.96 \times 0.01]$.
If the true unknown value p is of the order of 0.50, this leads to an acceptable error. However, if the true value of p is very small, the above value of n may be insufficient, if we want the error to be smaller than the quantity to be estimated.

Example 1.2.3. *(Though case)*

Imagine that we wish to compute $\mathbf{E}(exp(\beta Z))$ *where* Z *is a* $\mathbf{N}(0,1)$ *random variable, clearly*

$$\mathbf{E} = \mathbf{E}(\beta Z) = e^{\frac{\beta^2}{2}}$$

if we apply Method Monte Carlo to this case, we let $X = e^{\beta Z}$ *the variance of* X *is* $\delta^2 = e^{2\beta^2} - e^{\beta^2}$
after n *simulations* $X_1, ..., X_n$ *according to the law of* X*, we have*

$$\mathbf{E}_n = \frac{X_1 + ... + X_n}{n} \sim \mathbf{E} + \frac{\delta}{\sqrt{n}Z}$$

the standard deviation of the relative error is

$$\frac{\delta}{\mathbf{E}\sqrt{n}} = \left(\frac{e^{\beta^2-1}}{n}\right)^{1/2}$$

if we want the quantity to be smaller than a given $\epsilon > 0$ *we should choose* $n \simeq \epsilon^{-2}(exp(\beta^2) - 1)$.
If $\epsilon = 1$ *and* $\beta = 5$*, this means* $n = 7 \times 10^{10}$*, which is far too high. After 105 simulations, the 0.95 confidence interval might be [467 647, 2 176 181], which is a disaster. The positive point is that we get to know that our estimate is terrible.*

This example shows a practical limitation of the Monte Carlo method, when we use random variables with large variances. This fact leads us to define the following rule: in any Monte Carlo computation, one must exploit the simulations, in order to estimate the variance of the random variable whose expectation we wish to compute.

1.2.4 Variance reduction techniques

We have observed that the rate of convergence of the Monte Carlo method is of order $\frac{\delta}{\sqrt{n}}$. Clearly, the convergence is accelerated if the variance is reduced. We now present different variance reduction methods.

Importance sampling
We try to compute $\mathbf{E}(g(X))$ where the law of X is $f(x)dx$ so we have

$$\mathbf{E}(g(X)) = \int_{\mathbb{R}} g(x)f(x)dx$$

but if $\tilde{f}$ is the density of a probability such that $\tilde{f} > 0$ so one can rewrite $\mathbf{E}(g(X))$ as

$$\mathbf{E}(g(X)) = \int_{\mathbb{R}} \frac{g(x)f(x)}{\tilde{f(x)}} \tilde{f(x)}dx$$

this means that $\mathbf{E}(g(X)) = \mathbf{E}(\frac{g(Y)f(Y)}{\tilde{f(Y)}})$ where Y has the law $\tilde{f(x)}dx$.
Then, there is another method for computing $\mathbf{E}(g(X))$ using n simulations $Y_1, .., Y_n$ of Y and approximating $\mathbf{E}(g(X))$ by

$$\frac{1}{n}(\frac{g(Y_1)f(Y_1)}{f(\tilde{Y}_1)} + ... + \frac{g(Y_n)f(Y_n)}{f(\tilde{Y}_n)})$$

if we let $Z = \frac{g(Y)f(Y)}{\tilde{f(Y)}}$ then this alternative method improves the convergence provided $var(Z) < var(g(X))$.
It's easy to compute the variance of Z:

$$var(Z) = \mathbf{E}(Z^2) - \mathbf{E}(Z)^2 = \int_{\mathbb{R}} \frac{g(x)^2 f(x)^2}{\tilde{f(x)}} dx - \mathbf{E}(g(x))^2$$

if $g(x) \geq 0$ it is easy to see that choosing $\tilde{f(x)} = \frac{g(x)f(x)}{\mathbf{E}(g(x))}$ makes $var(Z) = 0$ this relies on the fact that we can compute $\mathbf{E}(g(x))$ exactly.

Control variate
This method involves writing $\mathbf{E}(f(X))$ in the form

$$\mathbf{E}(f(X)) = \mathbf{E}(f(X) - h(X)) + \mathbf{E}(h(X))$$

where $\mathbf{E}(h(X))$ cane be explicitly calculated and $var(f(X) - h(X))$ is significantly smaller than the $var(f(X))$.
We then use a Monte Carlo method for the computation of $\mathbf{E}(f(X) - h(X))$ and a direct computation for $\mathbf{E}(h(X))$.

Example 1.2.4. *Suppose we wish to compute* $\int_0^1 e^x dx$, *since near* $x = 0$, $e^x \sim 1 + x$ *we can write*

$$\int_0^1 e^x dx = \int_0^1 (e^x - 1 - x)dx + \frac{3}{2}.$$

It is easy to see that the variance is significantly reduced.

Stratification method
This method is well known in the context of survey sample design. We search to compute

$$I = \mathbf{E}(g(X)) = \int g(x)f(x)dx$$

where X has the law $f(x)dx$.
We start by decomposing I into

$$I = \sum_{i=1}^{m} I_i = \sum_{i=1}^{m} \mathbf{E}(\mathbb{1}_{\{x \in D_i\}})g(X)$$

where D_i is a partition of the integration set. We then use n_i simulations for the computation of I_i.
Define $\delta_i^2 = var(\mathbb{1}_{\{X \in D_i\}} g(X))$. Then the variance of the approximation is

$$\sum_{i=1}^{m} \frac{\delta_i^2}{n_i}$$

if we minimize this quantity with the constraint that $\sum_{i=1}^{m} n_i = n$ is fixed, we get $n_i = \frac{n\delta_i}{\sum_{i=1}^{m} \delta_i}$ the minimum equals $n^{-1}(\sum_{i=1}^{m} \delta_i)^2$. We can show that it is smaller that the variance obtained with n simulations of a standard Monte Carlo procedure. Of course, one can rarely compute the δ_i, which limits the use of this technique.

Mean value
Suppose we wish to compute

$$\mathbf{E}(g(X,Y)) = \int g(x,y) f(x,y) dxdy$$

where $f(x,y)dxdy$ is the law of the pair (X,Y).
If we let

$$h(x) = \frac{1}{m(x)} \int g(x,y) f(x,y) dxdy$$

with $m(x) = \int f(x,y)dy$, it is easy to check that

$$\mathbf{E}(g(X,Y)) = \mathbf{E}(h(X)).$$

Indeed, the law of X is $m(x)dx$, hence

$$\mathbf{E}(h(X)) = \int m(x)h(x)dx = \int dx \int g(x,y) f(x,y) dxdy = \mathbf{E}(g(X,Y)).$$

On the other hand, interpreting $h(X)$ as a conditional expectation, we can show that

$$var(h(X)) \leq var(g(X,Y))$$

consequently, if we can compute the function h explicitly, it is preferable to use a Monte Carlo procedure for $h(X)$.

1.2.5 Markov chain Monte Carlo

MCMC methods are a family of algorithms that uses Markov Chains to perform Monte Carlo estimate. Let be an irreducible aperiodic HMC $\{X_n\}_{n \geq 0}$ with the state space E with the distribution π. Because of E is finite the chain is ergodic and for any initial distribution μ and all $i \in E$

$$\lim_{n \to \infty} \mathbf{P}_\mu(X_n = i) = \pi(i)$$

and

$$\lim_{N\to\infty}\frac{1}{N}\sum_{n=1}^{N}\psi(X_n)=\mathbf{E}_\pi\big[\psi(\mathbf{X})\big]$$

observing the limit just written, when n is *large* we can consider that X_n has a distribution near to π. We want to know how accurately X_n imitates an E-valued random variable $\mathbf{Z}$ with distribution π.
We are searching to obtain estimates of the form

$$|\mu P^n-\pi|\leq \mathbf{A}\alpha^n$$

where $\alpha<1$
The basic problem is that of designing the MCMC algorithm one must find an ergodic transition matrix P on E, the stationary distribution of which π is the target distribution, so we seek solutions of the form

$$p_{i,j}=q_{i,j}\alpha_{i,j}$$

for j not equal to i and where $\mathbf{Q}=q_{i,j}$ is an arbitrary irreducible transition matrix on E, called the *candidate-generating* matrix.
When the present state is i, the next tentative state j is chosen with the probability $q_{i,j}$ and when j is different form i, the new state is accepted with the probability $\alpha_{i,j}$,otherwise the next state is i.
Now we have to select the acceptance probabilities $\alpha_{i,j}$ and there is a general form

$$\alpha_{i,j}=\frac{s_{i,j}}{1+t_{i,j}}$$

where $\Sigma=s_{i,j}$ is a symmetric matrix and

$$t_{i,j}=\frac{\pi(i)q_{i,j}}{\pi(j)q_{j,i}}$$

A popular Monte Carlo Method is the *Metropolis Algorithm.*

Example 1.2.5. *(Metropolis Algorithm)*

The method is based on generating proposed values that are accepted or rejected so as to converge to the distribution π desired.

in order to satisfy the constraint $\alpha_{i,j}\in[0,1]$. One must have

$$s_{i,j}\leq 1+\min(t_{i,j},t_{j,i})$$

that corresponds to the Metropolis algorithm

$$\alpha_{i,j}=\min\big(1,\frac{\pi(j)q_{j,i}}{\pi(i)q_{i,j}}\big)$$

This means that if $\frac{\pi(j)q_{j,i}}{\pi(i)q_{i,j}}$ is greater than 1 then j is accepted, otherwise j is accepted with the probability equals to that ratio.

Hastings proposed a generalization of this algorithm, the Metropolis algorithm is a special case of the general Metropolis-Hastings algorithm. The main difference is that the Metropolis-Hastings algorithm does not have the symmetric distribution requirement.

Example 1.2.6. *(Gibbs Sampler)*

The Gibbs Sampler is a version of the Metropolis-Hastings algorithm. Let $\mathbf{X} = (X_1, ..., X_n)$ *be a discrete random vector with a known probability distribution* $\pi(\mathbf{x})$ *less than a multiplication factor,suppose we want to generate a random vector with the distribution of* $\mathbf{X}$*,i.e. we want to generate a vector with probability distribution*

$$\pi(\mathbf{x}) = Cg(\mathbf{x})$$

where C *is known.*

To use the Gibbs Sampler we assume to generate a variable X *with distribution*

$$\mathbf{P}(X = x) = \mathbf{P}(X_i = x | X_j = x_j, j \neq i).$$

we use the algorithm of Metropolis Hastings on a Markov chain with states $\mathbf{x} = (x_1, ..., x_n)$ *and with transition probability defined as follows.*

if the present state is $\mathbf{x}$*, a uniform index* i *from* 1 *to* n *is chosen and a random variable* X *is generated with distribution:*

$$\mathbf{P}(X = x) = \mathbf{P}(X_i = x | X_j = x_j, j \neq i).$$

If $X = x$ *the proposed state is* $\mathbf{y} = (x_1, ..., x - i - 1, x, x_{i+1}, ..., x_n)$*, i.d. the* Q *matrix is*

$$q_{x,y} = \frac{1}{n}\mathbf{P}(X_i = x | X_j = x_j, j \neq i) = \frac{\pi(\mathbf{y})}{n\mathbf{P}(X_j = x_j, j \neq i)}$$

From the Metropolis-Hastings algorithm we have that $\mathbf{y}$ *is accepted with probability*

$$\alpha_{\mathbf{x},\mathbf{y}} = \min\left(\frac{\pi(\mathbf{y})q_{\mathbf{y},\mathbf{x}}}{\pi(\mathbf{x})q_{\mathbf{x},\mathbf{y}}}, 1\right) = \min\left(\frac{\pi(\mathbf{y})\pi(\mathbf{x})}{\pi(\mathbf{x})\pi(\mathbf{y})}, 1\right) = 1$$

i.e. it is always accepted. (https://amslaurea.unibo.it/23629/1/Alessandro$_B$*orgia*$_t$*esi.pdf)*

Example 1.2.7. *(Baker's Algorithm)*

If we choose $s_{i,j} = 1$*, we have Baker's algorithm for which*

$$\alpha_{i,j} = \frac{\pi(j)q_{j,i}}{\pi(j)q_{j,i} + \pi(i)q_{i,j}}$$

in this special case of purely random selection of the candidate

$$\alpha_{i,j} = \frac{\pi(j)}{\pi(i) + \pi(j)}$$

Chapter 2

Simulated Annealing

Simulated annealing algorithm is a global search optimization algorithm that is inspired by the annealing technique in metallurgy.

In the early 1980s three IBM researchers, Kirkpatrick, Gelatt and Vecchi, introduced the concepts of annealing in combinatorial optimization. These concepts are based on an important analogy with the physical annealing of materials. This process involves bringing a solid to a low energy state after raising its temperature. It can be summarized by the following two steps:
• Bring the solid to a very high temperature until "melting" of the structure;
• Cooling the solid according to a very particular temperature decreasing scheme in order to reach a solid state of minimum energy.
It is shown that the minimum-energy state is reached provided that the initial temperature is sufficiently high and the cooling time is sufficiently long. Comparing this algorithm with those previously explained we see that the main difference is the definition of the distribution during the process. Monte Carlo identifies solutions by simulating a distribution. Annealing changes the distribution definition as it searches, gradually putting tighter restrictions on the probabilistic definition of *neighbor* for each state transition.

2.1 Stochastic Descent and Cooling

Let E be a finite set and $U : E \longrightarrow \mathbb{R}$ a function, called the *cost function*, to be minimized. More precisely, one is looking for any element $i_0 \in E$ such that

$$U(i_0) \leq U(i)$$

for all $i \in E$. The element just defined is called a *global minimum* of the cost function U. The so-called descent algorithms define for each state $i \in E$ a subset $N(i)$ of E, called the *neighborhood* of i, and proceed iteratively as follows:
Suppose that at a given stage solution i is examined. At the next stage one examines a solution $j \in N(i)$ chosen according to a rule specific to each algorithm, and compares $U(i)$ to $U(j)$.

If $U(i) \leq U(j)$, the procedure stops and i is the retained solution. Otherwise a new solution $k \in N(j)$ is examined and compared to j, and so on.

The algorithm eventually ends and produces a solution, since E is finite. Anyway, this solution is usually not optimal, due to the possible existence of local minima that are not global minima, where i is called a *local minimum* if

$$U(i) \leq U(j)$$

for all $j \in N(i)$.
In most interesting situations, local optima exist, and the algorithms often become trapped at one of these local minima, but thanks to the role of temperature in these algorithms it is possible to avoid such local minimums as we will see.

Definition 2.1.1. *(Neighborhood structure)*
Let E be a finite set and let $\{N(i), i \in E\}$ be a collection of subsets of E satisfying condition

$$i \notin N(i).$$

Such a collection is called a neighborhood structure. If for all pairs of state $i, j \in E$ there exists a path from i to j, that is, a sequence of states $i_1, ..., i_m \in E$ such that $i_1 \in N(i), i_2 \in N(i_1), ..., j \in N(i_m)$,the neighborhood structure is called communicating.

Definition 2.1.2. *(Generating mechanism)*
A generating mechanism is a mean for selecting a solution j in any neighborhood $N(i)$ of a given solution i.

A local search algorithm is an iterative algorithm that begins its search from a feasible point, randomly drawn in the state space. A generation mechanism is then successively applied in order to find a better solution, by exploring the neighborhood of the current solution. If such a solution is found, it becomes the current solution.
The algorithm ends when no improvement can be found, and the current solution is considered as the approximate solution of the optimization problem.

The important idea of stochastic combinatorial optimization is to leave a possibility to escape from a local minimum trap.
A canonical form of the stochastic descent algorithm is as follows: let $Q = \{q_{i,j}\}$ be an irreducible transition matrix on E. Also, for each parameter value T, and all states $i, j \in E$, let $\alpha_{i,j}(T)$ be a probability. Calling X_n the current solution at stage n, the process $\{X_n\}_{n\geq 0}$ is a homogeneous Markov chain with state space E and transition matrix $P(T)$ of general off-diagonal term

$$p_{i,j}(T) = q_{i,j}\alpha_{i,j}(T).$$

We assume the chain irreducible. It is positive recurrent, since the state space is finite. Therefore, it has a unique stationary distribution π.

One possible choice of the candidate-generating matrix Q consists in first choosing a communicating neighborhood structure and taking $q_{i,j} > 0$ only if $i = j$ or $j \in N(i)$. The matrix Q is then irreducible. Conversely, one can associate with an irreducible transition matrix Q a communicating neighborhood structure defined by $N(i) = \{j; j \neq i, q_{i,j} > 0\}$.

Optimization of a solution involves evaluating the neighbours of a state of the problem, which are new states produced through conservatively altering a given state. For example, in the *travelling salesman problem* that we will see after, each state is typically defined as a permutation of the cities to be visited, and the neighbors of any state are the set of permutations produced by swapping any two of these cities. The well-defined way in which the states are altered to produce neighboring states is called a "move", and different moves give different sets of neighboring states. These moves usually result in minimal alterations of the last state, in an attempt to progressively improve the solution through iteratively improving its parts.

(https://en.wikipedia.org/wiki/Simulated$_a nnealing$)

Remembering that E represents the state space and U is the function to be minimized.

Definition 2.1.3. *Let (E, U) be a instantiation of a combinatorial minimization problem, and, i, j two points of the state space. The acceptance criterion for accepting solution j form the current solution i is given by the following probability:*

$$\alpha_{i,j} = \begin{cases} 1 & if \quad U(i) < U(j) \\ e^{(\frac{U(i)-U(j)}{T})} & else \end{cases}$$

2.2 The Metropolis Sampler

The Metropolis algorithm is a random walk adaptation combined with acceptance-rejection sampling which converges on a specified target distribution. Suppose that the current solution at stage n is i. At stage $n + 1$, a tentative solution j is selected according to the probability $q_{i,j}$.
This solution is accepted with probability

1. $$\alpha_{i,j} = \min\left(1, e^{\frac{(U(i)-U(j))}{T}}\right) = e^{-\frac{(U(j)-U(i))^+}{T}},$$

   where T is a positive constant; otherwise, j is rejected. The meaning of (1) is that if $U(i) \geq U(j)$, then j is accepted, if instead $U(i) < U(j)$, a chance is left to the solution j, although it is worse than i.
   This particular algorithm as well as in others, the chance left to a candidate j that is worse than i diminishes as its deviation from i, measured by $U(j) - U(i)$, increases. Suppose that the matrix Q is symmetric. With this special structure, the stationary distribution $\pi(T)$ does not depend on Q and is given by

2. $$\pi_i(T) = \frac{e^{-U(i)/T}}{\sum_{k \in E} e^{-U(k)/T}}.$$

   The stationary probability is given by (2). Define the set of global minima

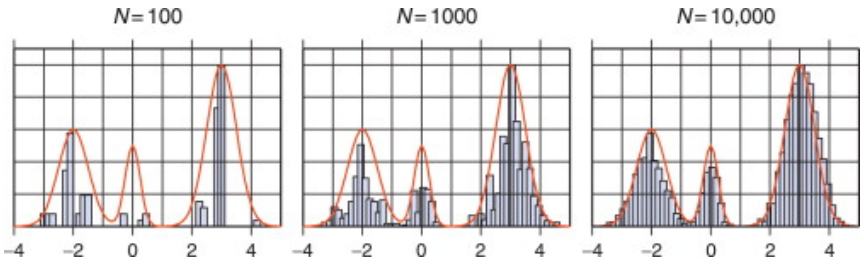

Figure 2.1: Histograms of samples generated by the Metropolis–Hastings algorithm from the target distribution (solid line) for small (left), intermediate (center), and large (right) numbers of samples. *Source*: Reprinted from De¸bski (https://www.sciencedirect.com/topics/mathematics/metropolis-hasting-algorithm)

3.

$$H = \{i \in E; U(i) \leq U(j), \mathrm{j} \in E\}$$

Then clearly, $\pi_i(T)$ is maximal on $i \in H$. But there is more:

4.

$$\lim_{T \to 0} \pi_i(T) = \begin{cases} \frac{1}{|H|} & if \quad i \in H \\ 0 & if \quad i \notin H \end{cases}$$

To see this, call $m = \min_{i \in E} U(i)$, and write the right-hand side of (4), after division of its numerator and denominator by $e^{-\frac{m}{T}}$ as

$$\frac{e^{-\frac{(U(i)-m)}{T}}}{|H| + \sum_{k \notin H} e^{-\frac{(U(k)-m)}{T}}}$$

The result follows, since as $T \downarrow 0$, $e^{-\frac{U(k)-m}{T}}$ tends to 0 if $U(k) > m$, and to 1 if $U(k) = m$.

This notion of slow cooling implemented in the Simulated annealing algorithm is interpreted as a slow decrease in the probability of accepting worse solutions as the solution space is visited. Accepting worse solutions allows for a more extensive search for the global optimal solution. In general, simulated annealing algorithms work as follows: the temperature progressively decreases from an initial positive value to zero. At each time step, the algorithm randomly selects a solution close to the current one, measures its quality, and moves to it according to the temperature-dependent probabilities of selecting better or worse solutions, which during the search respectively remain at 1 and decrease toward zero.(https://en.wikipedia.org/wiki/Simulated$_annealing$)

Simulated annealing algorithms all have a cooling schedule, that is, a sequence $\{T_n\}_{n \geq 0}$ of positive numbers decreasing to 0 controlling the transition rates of $\{X_n\}_{n \geq 0}$: at time n,

$$\mathbf{P}(X_{n+1} = j | X_n = i) = p_{i,j}(T_n).$$

The question becomes, How *slowly* must $\{T_n\}_{n \geq 0}$ converge to 0 so that

$$\lim_{n \to \infty} \mathbf{P}(X_n = i) = \begin{cases} 0 & if \quad i \notin H \\ \frac{1}{|H|} & if \quad i \in H \end{cases}$$

The answer to this question can be found in the implementation of the Travelling Salesman Problem, as we will see the temperature reduction is controlled by an alpha value: smaller values of alpha decrease temperature more quickly.

2.3 The travelling Salesman Problem

One of the most-studied problems in combinatorial optimisation is the *Travelling Salesman Problem*. In this problem, a traveling salesman has to visit all the K cities in a given list. The difficulty is that he has to do that by visiting each city only once, and by minimizing the traveled distance.
The traveling salesman problem is a minimization problem, since it consists in minimizing the distance traveled by the salesman during his tour. As the distance is what we want to minimize, it has to be our cost function. The parameters of this function are the cities in the list. Modifying a parameter of the function equals to changing the order of visit of the cities. Here E is the set of the $K!$ admissible routes, i is one route, and $U(i)$ is the length of route i.
In general applying the simulated annealing technique to the traveling salesman problem can be summed up as follow:

1. Create the initial list of cities by shuffling the input list.
2. At every iteration, two cities are swapped in the list.
3. If the new distance, computed after the change, is shorter than the current distance, it is kept.
4. If the new distance is longer than the current one, it is kept with a certain probability.
5. We update the temperature at every iteration by slowly cooling down.

(https://codecapsule.com/2010/04/06/simulated-annealing-traveling-salesman/)

In this case one choice for the neighborhood $N(i)$ of route i is all the routes j obtained form i by a 2-change:
if the cities are numbered $1, 2, ..., k$ a route i can be identified with a permutation δ of $\{1, 2, ..., k\}$, where $\delta(\alpha)$ is the order of the visit to city α in the route corresponding to δ. The 2-change involving cities α and β consists in cutting the segments $(\alpha, \alpha + 1)$ and $(\beta - 1, \beta)$, and in replacing them by the new segments $(\alpha, \beta - 1)$ and $(\alpha + 1, \beta)$. In this construction $|\beta - \alpha| \geq 3$ and $K \geq 4$, and one can count $K(K - 1) + 1$ neighbors of a given route.

Below are the figures that the Python program, whose code is in the appendix, gives as results by increasing the number of cities and observing how the temperature decreases.

The program consists of five codes: **Animated.py**

This code allows us to view the graph of the solution, that is, the minimum length journey. initializes the chart nodes and updates the solution for each frame.

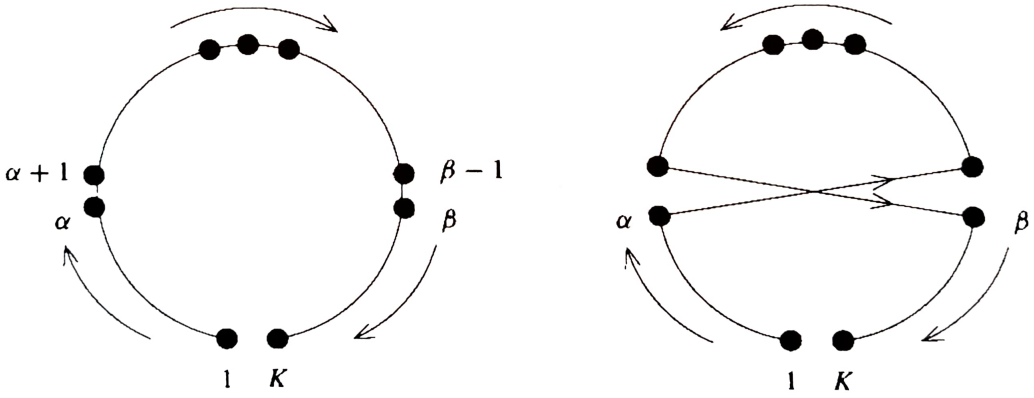

Figure 2.2: A 2-change of a salesman'route

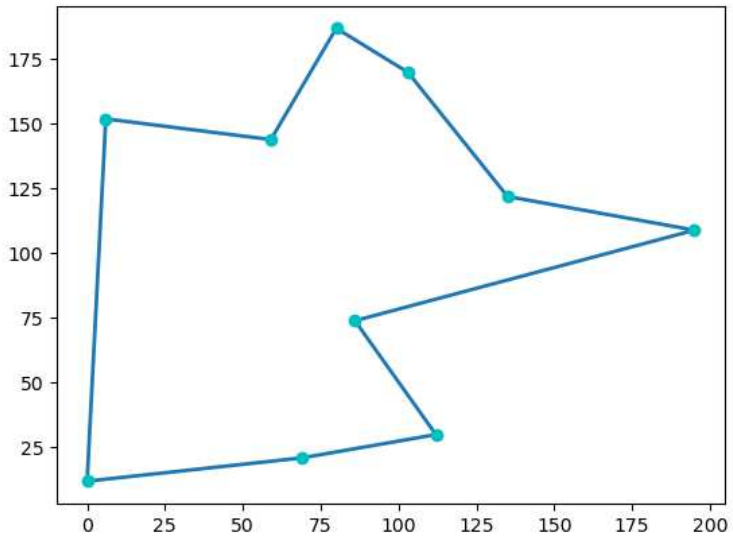

Figure 2.3: The minimum route (considering 10 cities to visit)

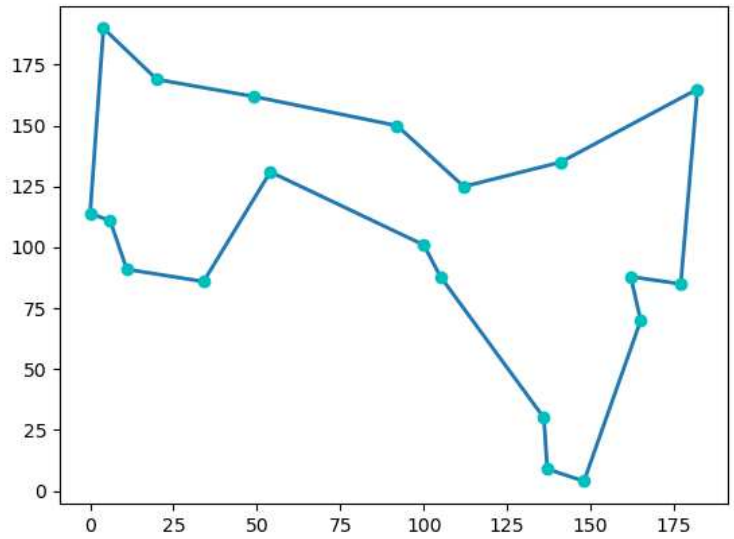

Figure 2.4: The minimum route (considering 20 cities to visit)

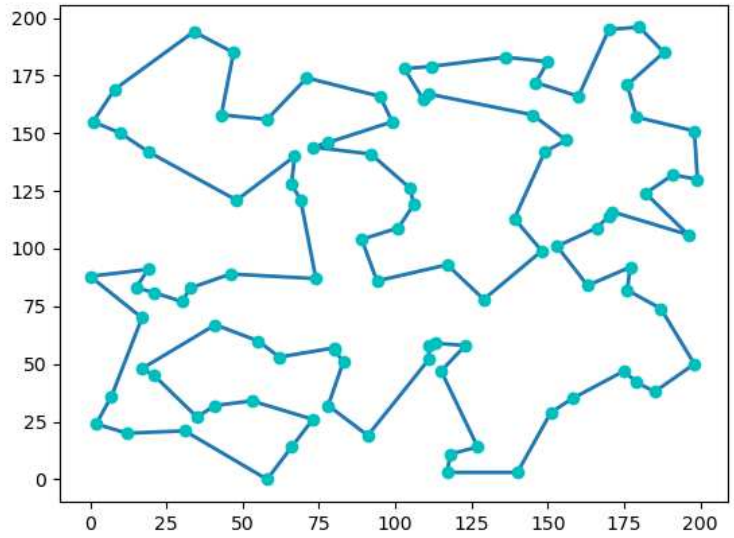

Figure 2.5: The minimum route (considering 100 cities)

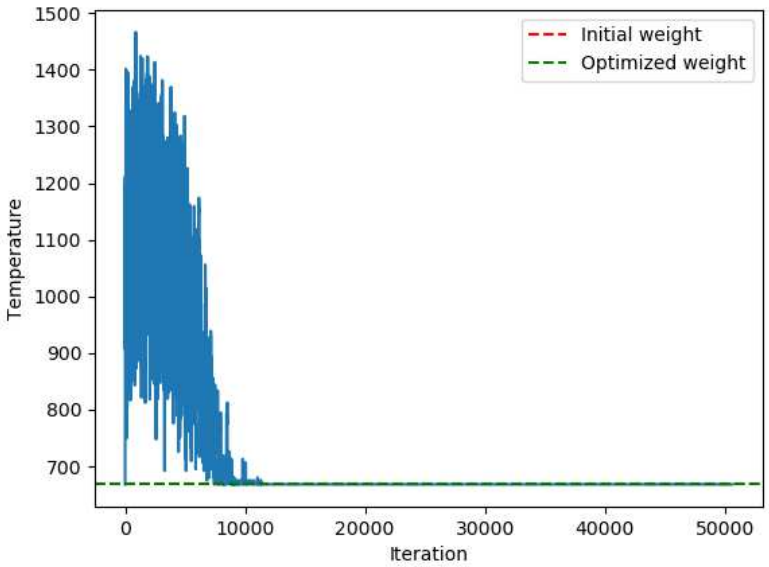

Figure 2.6: The temperature as a function of the iterations. The temperature reduction is controlled by the alpha value as we can see in the code. Suppose temperature = 1000.0 and alpha = 0.90, after the first processing iteration, temperature becomes 1000.0 * 0.90 = 900.0.And so on. Smaller values of alpha decrease temperature more quickly.

Cities.py

This part of the program randomly generates the coordinates of the cities , previously defining their positions according to the two axes of the Cartesian plane (width and height). It defines two vectors of integers corresponding to the location of the initial city and so on.

Simulated.py

This part of the program initially outlines the main elements that will serve the code as the alpha factor, it calculates the probability of acceptance using the temperature previously found and finally implement the Annealing method, reporting the graph

Utility.py

This code creates the distance array. First I define the vector with the distances of the cities and then the solution.

TravellingSalesman.py

This code defines the parameters of the algorithm as the maximum number of iterations, the number of cities that make up the route. Determines the Cartesian plane in which cities are represented. Then it runs the program and allows you to observe the graphs.

Chapter 3

Conclusions

In this thesis we analyzed Markov chains as applied to the Monte Carlo method (Chapter 1) which encapsulates a series of algorithms for sampling from a precise distribution. The guiding principle of Markov chains we have seen to be the central theme of such algorithms: in each MCMC method, the next sample is always chosen based on the previous sample. We have seen the usefulness of such methods for solving complex integrals and two examples of methods in which such a procedure is unavoidable (the Inverse Distrbution Method and Acceptance-Rejection Method)

The general structure of an algorithm belonging to the class of such methods was introduced with special attention to the form of the probability of choice $q_{i,j}$ and that of acceptance $\alpha_{i,j}$ with the special case of the Metropolis algorithm.

In the second part of the first chapter we studied the convergence of such methods using two important theorems: *The law of large numbers* and *The central limit theorem* the first allowed us to delineate the error and the second to define the convergence by concluding that one way to describe the error is either by providing the standard deviation of ϵ_n which is equals to $\frac{\delta}{\sqrt{n}}$ or by providing a 95% confidence interval for the result.

Thanks to these observations we are able to say that the rate of convergence of the Monte Carlo method is of the order of $\frac{\delta}{\sqrt{n}}$, and therefore that to accelerate the convergence of such methods we need to reduce the variance: variance reduction techniques.

In the second chapter, a global search optimization algorithm was presented: Simulated Annealing, which is extremely connected to the Monte Carlo Markov Chain methods as both explore a stochastic surface using the Metropolis acceptance criterion.

We dwelt on the structure of a descent algorithm which given a state i in the following step i examine state j by comparing the two values they have with respect to the cost function to be minimized: $U(i)$ and $U(j)$. We saw an important example: Metropolis Sampler

Finally we studied *The Travelling Salesman* problem by implementing it on python.

Appendix A

Python Code

Cities.py

```
import random
import numpy as np

#Random generates coordinates of cities
#In order to display them, it is necessary to define the maximum value
                                    of the coordinates and the number of
                                     cities in the route

class Generator:
    def __init__(self, width, height, cityNumber):
        self.width = width
        self.height = height
        self.cityNumber = cityNumber
        #It generates two vectors of integers corresponding to the
                                            location of the initial city
                                             and thus
    def generate(self):
        xs = np.random.randint(self.width, size=self.cityNumber)
        ys = np.random.randint(self.height, size=self.cityNumber)
        return np.column_stack((xs, ys))
```

Simulated.py

```
    import math
import random
import matplotlib.pyplot as plt
import utility
import animated
```

```
class Annealing:
    def __init__(self, coords, temp, alpha, stopping_temp, stopping_iter
                                            ):
        # animate the solution over time

        #Parameters
        #coords: array,list of coordinates
        #temp: float,initial temperature
        #alpha: float,rate at which temperature decreases
        #stopping_temp: float,temperature at which annealing process
                                                terminates
        #stopping_iter: integer, interation at which annealing process
                                                terminates



        self.coords = coords
        self.sample_size = len(coords)
        self.temp = temp
        self.alpha = alpha
        self.stopping_temp = stopping_temp
        self.stopping_iter = stopping_iter
        self.iteration = 1
        self.dist_matrix = utility.vectorToDistMatrix(coords)
        self.curr_solution = utility.Solution(self.dist_matrix)
        self.best_solution = self.curr_solution
        self.solution_history = [self.curr_solution]
        self.curr_temperature = self.temperature(self.curr_solution)
        self.initial_temperature = self.curr_temperature
        self.min_temperature = self.curr_temperature
        self.temperature_list = [self.curr_temperature]
        print('Intial temperature: ', self.curr_temperature)

    def temperature(self, sol):
        #It calculates temperature
        result=sum([self.dist_matrix[i, j] for i, j in zip(sol, sol[1:]
                                                + [sol[0]])])
        return result

    def acceptance_probability(self, candidate_temperature):

        #Acceptance probability as described in: https://stackoverflow.
                                                com/questions/19757551/
                                                basics-of-simulated-
                                                annealing-in-python

        return math.exp(-abs(candidate_temperature - self.
                                                curr_temperature) / self.
                                                temp)

    def accept(self, candidate):
```

```
        #Accept with probability 1 if candidate solution is better than
        # current solution, else accept with probability equal to the
                                            acceptance_probability()

        candidate_temperature = self.temperature(candidate)
        if candidate_temperature < self.curr_temperature:
            self.curr_temperature = candidate_temperature
            self.curr_solution = candidate
            if candidate_temperature < self.min_temperature:
                self.min_temperature = candidate_temperature
                self.best_solution = candidate
        else:
            if random.random()<self.acceptance_probability(
                                                candidate_temperature):
                self.curr_temperature = candidate_temperature
                self.curr_solution = candidate

    def anneal(self):

        #Annealing process with 2-opt
        while self.temp >= self.stopping_temp and self.iteration < self.
                                            stopping_iter:
            candidate = list(self.curr_solution)
            l = random.randint(2, self.sample_size - 1)
            i = random.randint(0, self.sample_size - l)
            candidate[i: (i + l)] = reversed(candidate[i: (i + l)])
            self.accept(candidate)
            self.temp *= self.alpha
            self.iteration += 1
            self.temperature_list.append(self.curr_temperature)
            self.solution_history.append(self.curr_solution)
            print('Minimum temperature: ', self.min_temperature)
            print('Improvement: ',round((self.initial_temperature - self
                                                .min_temperature) / (
                                                self.initial_temperature
                                                ), 4) * 100, '%')

    def animateSolutions(self):
        animated.Visualize(self.solution_history, self.coords)

    def plotLearning(self):
        plt.plot([i for i in range(len(self.temperature_list))], self.
                                            temperature_list)
        line_init = plt.axhline(y=self.initial_temperature, color='r',
                                            linestyle='--')
        line_min = plt.axhline(y=self.min_temperature, color='g',
                                            linestyle='--')
        plt.legend([line_init, line_min], ['Initial Temperature', '
                                            Optimized Temperature'])
        plt.ylabel('Temperature')
        plt.xlabel('Iteration')
        plt.show()
```

Animated.py

```
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
import numpy as np


def Visualize(history, points):
    ''' animate the solution over time

        Parameters
        ----------
        hisotry : list
            history of the solutions chosen by the algorith
        points: array_like
            points with the coordinates
    '''

    #Approx 1500 frames for animation
    key_frames_mult = len(history) // 1500

    fig, ax = plt.subplots()

    #Path is a line coming through all the nodes
    line, = plt.plot([], [], lw=2)

    def init():
        #Initialize node dots on graph
        x = [points[i][0] for i in history[0]]
        y = [points[i][1] for i in history[0]]
        plt.plot(x, y, 'co')

        #Draw axes slighty bigger
        extra_x = (max(x) - min(x)) * 0.05
        extra_y = (max(y) - min(y)) * 0.05
        ax.set_xlim(min(x) - extra_x, max(x) + extra_x)
        ax.set_ylim(min(y) - extra_y, max(y) + extra_y)

        #Initialize solution to be empty
        line.set_data([], [])
        return line,

    def update(frame):
        #For every frame update the solution on the graph
        x = [points[i, 0] for i in history[frame] + [history[frame][0]]]
        y = [points[i, 1] for i in history[frame] + [history[frame][0]]]
```

```
        line.set_data(x, y)
        return line

    #Animate precalulated solutions

    ani = FuncAnimation(fig, update, frames=range(0, len(history),
                                        key_frames_mult),
                        init_func=init, interval=10, repeat=False)

    plt.show()
```

Utility.py

```
import math
import random
import numpy as np
#It creates the distance matrix

def vectorToDistMatrix(coords):
    #The vector wih the distance of the cities
    te=coords[:, np.newaxis] - coords
    temp=(np.square(te).sum(axis=2))
    return np.sqrt(temp)

def Solution(dist_matrix):
    #It takes a random node in the vector from 0 to N
    node = random.randrange(len(dist_matrix))
    result = [node]

    nodes_to_visit = list(range(len(dist_matrix)))
    nodes_to_visit.remove(node)

    while nodes_to_visit:
        # Simplifies swapping

        nearest_node = min([(dist_matrix[node][j], j) for j in
                                            nodes_to_visit], key=lambda
                                            x: x[0])
        node = nearest_node[1]
        nodes_to_visit.remove(node)
        result.append(node)

    return result
```

TravellingSalesman.py

```
from cites import Generator
from simulated import Annealing


def main():
    #It determinates the parameters of the algorithm
    temperature = 1000
    stopping_temp = 0.00000001
    alpha = 0.9995
    max_iter = 10000000

    #Cartesian plane dimensions
    size_width = 200
    size_height = 200

    #Number of cities
    cities_size = 100
    #It generates a random list of cities

    cities= Generator(size_width, size_height, cities_size).generate()

    #run simulated annealing algorithm with 2-opt
    sa = Annealing(cities, temperature, alpha, stopping_temp, max_iter)
    sa.anneal()

    # It gives the graph
    sa.animateSolutions()

    #show the improvement over time
    sa.plotLearning()


if __name__ == "__main__":
    main()
```

Bibliography

[1] Etienne Pardoux. *Markov Process and Applications*, Algorithms,Networks, Genome and Finance. pp. 1-43.

[2] Pierre Bremaud. *Markov Chains*, Gibbs Fields, Monte Carlo Simulation, and Queues. pp. 290-311.

[3] Daniel Delahaye, Supatcha Chaimatanan and Marcel Mongeau. *Simulated Annealing:From Basics to Applications*, pp.3-8.