Final Project

MATHEMATICS DEGREE

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

PROBABILISTIC PROGRAMMING: the Monte Carlo boost

Author: Gutiérrez Galopa, Alejandro

Director: **Vitrià Marca, Jordi**
Departament de
Matemàtica i Informàtica

Barcelona, **June 2017**

*"Although our intellect always longs for clarity and certainty,
our nature often finds uncertainty fascinating."*

- Carl Von Clausewitz

Abstract

The main goal of this project is to demonstrate the existing symbiosis between Monte Carlo Markov Chain (MCMC) methods and Probabilistic Programming. By carrying out a thorough study of how MCMC methods work, including an analysis of their algorithms and convergence, the in and outs of Probabilistic Programming can be better understood.

One of the many applications of Probabilistic Programming is then employed in the study of the performance of a professional basketball player during his career. The model has been implemented using PyMC3, a Python package for sampling data using Monte Carlo Markov Chain methods.

Resumen

Este trabajo pretende ser una introducción a la Programación Probabilística, al mismo tiempo que enfatizar su estrecho vínculo con los métodos de Monte Carlo mediante cadenas de Markov, también conocidos como métodos MCMC (*Markov Chain Monte Carlo*). Haciendo un estudio en profundidad de los algoritmos de los métodos MCMC y analizando su convergencia, llegamos a entender los entresijos del proceso que sigue la Programación Probabilística.

Después de estudiar las cadenas de Markov, los métodos MCMC, y dar una explicación teórica sobre la Programación Probabilística, se usa ésta última para analizar el rendimiento de un jugador de baloncesto durante su carrera profesional. Esta funcionalidad, basada en la detección del cambio de tendencia en un conjunto de observaciones, es una de las muchas que ofrecen los modelos de Programación Probabilística.

Acknowledgements

I would like to thank Jordi Vitrià for all his support and tuition during these months, and all those close to me, especially my family, friends and classmates, all of whom have encouraged me and given me inspiration when I needed it most.
My thanks also to Rosemary who has kept my English on the straight and narrow.

Contents

Introduction

Nowadays, as a consequence of huge advances in computing, words like *modeling*, *estimation* and *prediction* are in fashion; almost every imaginable field is susceptible to statistical modeling. The motivation for choosing this topic comes from my interest in the probabilistic and statistical fields. Meanwhile, my passion for basketball leads me to see enormous potential in the application of statistical modeling to this field. This is the reason why, after considering some statistical-related topics, I decided to choose Probabilistic Programming as the central theme of my project, and apply it to an analysis of the performance of a professional basketball player to conclude this project.

Probabilistic Programming is a Bayesian modeling tool that operates by suggesting probabilistic models and then applying inference to reach the desired probability distribution. It is sustained on three pillars:

- ***Bayesian inference:*** Bayesian inference is a statistical procedure which uses both data and prior knowledge to apply statistical inference using the Bayes Theorem. Starting from a prior probability density, the model is repeatedly updated by using inference. The desired distribution is obtained once inference reaches convergence.

  In the Bayes formula
  $$P(A \mid X) = \frac{P(X \mid A)P(A)}{P(X)}$$
  X is the evidence observed, $P(A \mid X)$ the updated posterior probability, and $P(A)$ the prior probability. By using inference, we eliminate the need to analytically calculate $P(X)$, which most of the time is not possible.

- ***Markov Chain Monte Carlo methods:*** Monte Carlo methods as we know them nowadays have existed for a hundred years, but applying them to solve real problems was somewhat time-consuming due to their difficult implementation. Then, with advances in the computational field, the use of MCMC methods became more and more frequent. Probabilistic Programming uses MCMC methods to implement inference to models in order to find the distribution of the random variables.

- ***Law of Large Numbers:*** The interpretation of the Law of Large Numbers is that, when data observed for a certain random variable is large enough, the mean of the observations will be the expected value of the random variable (or at least it will be close enough). Probabilistic Programming, when applying inference, takes into account the Law of Large Numbers and this will be seen in the fourth chapter.

Objectives and structure of the project

The aim of the project is to show the interdependence between Monte Carlo Markov Chain methods and Probabilistic Programming, and how the former have gained importance in the statistical modeling field on account of the latter. The following paragraph details how the project has been structured in order to accomplish this objective.

The project begins by setting the context for Probabilistic Programming, and giving a brief historical introduction of MCMC methods and their evolution from their origin. After that, some basic definitions for Markov chains are provided, followed by the presentation of the Fundamental Theorem of Markov chains, which is necessary in order to understand the convergence of MCMC methods. In Chapter 3, MCMC methods are analyzed in-depth. Emphasis has been placed on this analysis in order to study Probabilistic Programming from a mathematical perspective, and the core of the Probabilistic Programming mechanism is MCMC methods. Once MCMC methods have been discussed, the fourth chapter looks at Probabilistic Programming, introducing Bayesian statistics and discussing the synergy between MCMC methods and Probabilistic Programming. Finally, in the last chapter of the project, Probabilistic Programming is carried out in order to detect the existence of a pattern change in the performance of a professional basketball player.

The Appendix includes three essential theorems for the development of this project, the Bayes Theorem, the Law of Large Numbers, and the Central Limit Theorem, as well as some definitions of concepts mentioned during the project and all the Python pieces of code employed for the realization of the project.

Chapter 1

A bit of history

The aim of this chapter is to provide some background to Monte Carlo methods. First, the Buffon's needle experiment is explained as a first approach to Monte Carlo methods. The second and third sections of the chapter offer a brief explanation of the origins of Monte Carlo methods together with some key historical dates.

1.1 Buffon's needle experiment

Buffon's needle experiment, in 1777, is the first approximation to a Monte Carlo method. It is considered to be one of the oldest problems in the field of geometrical probability, and its goal is to find an estimation of the value of π.
Studying this experiment will provide an introduction to understanding the basic idea behind the Monte Carlo methods. Let's take the easiest case of the problem, where we have needles with a length of one unit and an infinite rectangular board with vertical lines at a distance of 1 unit between each (the problem also has a resolution for a general case with an undetermined needle length and also undetermined distance between the lines). We place a needle randomly on the board. Let's call D the distance from the centre to the closest vertical line, and θ the angle formed by the needle and the x axis, as Figure 1.1 shows:

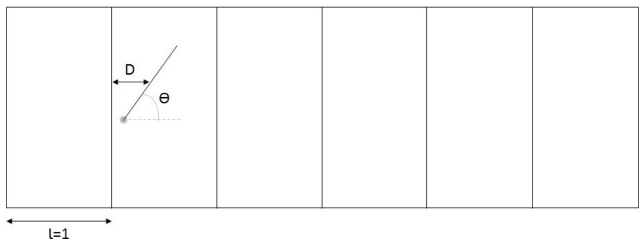

Figure 1.1: Graphical idea of Buffon's needle experiment.

A needle crosses the closest line when $D < \frac{1}{2}sin(\theta)$. We will not prove this as it will mean going too deep into our example. So, how do we relate this to an estimation of the number π? Plotting a graph where the x axis shows the possible values for θ and the y axis is the distance from the center of the needle to the nearest line, the area below the function $f(\theta) = \frac{1}{2}sin(\theta)$ represents the crossings; i.e., where $D < \frac{1}{2}sin(\theta)$.
Figure 1.2 below shows the mentioned graph:

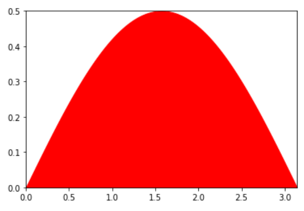

Figure 1.2: Geometrical interpretation of the approximation for number π.

Therefore, the probability of a needle crossing a line is the ratio defined by the red area over the rectangle (of area $\frac{\pi}{2}$) in the graph above (section A.2.1 in the Appendix includes all the Python code employed for generating the graph).
The red area can be computed using the following integral:

$$\int_0^\pi \frac{1}{2}sin(\theta)d\theta = -\frac{1}{2}(cos(\pi) - cos(0)) = 1$$

Therefore, the probability of a needle crossing is

$$\frac{1}{\frac{\pi}{2}} = \frac{2}{\pi}$$

This gives us the following approximation of the number π:

$$\pi \approx \frac{2 * \text{Drops}}{\text{Crossings}},$$

where Drops means the total needles dropped on the board, and Crossings is the total number of needles which cross a vertical line.

1.2 Monte Carlo method and its origins

In the late 1930s, Enrico Fermi[1] used Monte Carlo simulation to calculate neutron diffusion, but the origin of Monte Carlo methods is in the late 1940s. Stanislaw Ulam, together with John Von Neumann, was the promoter of Markov Chain Monte Carlo methods, as they are known nowadays; Von Neumann and Ulam decided to apply to the study of neutron diffusion the method for calculating the probabilities of winning at

[1]Enrico Fermi (September 29th, 1901 - November 28th, 1954) was an Italian physicist, awarded with the 1938 Nobel Prize in Physics.

Solitaire. The impact of this was an immediate introduction of the methods in the fields of calculation and physics; the ENIAC [4] computer and the Los Alamos Laboratory [5] radiation investigation were the first foci of MCMC (Markov Chain Monte Carlo) methods. As a curiosity, the name *Monte Carlo methods* was chosen in honor of Ulam's uncle, who would borrow money from relatives saying that he "just had to go to Monte Carlo", referring to his addiction to gambling.
In 1953, the first MCMC method, Metropolis algorithm, was published by Nicholas Metropolis[2] in the *The Journal of Chemical Physics*[3]. N. Metropolis and his research team focused on solving the integrals

$$\int F(\theta)e^{\frac{-E(\theta)}{kT}}\,d\theta,$$

where T is temperature, k is a constant (known as the Boltzmann constant), and E is energy, defined as

$$E(\theta) = \frac{1}{2}\sum_{i=1}^{n}\sum_{i\neq j} P(d_{ij}),$$

where P is is a potential function, and d_{ij} the distance between the i-th and the j-th component of θ.
Clearly, direct integration is not possible, and the Monte Carlo techniques in existence were not enough. Hence, Metropolis created this algorithm which increased efficiency by modifying the random walk that generates the Markov Chain.

1.3 Recent history

1970, Metropolis-Hastings algorithm. Wilfred K. Hastings[4] extended the Metropolis algorithm and created one of the most important and most employed MCMC methods, the Metropolis-Hastings algorithm. With the assistance of one of his students, Peter H. Peskun, he developed the algorithm as a simulation tool which could manage multidimensionality.

1974, Hammersley-Clifford theorem. Hammersley, Clifford and Besag developed their work in analyzing joint distributions. Their research ended up in what is known as the Hammersley-Clifford theorem, which provides a relation between joint distributions and maximal subsets between these elements.

[2]Nicholas Metropolis (June 11th, 1915 - October 17th, 1999) was a Greek-American physicist who collaborated with Enrico Fermi and then worked for Los Alamos National Laboratory.

[3]The Journal of Chemical Physics is a scientific journal published by the American Institute of Physics, established in 1933.

[4]W.K. Hastings (July 21st, 1930 - May 13th, 2016) was a Canadian statistician who completed his Ph.D. and worked for the University of Toronto.

1984, Gibbs sampling. Geman and Geman's paper *Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images*[5] popularised Gibbs sampling, an MCMC method based on joint distributions. Gibbs sampling comes from a previously developed algorithm for image processing which Geman and Geman introduced into the statistical field. The name *Gibbs sampling* was in honor of Josiah W. Gibbs[6], because in their paper Geman and Geman used the method to study Gibbs random fields.

1987, Pearl algorithm. Pearl algorithm meant a huge contribution of MCMC methods to artificial intelligence. The algorithm is based on obtaining a Markov chain simulating from the conditional distribution of each variable assuming the current values of the other variables (which in fact can be interpreted as a variation of Gibbs sampling).
Since then, artificial intelligence and MCMC methods have been closely tied; MCMC methods are a powerful tool for generating stochastic simulations to estimate inference.

1990, Gelfland and Smith. Alan E. Gelfland and Adrian F.M. Smith published a paper in the *Journal of the American Statistical Association*[7] about approximating marginal densities by using sampling MCMC methods. This publication led to MCMC being widely applied in the statistical field; thereafter, many statistical articles consider MCMC as a calculation tool for statistical distributions which cannot be computed analytically.

1994, Luke Tierney. Luke Tierney[8] published a paper proposing Gibbs sampling and Metropolis algorithm as two powerful tools for sampling from a posterior distribution. In his paper, Tierney gives some guidelines about how more optimal algorithms can be developed using some theoretical results such as Central Limit Theorem and Law of Large Numbers.

1994, Mergensen and Tweedie. Kerrie L. Mengersen[9] and Richard L. Tweedie[10] provided the necessary conditions for the convergence of both the Metropolis and Metropolis-Hastings algorithms and established some theoretical convergence bounds. These results provided a further understanding of these algorithms and optimized their application, as well as increased their frequency of use in many different fields.

[5]Paper written by Stuart Geman and Donald Geman, two American mathematicians who worked on machine learning and pattern recognition.

[6]J.W. Gibbs (February 11st, 1839 - April 28th, 1903) was an American scientist who developed his work in the field of thermodynamics; one of his major contributions was Gibbs free energy.

[7]Also known as JASA, the Journal of the American Statistical Association is considered to be the leading publication in the USA in the field of statistics.

[8]Luke Tierney is an American Statistician who has developed his research in Bayesian Methods, MCMC and Statistical Computing. He leads the Department of Statistics and Actuarial Science at the University of Iowa.

[9]Kerrie L. Mengersen (January 1st, 1962 -) is an Australian Statistician and Mathematician. She is a Distinguished Professor of Queensland University and is a member of the Australian Research Council for Mathematical and Statistical Frontiers.

[10]Richard L. Tweedie (August 22nd, 1947 - June 7th, 2001) was a Statistician who dedicated his research to Markov chains and statistical methodology for epidemiology.

21st century, Probabilistic Programming. Advances in computing have made possible the programming of powerful tools for the creation of probabilistic models by using MCMC methods. Probabilistic Programming has been applied to many different fields:

- Finance: Creation of models for investment purposes which measure share volatility, price and stability.
- Business: Creation of models for pricing optimization, competition analysis and marketing campaigns analysis, among others.
- Machine learning: Creation of models for image recognition, NPL, and many other purposes. Also projects with ambitious goals such as *PPAML(Probabilistic Programming for Advanced Machine Learning)* have been created.
- Engineering: Another field where MCMC methods have made the difference. As in Machine Learning, some promising projects have arisen such as *MIT Computing Probabilistic Programming Project*.

Chapter 2

Markov chains

Before going into detail into the different MCMC methods, which are covered in Chapter three, an introduction to Markov chains is essential.
The main goal of this chapter is to present and demonstrate the Fundamental Theorem of Markov Chains. Before that, some definitions are provided to ensure comprehension and proper presentation of the theorem.

2.1 Definitions related to Markov chains

Definition 2.1. Let $\mathcal{C} = \{x_0, x_1, ...\}$ be a sequence of random variables, and $\mathbb{S} = \{s_1, ..., s_n\}$ the possible values the random variables can take (we call $\mathbb{S}$ the state space of $\mathcal{C}$).
Thus, $\mathcal{C}$ is called a ***Markov Chain with state space*** $\mathbb{S}$ if the transition probabilities of jumping from a state to another only depend on the last state.

$$P(x_{k+1} = s_{k+1} \mid x_k = s_k, ..., x_1 = s_1) = P(x_{k+1} = s_{k+1} \mid x_k = s_k)$$

Definition 2.2. A Markov Chain is ***homogeneous*** if its transition probabilities do not change over time, i.e.,

$$\forall m, n \in \mathbb{N}\ ,\ p_{ij}(m) := P(x_{m+1} = j \mid x_m = i) = P(x_{n+1} = j \mid x_n = i) =: p_{ij}(n)$$

The following theorem relates the joint probability distributions for different sets:

Theorem 2.3. (Chapman-Kolmogorov equation) *Let $X = \{x_1, x_2, ..., x_n\}$ be a set, and $\mathbb{S}$ its state space. Denoting $p_{ab}^{(t)}$ as the probability $P(x_t = b \mid x_0 = a)$, for all $i, k \in \mathbb{S}$, $n, m \in \mathbb{N}$, $0 \leq n \leq m$, gives*

$$p_{ik}^{(m+n)} = \sum_{j \in \mathbb{Z}} p_{ij}^{(n)} p_{jk}^{(m)}$$

Proof.

$$\begin{aligned}
p_{ik}^{(m+n)} &= P(x_{m+n} = k \mid x_0 = i) = \sum_{j\in\mathbb{Z}} P(x_{m+n} = k\,,\, x_n = j \mid x_0 = i) = \\
&= \sum_{j\in\mathbb{Z}} P(x_{m+n} = k \mid x_n = j\,,\, x_0 = i) P(x_n = j \mid x_0 = i) = \\
&= \sum_{j\in\mathbb{Z}} P(x_{m+n} = k \mid x_n = j) P(x_n = j \mid x_0 = i) = \sum_{j\in\mathbb{Z}} p_{jk}^{(m)} p_{ij}^{(n)}
\end{aligned}$$

□

Definition 2.4. Let i and j be two states of a Markov Chain ($i, j \in \mathbb{S}$). We say i and j are ***communicated states*** ($i \leftrightarrow j$) if we can jump from i to j ($i \rightarrow j$) and from j to i ($j \rightarrow i$).

Definition 2.5. A Markov Chain is called ***irreducible*** if all the states are communicated, i.e.,

$$\forall i, j \in \mathbb{S},\ i \leftrightarrow j$$

Definition 2.6. Let $i, j \in \mathbb{S}$ be two states of a Markov Chain $\mathcal{C}$. We denote p_{ij} as the probability of jumping from the state i to the state j. We define the ***transition matrix*** of $\mathcal{C}$ as $P_{\mathcal{C}} = (p_{ij})$.

Definition 2.7. The ***period*** of a state $j \in \mathbb{S}$ is defined as

$$d_j = gcd\{n; p_{ii}^{(n)} > 0\}$$

The state j is said to be ***aperiodic*** if $d_j = 1$.
We say a Markov Chain $\mathcal{C}$ is aperiodic if all its states are aperiodic.

Definition 2.8. Let $i \in \mathbb{S}$ be a state of a Markov Chain, and

$$p_i^b := P(x \text{ returns to state i} \mid x_0 = i).$$

The state i is called ***recurrent*** if $p_i^b = 1$ (otherwise, we will call the state i transient).
The state i is called a ***positive-recurrent*** state if the expected state returning time t_i is finite.

$$E(t_i \mid x_0 = i) < \infty$$

2.2 Fundamental Theorem of Markov chains

Having introduced the definitions in the previous section, the Fundamental Theorem of Markov Chains can now be presented:

Theorem 2.9. (Fundamental Theorem of Markov Chains) *For any irreducible, aperiodic, positive-recurrent Markov Chain $\mathcal{C}$ there exists a unique stationary distribution $\{\pi_j\,,\, j \in \mathbb{Z}\}$.*

Proof. Considering any $m \in \mathbb{N}$,

$$\sum_{i=0}^{m} p_{ij}^{(m)} \leq \sum_{i=0}^{\infty} p_{ij}^{(m)} \leq 1$$

Taking the limit as $m \to \infty$,

$$\lim_{m\to\infty} \sum_{i=0}^{m} p_{ij}^{(m)} = \sum_{i=0}^{\infty} \pi_j \leq 1$$

From here we can state that

$$\sum_{i=0}^{M} \pi_j \leq 1 \,, \; \forall M$$

Using the Chapman-Kolmogorov equation,

$$p_{ij}^{(m+1)} = \sum_{i=0}^{\infty} p_{ik}^{(m)} p_{kj} \geq \sum_{i=0}^{M} p_{ik}^{(m)} p_{kj}$$

and taking the limit $m, M \to \infty$, we obtain

$$\pi_j \geq \sum_{k=0}^{\infty} \pi_k p_{kj} \,, \; \forall j \in \mathbb{Z}$$

Having reached this point, and taking into account that we want to prove that there is only one unique distribution $\{\pi_j \,, \, j \in \mathbb{Z}\}$, if we can demonstrate that this last inequality is in fact an equality for each $j \in \mathbb{Z}$ then the theorem will be proven.
Therefore, we assume that for some state j strict inequality holds and we try to reach a contradiction. Adding all the states,

$$\sum_{j=0}^{\infty} \pi_j > \sum_{j=0}^{\infty} \sum_{k=0}^{\infty} \pi_k p_{kj} = \sum_{j=0}^{\infty} p_{kj} \sum_{k=0}^{\infty} \pi_k$$

By definition,

$$\sum_{j=0}^{\infty} p_{kj} = 1,$$

therefore

$$\sum_{j=0}^{\infty} \pi_j > \sum_{k=0}^{\infty} \pi_k$$

which is clearly a contradiction.
Then, for each state j equality holds, and we obtain the following equation

$$\pi_j = \sum_{k=0}^{\infty} \pi_k p_{kj} \,, \; \forall j \in \mathbb{Z}$$

which defines a unique stationary distribution. □

Example 2.10. The aim of this example is to show a particular case of an irreducible, aperiodic, positive-recurrent Markov Chain converging to a unique stationary distribution. Consider a Markov chain $\mathcal{C}$, $\mathbb{S} = \{a, b, c\}$ its discrete state space, and the estimated initial probability distribution $p_0 = \{0.6, 0.2, 0.2\}$. Also consider the transition matrix of $\mathcal{C}$

$$P_{\mathcal{C}} = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \end{array} \begin{array}{c} \begin{array}{ccc} x_1 & x_2 & x_3 \end{array} \\ \begin{pmatrix} 0.6 & 0.3 & 0.1 \\ 0.4 & 0.5 & 0.1 \\ 0.3 & 0 & 0.7 \end{pmatrix} \end{array}$$

Using a Python code (see section A.2.2 in the Appendix), we calculate the vectors $p_i = (P(x_i = a)\ ,\ P(y_i = b)\ ,\ P(z_i = c))$, $\forall i \in \mathbb{N}$, until the difference between p_i and p_{i-1} is lower than 10^{-5}. At this point, we consider that the vector of probabilities has converged to its stationary distribution.
The result given by the program provides us with the following stationary distribution:

$$p_* = \{0.46875076169968116, 0.2812522850990188, 0.2499969532012999\}$$

Remark 2.11. In the above example, convergence has been reached in few steps because the fixed tolerance for convergence is 10^{-5}, which is quite a high value to consider if the probability vector has converged.

Chapter 3

MCMC methods

This chapter is an in-depth analysis of MCMC methods.
The preface of the chapter introduces the general structure of MCMC methods. Then, first section introduces a pseudo MCMC method; the Acceptance-Rejection sampling. From section two to five, each section analyzes a random walk MCMC method, explaining its procedure and demonstrating its convergence to the desired distribution. From section six to eight, three different non-random walk MCMC methods are analyzed; each one of these sections includes the discussion of the idea and detail of the steps for each method.

Preface

Markov Chain Monte Carlo methods, from now on MCMC methods, are a sample collecting methodology based on building a Markov chain generating the elements from a probability distribution.

The convergence speed of a method depends on its potential. For simple MCMC methods, known as random walk methods, it can take a large number of steps to attain convergence. More complex and sophisticated MCMC methods, known as non-random walk methods, need much fewer steps to achieve convergence, but at the same time their implementation is far more complicated.

Generally, MCMC methods follow the structure below:

1. Begin from an initial position $x^{(0)}$, which can be either a random or a selected value.
2. For every iteration i, suggest a new position $x^{(1)}$.
3. Accept or reject the new position.
4. Go back to step 2., until the wished number of samples n has been reached. At this point, return the values for $x^{(j)}, j \in \{0, 1, ..., n\}$.

The following sections, apart from explaining the methodology for each algorithm, also demonstrate that the algorithms converge to the desired distribution; in most cases, demonstrations use the theorem 2.9 (*Fundamental Theorem of Markov chains*) to prove convergence of the methods.

Remark 3.1. In chapter 2, the notation used for Markov chains is $\mathcal{C} = \{x_0, x_1, \ldots, x_n\}$. From now on, the notation used will be $\mathcal{C} = \{x^{(0)}, x^{(1)}, \ldots, x^{(n)}\}$ as we will use subindices for dimensions; i.e., we will denote a k-dimensional parameter λ as $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_k)$.

3.1 Acceptance-Rejection sampling

Acceptance-Rejection method (also known as A-R sampling) is a non MCMC method which represents classical simulation techniques. A-R sampling generates non-Markov samples, as the successive generated elements $x^{(i)}$ are independent from each other; recall that a sequence $\{x^{(0)}, x^{(1)}, \ldots, x^{(n)}\}$ is a Markov chain if the transition probabilities from one state to another depend on the last element.
By using the A-R method a distribution function for our random variable that tends to the real one can be estimated. The bigger the sampling is, the more accurate our function will be.

Steps in the algorithm

Let f be the probability function we want to simulate, and g the alternative density function. We assume that for all x and for $c \geq 0$, $f(x) \leq c * g(x)$, or equivalently,

$$sup_x \frac{f(x)}{g(x)} \leq c\,, \quad c \geq 0$$

The steps to generate the sampling are:

1. An initial value $x^{(0)}$ is set. It can be either selected or a random value.
2. For every iteration i, sample a candidate $x_* \sim g(\cdot)$.
3. Sample $u \sim \mathcal{U}_{[0,1]}$, where $\mathcal{U}_{[0,1]}$ is the uniform distribution between 0 and 1.
4. If
$$u \leq \frac{f(x_*)}{c \cdot g(x_*)},$$
then
$$x^{(i+1)} = x_*.$$
Otherwise, we reject the new sample x_* and go back to step 2.

Remark 3.2. Note that the method is well-defined as the quotient f over $c * g$ is between 0 and 1, therefore the following inequality is coherent

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

As f and g are density functions, $f(x), g(x) > 0$ for any x found running the algorithm. Moreover, $c > 0$, therefore

$$\frac{1}{c} > 0\,,\ \frac{f(x)}{g(x)} > 0\ \Rightarrow\ \frac{f(x)}{c \cdot g(x)} > 0,\ \forall x.$$

On the other hand,

$$sup_x \frac{f(x)}{g(x)} \leq c,\ c \geq 0\ \Rightarrow\ \frac{f(x)}{c * g(x)} \leq 1.$$

Therefore,

$$0 < \frac{f(x)}{c * g(x)} \leq 1$$

Proof of the convergence of the method

Consider $F(x)$ and $G(x)$ the cdf's (cumulative distribution functions) of the pdf's (probability density functions) f and g respectively:

$$F(x) = \int_{-\infty}^{x} f(t)dt \quad , \quad G(x) = \int_{-\infty}^{x} g(t)dt.$$

To demonstrate the convergence of the generated sample to the desired distribution we have to prove that F is the cumulative distribution function of x given that $u \leq \frac{f(x)}{c \cdot g(x)}$; i.e.,

$$P\Big(X \leq x \mid u \leq \frac{f(x)}{c \cdot g(x)}\Big) = F(x).$$

Proof. Consider the Bayes equation

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

with $A = \{X \leq x\}$ and $B = \{u \leq \frac{f(X)}{c \cdot g(X)}\}$.
Then,

$$P\Big(X \leq x \mid u \leq \frac{f(X)}{c \cdot g(X)}\Big) = \frac{P\Big(u \leq \frac{f(X)}{c \cdot g(X)} \mid X \leq x\Big) \cdot P(X \leq x)}{P\Big(u \leq \frac{f(X)}{c \cdot g(X)}\Big)}$$

Let's compute sepparately $P(B)$, $P(A)$ and $P(B \mid A)$.

- $P(A)$: By hypothesis,

$$P(A) = P(X \leq x) = G(x).$$

- $P(B)$:

$$P(B) = P\left(u \leq \frac{f(X)}{c \cdot g(X)}\right) = \int_{-\infty}^{\infty} \frac{f(x)}{c \cdot g(x)}\, g(x)\, dx = \frac{1}{c}\int_{-\infty}^{\infty} f(x)\, dx = \frac{1}{c}$$

- $P(B \mid A)$: Using the Bayes equation,

$$\begin{aligned}
P(B \mid A) = P\left(u \leq \frac{f(X)}{c \cdot g(X)} \mid X \leq x\right) &= \frac{P\left(u \leq \frac{f(X)}{c \cdot g(X)}\ ,\ X \leq x\right)}{G(x)} = \\
&= \int_{-\infty}^{x} \frac{P\left(u \leq \frac{f(X)}{c \cdot g(X)}\ ,\ X = t \leq x\right)}{G(x)}\, g(t)\, dt = \\
&= \frac{1}{G(x)} \int_{-\infty}^{x} \frac{f(t)}{c \cdot g(t)}\, g(t)\, dt = \frac{1}{c \cdot G(x)} \int_{-\infty}^{x} f(t) dt = \\
&= \frac{F(x)}{c \cdot G(x)}
\end{aligned}$$

Now, knowing $P(A)$, $P(B)$ and $P(B \mid A)$ we can calculate $P(A \mid B)$:

$$P(A \mid B) = \frac{P(B \mid A) P(A)}{P(B)} \Leftrightarrow$$

$$\Leftrightarrow P\left(X \leq x \mid u \leq \frac{f(X)}{c \cdot g(X)}\right) = \frac{P\left(u \leq \frac{f(X)}{c \cdot g(X)} \mid X \leq x\right) \cdot P(X \leq x)}{P\left(u \leq \frac{f(X)}{c \cdot g(X)}\right)} \Leftrightarrow$$

$$\Leftrightarrow P\left(X \leq x \mid u \leq \frac{f(X)}{c \cdot g(X)}\right) = \frac{\frac{F(x)}{c \cdot G(x)} \cdot G(x)}{\frac{1}{c}} = F(x)$$

Therefore,

$$P\left(X \leq x \mid u \leq \frac{f(x)}{c \cdot g(x)}\right) = F(x)$$

□

The following methods are known as random walk MCMC methods. These methods generate a random walk Markov chain; that means that when choosing the following element for the sample, the algoritihm does it in a random way (of course, following the probabilities assigned by the distribution). The random walk methods analyzed in this project are:

- Metropolis algorithm
- Metropolis-Hastings algorithm
- Gibbs sampling
- Single-variable slice sampling

3.2 Metropolis algorithm

Metropolis algorithm is one of the most popular and simple MCMC methods.
Is a particular case of Metropolis-Hastings algorithm, explained in the following section. In this method, the proposal density distribution h is considered to be symmetric, that means that, for any two elements x,y in the state space, the probability of jumping from x to y is the same than the probability of jumping from y to x,

$$h(x \mid y) = h(y \mid x)$$

Remark 3.3. We call h a density distribution but, in fact, it can be a function proportional to the density function. Sometimes it is a very difficult task to find the normalizing value, so this makes the algorithm very useful.

Steps in the algorithm

The first step is choosing a starting value $x^{(0)}$ following the proposal distribution $h(x)$. Then, the method employs the following scheme to generate the Markov chain:

1. Begin from a starting point $x^{(0)} \sim h(x)$
2. Having generated the first i elements, sample $x_* \sim h(x_* \mid x^{(i)})$
3. Sample $u \sim \mathcal{U}_{[0,1]}$
4. If
$$u < min\left\{ \frac{p(x_*)}{p(x^{(i)})}, 1 \right\}$$
then
$$x^{(i+1)} = x_*$$
Otherwise,
$$x^{(i+1)} = x^{(i)}$$
5. Iterate the process again from Step 2., until the number of samples generated is the desired.

Proof of the convergence of the method

The following lines prove the validity of the Metropolis algorithm for a Markov chain $\mathcal{C} = \{x^{(n)}\}_n$ with a discrete state space $\mathcal{S} = \{y_1, ..., y_m\}$.

Proof. (Consistence of the Metropolis algorithm) Considering the way the algorithm builds the random walk it is clearly a homogeneous Markov chain.
As we have an homogeneous Markov chain, it makes sense to define the transition probability for any two elements $y_j, y_k \in \mathcal{S}$ as

$$P_{j\to k}(y_j, y_k) := P(x^{(i+1)} = y_k \mid x^{(i)} = y_j), \quad \forall i \in \mathbb{N}$$

The transition probability function satisfies

I) $P_{j\to k}(y_j, y_k) \geq 0\,, \quad \forall y_j, y_k \in \mathcal{S}$

II) $\sum_{y_k \in \mathcal{S}} P_{j\to k}(y_j, y_k) = 1\,, \quad \forall y_j \in \mathcal{S}$

We now consider $M_T = (m_{ij})$ the transition matrix, where $m_{i_* j_*}$ is the probability of going from x_{i_*} to x_{j_*}, where $x_{i_*}, x_{j_*} \in \mathcal{S}$, and let $\omega_t = (\rho_1^{(t)}, ..., \rho_n^{(t)})$ be the probability vector at time t, where

$$\rho_k^{(t)} := P(x^{(t)} = y_k)$$

For ω_1, every element $\rho_k^{(1)}$, with $k \in \{1, ..., n\}$, satisfies

$$\begin{aligned}\rho_k^{(1)} := P(x^{(1)} = x_k) &= \sum_{j\in\{1,...,m\}} P(x^{(0)} = y_j)\; P(x^{(1)} = y_k | x^{(0)} = y_j) = \\ &= \sum_{j\in\{1,...,m\}} \rho_i^{(0)}\; P(x^{(1)} = y_k | x^{(0)} = x_j) = \\ &= \sum_{j\in\{1,...,m\}} \rho_i^{(0)}\; P_{j\to k}(y_j, y_k),\end{aligned}$$

which can also be written in matrix notation as

$$\omega_1 = \omega_0\; M_T.$$

Reasoning the same way,

$$\omega_2 = \omega_1\; M_T = \omega_0\; M_T^2$$

therefore, generally,

$$\omega_t = \omega_0\; M_T^t$$

Moreover, ω_t is an ergodic Markov chain. Therefore, using the Fundamental Theorem of Markov chains (Theorem 2.9 in this project), the behaviour of ω_t at the limit is independent of ω_0; that means, that there exists $t_* \in \mathbb{N}$ such that

$$\omega_{t_*+1} = \omega_{t_*} =: \omega_*,$$

and therefore $\omega_* = (\rho_1^*, ..., \rho_n^*)$ is the stationary distribution of the chain.
To corroborate that the Metropolis algorithm is well-defined it has to be demonstrated

that the Markov chain $\mathcal{C}$ obtained running the algorithm follows the correct stationary distribution, that means, the target posterior distribution.
Let's consider the stochastic process of running $\mathcal{C}$ backwards in time. This process is also a Markov chain, though not necessarily homogeneous. If it is homogeneous and the transition probabilities are the same that the ones of the original process, the Markov chain $\mathcal{C}$ satisfies the detailed balance equation (defined in A.6 in the Appendix)

$$\rho_k^* \; P_{j\to k}(y_k, y_j) = \rho_j^* \; P_{j\to j}(y_j, y_k) \quad , \quad \forall y_j, y_k \in \mathcal{S}$$

The following lemma will be key for the demonstration:

Lemma 3.4. *Let $\mathcal{D} = (d_1, ..., d_m)$ be a distribution satisfying the detailed balance equation*

$$d_k \; P_{k\to j}(y_k, y_j) = d_j \; P_{j\to k}(y_j, y_k) \quad , \quad \forall y_j, y_k \in \mathcal{S}$$

for an irreducible Markov chain $\mathcal{C}$ with state space $\mathcal{S} = (y_1, ..., y_m)$.
Thus, $\mathcal{C}$ is an ergodic and reversible Markov chain, and its stationary distribution is $\mathcal{D}$.

Proof. Lemma. By hypothesis, $\mathcal{D} = (d_1, ..., d_n)$ satisfies

$$d_k \; P_{k\to j}(y_k, y_j) = d_j \; P_{j\to k}(y_j, y_k) \quad , \quad \forall y_j, y_k \in \mathcal{S}.$$

Therefore, for all $k \in \{1, ..., m\}$, summing over $j \in \{1, ..., m\}$,

$$\sum_{j\in\{1,...,m\}} d_k \; P_{k\to j}(y_k, y_j) = \sum_{j\in\{1,...,m\}} d_j \; P_{j\to k}(y_j, y_k) \Leftrightarrow$$
$$\Leftrightarrow d_k = \sum_{j\in\{1,...,m\}} d_j \; P_{j\to k}(y_j, y_k),$$

which proves that $\mathcal{D}$ is the stationary distribution for $\mathcal{C}$. □

Considering a target distribution $\mathcal{P}_k$ on $\mathcal{S}$, the goal is to build a Markov chain with stationary distribution $\mathcal{D} = (d_1, ..., d_n)$ such that

$$d_k = \mathcal{P}_k \quad , \quad \forall k \in \{1, ..., m\}$$

Defining

$$A = \{\text{The transition move is accepted}\},$$
$$T(y_j, y_k) = \{\text{The transition for the chain induced by the proposal distribution}\},$$

note that $P_{j\to k}(y_j, y_k)$ can be written as

$$P_{j\to k}(y_j, y_k) = T(y_j, y_k)P(A) = T(y_j, y_k)min\left(1, \frac{y_k}{y_j}\right).$$

a) If $j \neq k$, the transition probabilities are as follows:

$$\begin{aligned} P_{k\to j}(y_k, y_j) &= P(x^{(t+1)} = y_k \mid x^{(i)} = y_j) = \\ &= P(x^{(i+1)} = y_k \ , \ A \mid x^{(i)} = y_j) + P(x^{(i+1)} = y_k \ , \ A^c \mid x^{(i)} = y_j) = \\ &= P(x^{(i+1)} = y_k \mid A \ , \ x^{(i)} = y_j) \ P(A \mid x^{(i)} = y_j) = \\ &= P(x^{(i+1)} = y_k \mid A \ , \ x^{(i)} = y_j) \ P(A) = \\ &= T(y_k, y_j) \ min\left(1, \frac{p_k}{p_j}\right). \end{aligned}$$

When $j \neq k$, we distinguish two cases; $p_j \leq p_k$ and $p_j > p_k$. In the following lines, we prove that for both cases the detailed balance equation is satisfied.

- $0 < p_j \leq p_k$: Note that

$$min\left(1, \frac{p_j}{p_k}\right) p_k = min(p_k, p_j) = p_j.$$

Then,

$$\begin{aligned} p_j \ P_{k\to j}(y_k, y_j) = p_j \ T(y_k, y_j) \ min\left(1, \frac{p_k}{p_j}\right) = p_j \ T(y_k, y_j) = p_j \ T(y_j, y_k) = \\ = T(y_j, y_k) \ min\left(1, \frac{p_j}{p_k}\right) \ p_k = p_k P_{j\to k}(y_j, y_k) \end{aligned}$$

- $0 < p_k < p_j$: Analogously to the case above,

$$p_j \ P_{k\to j}(y_k, y_j) = p_j \ P_{j\to k}(y_k, y_j).$$

b) If $j = k$, the detailed balance equation is trivially satisfied

$$p_j P_{j\to j} = p_j P_{j\to j}.$$

Using lemma 3.4, the demonstration is completed. □

3.3 Metropolis-Hastings algorithm

As mentioned in section 3.2, the Metropolis-Hastings (MH) algorithm is a generalization of the Metropolis algorithm; in MH algorithm, the distribution h is not required to be symmetric.

Remark 3.5. Trivially, as the Metropolis algorithm is a particular case of the MH algorithm, h can be a function proportional to the density function.

Steps in the algorithm

As in the Metropolis algorithm, the first step is to choose a starting value $x^{(0)}$ following the proposal distribution $h(x)$. The steps for the construction of a Markov chain are the following:

1. Begin from a starting point $x^{(0)} \sim h(x)$
2. Having generated the first i elements, sample $x_* \sim h(x_* \mid x^{(i)})$
3. Sample $u \sim \mathcal{U}_{[0,1]}$
4. If
$$u < min\left\{ \frac{p(x_*)\, h\big(x^{(i)} \mid x_*\big)}{p(x^{(i)})\, h\big(x_* \mid x^{(i)}\big)}, 1 \right\},$$
then
$$x^{(i+1)} = x_*.$$
Otherwise,
$$x^{(i+1)} = x^{(i)}.$$
5. Iterate the process again from Step 2., until the number of samples generated is the desired.

Proof of the convergence of the method

The following lines prove the validity of the Metropolis-Hastings algorithm for a Markov chain $\mathcal{C} = \{x^{(n)}\}_n$ with a discrete state space $\mathcal{S} = \{y_1, ..., y_m\}$.

Proof. (Consistence of the Metropolis-Hastings algorithm) Let $\mathcal{Q}$ be the proposed distribution, and $\mathcal{P}$ the desired distribution. The goal is to demonstrate that the states of the Markov chain generated by the Metropolis-Hastings algorithm satisfy the requirement. Let's also define

$$\tau(y_j, y_k) = min\left(1, \frac{\mathcal{P}(y_k)\ \mathcal{Q}(y_j \mid y_k)}{\mathcal{P}(y_j)\ \mathcal{Q}(y_k \mid y_j)}\right)$$

as the acceptance probability of jumping from the state y_j to the state y_k. Consider the transition probability from a state y_j to another state y_k

$$P(x^{(i+1)} = y_k \mid x^{(i)} = y_j) = \mathcal{Q}(y_k \mid y_j)\ \tau(y_j, y_k)$$

where $\mathcal{Q}(y_k \mid y_j)$ is the probability that y_k is generated after the state y_j, and $\tau(y_j, y_k)$ is the probability that y_k is accepted as the new state in the Markov chain after y_j.

As in the Metropolis algorithm, we will prove the validity of the algorithm by demonstrating that the detailed balance equation holds for any $y_j, y_k \in \mathcal{S}$:

$$\begin{aligned} P(x^{(i+1)} = y_k \mid x^{(i)} = y_j)\mathcal{P}(y_j) = P(x^{(i+1)} = y_j \mid x^{(i)} = y_k)\mathcal{P}(y_k) \Leftrightarrow \\ \Leftrightarrow \mathcal{Q}(y_k \mid y_j)\tau(y_j, y_k)\mathcal{P}(y_j) = \mathcal{Q}(y_j \mid y_k)\tau(y_k, y_j)\mathcal{P}(y_k) \end{aligned}$$

Proof will be done by distinguishing three complementary cases and demonstrating that the equality holds in each case:

a) $\mathcal{P}(y_k)\ \mathcal{Q}(y_j \mid y_k) = \mathcal{P}(y_j)\ \mathcal{Q}(y_k \mid y_j)$: Recall that τ has been defined as

$$\tau(y_j, y_k) = min\left(1, \frac{\mathcal{P}(y_k)\ \mathcal{Q}(y_j \mid y_k)}{\mathcal{P}(y_j)\ \mathcal{Q}(y_k \mid y_j)}\right).$$

Therefore, considering the hypothesis that $\mathcal{P}(y_k)\ \mathcal{Q}(y_j \mid y_k) = \mathcal{P}(y_j)\ \mathcal{Q}(y_k \mid y_j)$, then

$$\tau(y_j, y_k) = \tau(y_k, y_j) = 1.$$

Therefore,

$$P(x^{(i+1)} = y_k \mid x^{(i)} = y_j)\mathcal{P}(y_j) = \mathcal{Q}(y_k \mid y_j)\mathcal{P}(y_j)\tau(y_j, y_k) = \mathcal{Q}(y_k \mid y_j)\mathcal{P}(y_j) = P(y_j, y_k)$$

$$P(x^{(i+1)} = y_j \mid x^{(i)} = y_k)\mathcal{P}(y_k) = \mathcal{Q}(y_j \mid y_k)\mathcal{P}(y_k)\tau(y_k, y_j) = \mathcal{Q}(y_j \mid y_k)\mathcal{P}(y_k) = P(y_j, y_k)$$

which proves that, for this particular case,

$$P(x^{(i+1)} = y_k \mid x^{(i)} = y_j)\mathcal{P}(y_j) = P(x(i+1) = y_j \mid x^{(i)} = y_k)\mathcal{P}(y_k).$$

b) $\mathcal{P}(y_k)\ \mathcal{Q}(y_j \mid y_k) > \mathcal{P}(y_j)\ \mathcal{Q}(y_k \mid y_j)$: In this case,

$$\begin{aligned} \tau(y_j, y_k) &= min\left(1, \frac{\mathcal{P}(y_k)\ \mathcal{Q}(y_j \mid y_k)}{\mathcal{P}(y_j)\ \mathcal{Q}(y_k \mid y_j)}\right) = 1 \\ \tau(y_k, y_j) &= min\left(1, \frac{\mathcal{P}(y_j)\ \mathcal{Q}(y_k \mid y_j)}{\mathcal{P}(y_k)\ \mathcal{Q}(y_j \mid y_k)}\right) = \frac{\mathcal{P}(y_j)\ \mathcal{Q}(y_k \mid y_j)}{\mathcal{P}(y_k)\ \mathcal{Q}(y_j \mid y_k)} < 1 \end{aligned}$$

Then,

$$\begin{aligned} P(x^{(i+1)} = y_j \mid x^{(i)} = y_k)\mathcal{P}(y_k) &= \mathcal{Q}(y_j \mid y_k)\mathcal{P}(y_k)\tau(y_k, y_j) = \\ &= \mathcal{Q}(y_j \mid y_k)\mathcal{P}(y_k)\frac{\mathcal{P}(y_j)\ \mathcal{Q}(y_k \mid y_j)}{\mathcal{P}(y_k)\ \mathcal{Q}(y_j \mid y_k)} = \\ &= \mathcal{P}(y_j)\mathcal{Q}(y_k \mid y_j) = \mathcal{P}(y_j)\mathcal{Q}(y_k \mid y_j)\tau(y_j, y_k) = \\ &= P(x^{(i+1)} = y_k \mid x^{(t)} = y_j)\mathcal{P}(y_j). \end{aligned}$$

c) $\mathcal{P}(x_j)\ \mathcal{Q}(x_i \mid x_j) \leq \mathcal{P}(x_i)\ \mathcal{Q}(x_j \mid x_i)$: The detailed balance equation is proved proceeding analogously to case b).

At this point of the demonstration it has already been proved that the detailed balance equation holds for any y_j, y_k in the state space. The detailed balance equation will help proving that, if $x^{(i)}$ is from $\mathcal{P}(X)$, then the next element $x^{(i+1)}$ in the Markov chain generated by the Metropolis-Hastings algorithm is also from $\mathcal{P}(X)$. Writing the balance equation as

$$P\big(x^{(i+1)} \mid x^{(i)}\big)\ \mathcal{P}\big(x^{(i)}\big) = P\big(x^{(i)} \mid x^{(i+1)}\big)\ \mathcal{P}\big(x^{(i+1)}\big)$$

and integrating at both sides respect to $x^{(i)}$,

$$\begin{aligned}
&\int P(x^{(i+1)} \mid x^{(i)})\ \mathcal{P}(x^{(i)})\ dx^{(i)} = \int P(x^{(i)} \mid x^{(i+1)})\ \mathcal{P}(x^{(i+1)})\ dx^{(i)} \Leftrightarrow \\
\Leftrightarrow &\int P(x^{(i+1)} \mid x^{(i)})\ \mathcal{P}(x^{(i)})\ dx^{(i)} = \mathcal{P}(x^{(i+1)}) \int P(x^{(i)} \mid x^{(i+1)})\ dx^{(i)} \Leftrightarrow \\
\Leftrightarrow &\int P(x^{(i+1)} \mid x^{(i)})\ \mathcal{P}(x^{(i)})\ dx^{(i)} = \mathcal{P}(x^{(i+1)})
\end{aligned}$$

left-hand side of the equation gives the marginal distribution of $x^{(i+1)}$ under the assumption that $x^{(i)}$ is from $\mathcal{P}(X)$, and the right-hand side results to be $\mathcal{P}\big(x^{(i+1)}\big)$.
This shows that $x^{(i+1)}$ is from the distribution $\mathcal{P}$, what proves the consistence of the M-H algorithm. □

3.4 Gibbs sampling

Let $x = (x_1, x_2, \dots, x_m)$ be an m-dimensional random variable that satisfies

$$(x_1, x_2, \dots, x_m) \sim \mathcal{P}(x_1, x_2, \dots, x_m),$$

i.e., $x \sim \mathcal{P}(x)$.
The idea is to generate posterior samples using for each variable its conditional distribution given the current values of the remaining variables. Gibbs sampling, different from the other MCMC methods, samples one component x_i at a time from a marginal distribution $\mathcal{P}_i$, where $\mathcal{P} = (\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_m)$.

Remark 3.6. Note that $x = (x_1, x_2, \dots, x_m)$ has been defined as an m-dimensional random variable, but in fact these m components can be m different unidimensional random variables, or a set of multidimensional random variables of any dimensions.

Steps in the algorithm

The algorithm is applied as follows:

I) An initial value $x^{(0)} = (x_1^{(0)}, x_2^{(0)}, \dots, x_m^{(0)})$ is set.

II) Having generated the first i elements,

$$\begin{aligned} x_1^{(i+1)} &\sim \mathcal{P}_1\big(x_1 \mid x_2^{(i)}, x_3^{(i)}, ..., x_m^{(i)}\big), \\ x_2^{(i+1)} &\sim \mathcal{P}_2\big(x_2 \mid x_1^{(i+1)}, x_3^{(i)}, ..., x_m^{(i)}\big), \\ &\vdots \\ x_m^{(i+1)} &\sim \mathcal{P}_m\big(x_m \mid x_1^{(i+1)}, x_2^{(i+1)}, ..., x_{m-1}^{(i+1)}\big) \end{aligned}$$

where $\mathcal{P} = (\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_m)$ are the marginal distributions.

III) Go back to step 2., until convergence of $\big(x_1^{(i)}, x_2^{(i)}, ..., x_m^{(i)}\big)$.

Proof of the convergence of the method

To show that the Gibbs sampling works, we will split the demonstration in two parts:

I. Show that $\mathcal{P}$ is the stationary distribution for the Markov chain $\{x^{(n)}\}_n$ generated.

II. Show that the Markov chain $\{x^{(n)}\}_n$ generated by using Gibbs sampling is ergodic.

By showing that $\{x^{(n)}\}_n$ is ergodic the convergence is demonstrated, and the fact that $\mathcal{P}$ is the stationary distribution for the Markov chain proves that it converges to $\mathcal{P}$.

I. $\mathcal{P}$ is the stationary distribution for the Markov chain $\{x^{(n)}\}_n$ generated

Proof. The demonstration will be done for the case where x is a 2-dimensional parameter,

$$x = (x_1, x_2).$$

Demonstrating that for higher dimensions follow by induction.
Following the notation used in the lines above, $x = (x_1, x_2) \sim P(x_1, x_2)$. Let $x' = (x_1', x_2')$ be another point of the sample space. Then, the transition density from x to x' is

$$K(x, x') = \mathcal{P}(x_1' \mid x_2)\mathcal{P}(x_2' \mid x_1').$$

We want to prove that $\mathcal{P}$ is the stationary distribution for the Markov chain, that means, that every element of the Markov chain generated will have $\mathcal{P}$ as its distribution.
Therefore, it has to be demonstrated that the marginal distribution $g(x')$ for x' is $\mathcal{P}$; i.e.,

$$g(x') = \int_A \mathcal{P}(x', \cdot) d\cdot$$

Then,

$$\begin{aligned}
P(x' \in A) &= \int \mathbb{1}_A \mathcal{P}(x')dx' = \int \mathbb{1}_A K(x,x')\mathcal{P}(x)dx'dx = \\
&= \int \mathbb{1}_A \mathcal{P}_1(x_1' \mid x_2)\mathcal{P}_2(x_2' \mid x_1')\mathcal{P}(x_1,x_2)dx_1'dx_2'dx_1dx_2 = \\
&= \int \mathbb{1}_A \mathcal{P}_1(x_1' \mid x_2)\mathcal{P}_2(x_2' \mid x_1')\Big(\int \mathcal{P}(x_1,x_2)dx_1\Big)dx_1'dx_2'dx_2 = \\
&= \int \mathbb{1}_A \mathcal{P}_1(x_1' \mid x_2)\mathcal{P}_2(x_2' \mid x_1')\mathcal{P}_2(x_2)dx_1'dx_2'dx_2 = \\
&= \int \mathbb{1}_A \Big(\int \mathcal{P}_1(x_1' \mid x_2)\mathcal{P}_2(x_2)dx_2\Big)\mathcal{P}_2(x_2' \mid x_1')dx_1'dx_2' = \\
&= \int \mathbb{1}_A \mathcal{P}_1(x_1')\mathcal{P}_2(x_2' \mid x_1')dx_1'dx_2' = \\
&= \int \mathbb{1}_A \mathcal{P}_(x_1',x_2')dx_1'dx_2' = \\
&= \int \mathbb{1}_A \mathcal{P}_1(x')dx'
\end{aligned}$$

□

II. Ergodicity of the Markov chain $\{x^{(n)}\}_n$

After demonstrating that $\mathcal{P}$ is the stationary distribution for the Markov chain $\{x^{(n)}\}_n$, we only need to prove the ergodicity of the chain.

Proof. Let $\mathcal{X}$ be the sample space of the parameter x, $\mathcal{A}$ the sigma-algebra on Θ, and K the transition kernel (defined in A.9 in the Appendix) for $\mathcal{X}$ and $\mathcal{A}$.
For every point $x \in \mathcal{X}$, and for every subset $A \in \mathcal{A}$ with

$$\int \mathbb{1}_A \mathcal{P}(x)dx > 0,$$

then $K(x,A) > 0$.
Therefore, the Gibbs sampler is ergodic. □

3.5 Single-variable slice sampling

Slice sampling method is based on the fact that generating a sample from a density distribution $f(x)$ is the same as sampling, uniformly, from the points underneath the curve of $f(x)$.

Remark 3.7. As in other MCMC methods seen before, f can be an unnormalized density function.

Remark 3.8. Slice sampling comprise a various range of methods, some of them very sophisticated and very difficult to implement. In this project, we will focus on the simple slice sampling method.

Steps in the algorithm

Keeping the same notation as above, consider f as the distribution of a random variable x. The Slice sampling method generates the Markov chain $\mathcal{C} = \left\{x^{(n)}\right\}_n$ following the steps detailed below:

1. Begin from a starting point $x^{(0)}$ such that $f(x^{(0)}) > 0$.
2. Having generated the first i elements, sample

$$y_* \sim \mathcal{U}_{[0,f(x^{(i)})]}$$

3. Sample a point (x, y_*) uniformly such that it is within the distribution curve $f(x)$
4. Go back to step 2., taking x as the new value to sample $y \sim \mathcal{U}_{[0,f(x)]}$
5. Iteration of steps two to four is done until convergence.

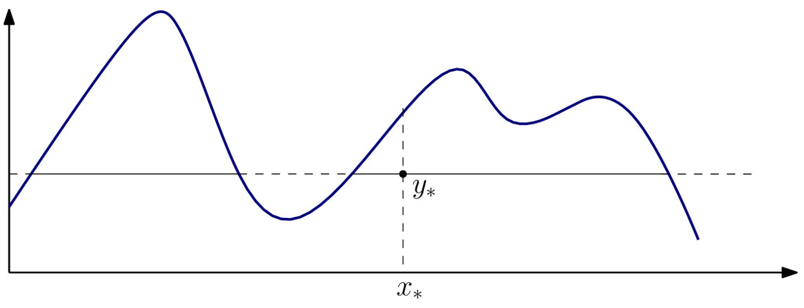

Figure 3.1: Graphical interpretation of the method.

Summing up the method, given a element $x^{(i)}$ in the Markov chain, the following element $x^{(i+1)}$ is generated in three steps: First, sample a value y_* uniformly from $[0, f(x^{(i)})]$. Second, defining the slice $S_{y_*} := \{x \ ; \ y_* < f(x)\}$ containing all the values x which its probability is higher than y_* (i.e., the ones which are within the curve of f). Finally, build an interval I_{y_*}, and sample the value $x^{(i+1)}$ uniformly from the interval. Figure 3.1 represents graphically the process of finding the new element for the sampling.
For the basic slice sampling method, there are two different ways of building the interval I_{y_*}:

I) Stepping out and shrinkage: An interval of length l is placed around $x^{(i)}$ (not necessarily centered in $x^{(i)}$). If both ends of the interval are not in S_{y_*}, we take this interval as I_{y_*}. Otherwise, we lengthen the interval adding another interval of size l (randomly at the left or at the right). We iterate this process until both sides of the intervals are not in S_y.

   Then, we sample a point randomly from I_{y_*}. If the value randomly sampled x_* is in S_{y_*}, then $x^{(i+1)} = x_*$. If $x_* \notin S_y$, it is rejected and another element is generated from

I_{y_*} until finding one in S_{y_*}. When selecting a value which is not in S_{y_*}, this point is taken as the new margin of the interval, to shrink it and avoid ineffective sampling.

II) Doubling: As in the stepping out and shrinkage method, an interval of length l is placed around $x^{(i)}$. If both ends of the interval are not in S_{y_*}, we take this interval as I_{y_*}. Otherwise, the interval is doubled as many times as needed until both ends of the interval are not in S_{y_*}. To select the value for $x^{(i+1)}$ we sample randomly from I_{y_*} until an element in S_{y_*} is found.

Proof of the convergence of the method

To proof the convergence of the method, it must be shown that for each step in the iteration, the desired distribution remains the same and, moreover, that the Markov chain generated by the method is ergodic.
Proving these two statements the convergence of the Markov chain to the desired distribution will be demonstrated.
Considering $\mathcal{S}$ the state space of X, i.e., such that $f(x) > 0$ for all $x \in \mathcal{S}$, it is trivial to prove that the Markov chain generated by the algorithm is ergodic.
Demonstration will focus on showing the invariance of the desired distribution f.

Proof. We assume that the initial state $x^{(0)}$ is distributed following $f(x)$. The following step in the algorithm is sampling a value y_* uniformly between 0 and $f(x^{(0)})$.
Therefore, the joint distribution for $x^{(0)}$ and y is

$$p(x^{(0)}, y_*) = \begin{cases} \frac{1}{\int f(x)dx} & , \ 0 < y_* < f(x^{(0)}) \\ 0 & , \ otherwise \end{cases} .$$

If the steps that generate $x^{(1)}$ from $x^{(0)}$ leave the joint distribution invariant, that means that $x^{(1)}$ will have the same distribution as $x^{(0)}$, as

$$p(x^{(1)}, y_*) = \begin{cases} \frac{1}{\int f(x)dx} & , \ 0 < y_* < f(x^{(1)}) \\ 0 & , \ otherwise \end{cases} .$$

and, computing the marginal distributions of $x^{(0)}$ and $x^{(1)}$ from the joint distributions $p(x^{(0)}, y_*)$ and $p(x^{(1)}, y_*)$ respectively:

$$p(x^{(0)}) = \int_0^{f(x)} \frac{1}{\int f(x)dx} dy = \frac{f(x^{(0)})}{\int f(x)dx}$$
$$p(x^{(1)}) = \int_0^{f(x)} \frac{1}{\int f(x)dx} dy = \frac{f(x^{(1)})}{\int f(x)dx}$$

We only have to demonstrate, then, that the steps to select $x^{(1)}$ to follow $x^{(0)}$ leave the joint distribution invariant.
This invariance of the joint distribution will be proved showing that the detailed balance equation is satisfied:

$$P(x^{(1)} \mid x^{(0)})\mathcal{P}(x^{(0)}) = P(x^{(0)} \mid x^{(1)})\mathcal{P}(x^{(1)}).$$

As the way of selecting y_* in the algorithm is based on an uniform distribution, $\mathcal{P}(x^{(0)}) = \mathcal{P}(x^{(1)})$, and consequently,

$$P(x_1 \mid x^{(0)}) = P(x^{(0)} \mid x^{(1)}).$$

But, in fact, when $x^{(1)}$ is selected as the new state in the Markov chain after $x^{(0)}$, some random acts take place before the selection (the value y_* chosen, the interval chosen,..). Let us denote by r all these random acts, and by π a one-to-one mapping such that $\pi(r)$ has Jacobian equal to 1. We will prove the equation

$$P(x_1 \, , \, r \mid x^{(0)}) = P(x^{(0)} \, , \, \pi(r) \mid x^{(1)}),$$

which is clearly more restrictive the detailed balance equation.
Independently from the procedure employed to build the interval I_{y_*} (stepping out or doubling), a value U is generated, which will be the center of the interval I_{y_*}. The mapping π is defined satisfying $\pi(U_0) = U_1$, where

$$U_1 = k - [k] \, , \quad \text{where } k = U_0 + \frac{x^{(1)} - x^{(0)}}{w}.$$

The mapping π, thus, as we have defined it, maps values that produce the same alignment of the initial interval.

I) If the stepping out method is employed,

- A Jacobian value J is also generated in the process. π maps J_0 with J_1, where

$$J_1 = J_0 + \left(\frac{1}{w}x^{(1)} - U_1\right) - \left(\frac{1}{w}x^{(0)} - U_0\right)$$

  Defining I_0 and I_1 as the intervals from where $x^{(0)}$ and $x^{(1)}$ have been respectively sampled, note that $z := \left(\frac{1}{w}x^{(1)} - U_1\right) - \left(\frac{1}{w}x^{(0)} - U_0\right)$ shows the number of steps of size w from the left end of I_0 to the left end of I_1. Adjusting by z the Jacobian J_0, we make sure that if the interval I_0 arrives to its maximum size, the interval I_1 will be identical.

II) If the double procedure method is employed,

- π maps the sequence of random decisions related to the expansion of the interval to the sequence that would make I_1 be exactly the same interval than I_0 when $x^{(1)} \in I_0$, and remain invariant otherwise.
- π maps the sequence of rejected points used to shrink I_0 to the same sequence when $x^{(1)}$ is the starting point.

Now that π is defined, we proceed to demonstrate the equality

$$P(x_1 \, , \, r \mid x^{(0)}) = P(x^{(0)} \, , \, \pi(r) \mid x^{(1)}).$$

When there is no possibility of going from $x^{(0)}$ to $x^{(1)}$ or viceversa the equality is trivially satisfied (both sides are zero).
Otherwise, the values U_0 and U_1 (calculated the same way for stepping out or doubling procedure) will have the same distribution as they are sampled from a uniform distribution.

I) For the stepping out method, J_0 and J_1 are also generated from a uniform distribution with possible values in $\{0, 1, ..., m-1\}$, so they always do have the same distribution.

II) For the doubling procedure, all the sequences of random decisions with the same length have the same probability.
Let's demonstrate that r and $\pi(r)$ have the same length.

- If the sequence from $x^{(1)}$ is shorter than the sequence from $x^{(0)}$, then $x^{(1)}$ is not a possible state immediately after $x^{(0)}$ due to the way that doubling procedure follows to generate new values.
- If the sequence from $x^{(0)}$ is shorter than the sequence from $x^{(1)}$, then $x^{(1)}$ is not a possible state immediately after $x^{(0)}$, because in that case $x^{(1)} \notin I_0$.

Thus, in the case where the length of r and $\pi(r)$ are not the same, then both sides of the equation
$$P(x_1\ ,\ r \mid x^{(0)}) = P(x^{(0)}\ ,\ \pi(r) \mid x^{(1)})$$
are again zero.

Finally, we will prove that for both stepping out and doubling method the intervals found are the same as long as π maps the random sequences for $x^{(0)}$ and $x^{(1)}$ and the transition from $x^{(0)}$ to $x^{(1)}$ is possible; i.e., $P(x^{(0)} \to x^{(1)}) > 0$.

I) For the stepping out method, we have just proved the relations between U_0 and U_1, and between J_0 and J_1. That means that the intervals will have the same maximum length. Furthermore, as if the interval I_0 contains $x^{(1)}$ then I_0 and I_1 expand the same way, we can conclude that I_0 and I_1 have to be necessarily the same interval.

II) For the doubling procedure, the definition of π implies
$$x^{(1)} \in I_0 \Rightarrow I_1 = I_0$$
In case the process finishes before $x^{(1)} \in I_0$, $x^{(1)}$ would be discarded as the state right after $x^{(0)}$.

Also, the elements q such that
$$P(\text{choosing } I \text{ in } q) = P(\text{choosing } I \text{ in } x)$$
will be the same for $x^{(0)}$ and $x^{(1)}$.

When we apply the shrinking procedure, the intermediate values can be thought as the rejected elements which make the interval shorter.
Mapped by π, the sequences of rejected elements are the same for $x^{(0)}$ and $x^{(1)}$. Hence, the probability density for the first elements in both sequences (for $x^{(0)}$ and $x^{(1)}$) are also

the same. As the first elements have the same probability distribution, the decision of shrinking or not the interval will be identical in both cases. Following this reasoning, every step will be the same in both sequences and therefore probability densities for every intermediate value will be identic for both $x^{(0)}$ and $x^{(1)}$.

After all the cases have been analyzed, the equation

$$P(x_1 \, , \, r \mid x^{(0)}) = P(x^{(0)} \, , \, \pi(r) \mid x^{(1)})$$

has been proved, and therefore the invariance of the desired distribution when going from a state $x^{(0)}$ to $x^{(1)}$.
Homogeneity of the Markov chain generated by the slice sampling generalises the demonstration for any point in the sampling. □

After introducing the random walk MCMC methods, we will study some algorithms with a higher degree of complexity. The algorithms below try to optimize the process by being selective when choosing the candidates for the next element in the Markov chain. In contrast to the random walk MCMC methods, these methods are known as non-random walk MCMC methods. The main ones will be covered in this project:

- Hamiltonian Monte Carlo
- No U-Turn Sampler
- Langevin algorithm

3.6 Hamiltonian Monte Carlo

Hamiltonian Monte-Carlo, also known as HMC or Hybrid Monte Carlo, stands from the other methods presented above because its convergence speed is higher. This faster convergence is achieved by simulating a physical system with Hamiltonian dynamics, which optimizes the process and avoids random walk.

Idea of the method

Consider $x = (x_1, x_2, \ldots, x_n)$ an n-dimensional parameter. We want to draw a Monte-Carlo sampling from the density distribution

$$\pi(x) \propto exp\{-P(x)\},$$

where $P(x)$ is a potential energy function. Let $\vec{v} = (v_1, ..., v_n)$ be a momentum vector. We define $K(v)$ as

$$K(v) = \frac{1}{2}\sum_{i=1}^{n}\frac{v_i^2}{w_i},$$

where w_i is the weight of the i-th component.
Then, the total energy is defined as

$$H(x,\vec{v}) = P(x) + K(\vec{v})$$

Note that if we sample the pair of elements $(x,\vec{v})$ from the joint distribution

$$\pi(x,\vec{v}) \propto exp\{-H(x,\vec{v})\}$$

the marginal distribution for x is $\pi(x)$:

Proof.

$$\begin{aligned}
\pi(x,\vec{v}) \propto exp\{-H(x,\vec{v})\} &\Leftrightarrow \pi(x,\vec{v}) \propto exp\{-P(x) - K(\vec{v})\} \\
&\Leftrightarrow \pi(x,\vec{v}) = c \cdot exp\{-P(x) - K(\vec{v})\} \\
&\Leftrightarrow \int_{\mathbb{D}_{\vec{v}}} \pi(x,\vec{v}) d\vec{v} = c \cdot exp\{-P(x)\} \int_{\mathbb{D}_{\vec{v}}} exp\{-K(\vec{v})\} d\vec{v} \\
&\Leftrightarrow \pi(x) = c \cdot exp\{-P(x)\} \cdot c_{\int} \\
&\Leftrightarrow \pi(x) \propto exp\{-P(x)\}
\end{aligned}$$

□

Remark 3.9. In the demonstration above, $c_{\int}$ is the value which results from resolving the integral

$$\int_{\mathbb{D}_{\vec{v}}} exp\{-K(\vec{v})\} d\vec{v}.$$

It is a defined integral over a closed domain ($\mathbb{D}_{\vec{v}}$ is the domain of the momentum vector), therefore the integral results in a value $c_{\int} \in \mathbb{R}$.

Moreover, if a system conserves energy, its evolution dynamics are described by the differential equations

$$\frac{\partial H}{\partial x} = -\dot{\vec{v}} \quad , \quad \frac{\partial H}{\partial \vec{v}} = \dot{x}$$

We will not prove the following properties, but the fact that Hamiltonian dynamics are time-reversible and they preserve volume and energy, leaves $\pi(x,\vec{v})$ invariant; i.e., if $(x^{(i)}, \vec{v}^{(i)}) \sim \pi(x,\vec{v})$ then $(x^{(i+1)}, \vec{v}^{(i+1)}) \sim \pi(x,\vec{v})$. The invariance of $\pi(x,\vec{v})$ and the ergodicity of the Markov chain generated by the algorithm demonstrate the validity and convergence of the method. The ergodic property will not be proved as it would mean going beyond the purpose of this project.

Steps in the algorithm

The Hamiltonian Monte Carlo algorithm follows the next steps to generate the sample:

1. Generate an initial vector $\vec{v}^{(0)}$ from the marginal distribution

$$\pi(\vec{v}^{(0)}) \propto -exp\{K(\vec{v}^{(0)})\}$$

2. Having generated the first i elements, use the leapfrog algorithm to generate a candidate $(x_*, \vec{v}_*) \sim pi(x, \vec{v})$

3. Generate $u \sim \mathcal{U}_{[0,1]}$.

4. If

$$u \leq min\big(1, exp\{-H(x_*, \vec{v}_*) + H(x^{(i)}, \vec{v})\}\big),$$

then

$$(x^{(i+1)}, \vec{v}^{(i+1)}) = (x_*, -\vec{v}_*).$$

Otherwise,

$$(x^{(i+1)}, \vec{v}^{(i+1)}) = (x^{(i)}, \vec{v}^{(i+1)}).$$

Remark 3.10. By using multi-point method or parallel tempering, parameters are tuned in order to optimize the efficiency of the method. These two methods are discussed in [16] in the Bibliography. On the other hand, for complete detail of the demonstration of the convergence of HMC method, see [17] in the Bibliography.

3.7 No U-turn Sampler

The No U-Turn Sampler (also known as NUTS) is an optimization of the Hamiltonian Monte Carlo method. As we have seen before, HMC method is a non-random walk which converges faster than the random walk methods. Despite that, when the number of elements sampled m is not large enough, the algorithm still shows a random walk behaviour.
The NUTS does not need the specification of m to run the algorithm. In fact, this is a big benefit because it is not easy to select the optimal number of steps; it has to be a number of steps big enough to get the method to converge, but not too large because, once convergence is reached, the algorithm generates samples which repeat sequences obtained before; which makes it inefficient.

Idea of the method

As mentioned before, the NUTS relies on detecting when the optimal number of steps has been reached and therefore when to stop the sampling process.
Consider $x^{(0)}$ and x_* the starting value and the current value for the parameter to model respectively. The inctuitive idea is that when $\mid x_* - x^{(0)} \mid$ does no longer increase, then convergence has been reached and therefore no more sampling is needed.

Remark 3.11. $x = (x_1, x_2 \ldots, x_n$ is an n-dimensional parameter, with $n \geq 1$.

As we will see when defining the conditions to run the algorithm, a must condition for a value x_* to be a candidate for the next element in the sampling is that the detailed balance equation has to be satisfied.

Steps in the algorithm

Before setting the steps of the NUTS algorithm, we have to define some sets and assume some conditions in order to ensure the proper development of the algorithm. Let $\vec{r}$ be the momentum of x, $\mathcal{L}$ the logarithm of the density of x, and let u be a slice variable (defined in A.8 in the Appendix). As in HMC, the joint probability $p(x, \vec{r})$ is

$$p(x, \vec{r}) \propto exp\{\mathcal{L}(x) - \frac{1}{2}\vec{r}\vec{r}\},$$

and the joint probability for $x, \vec{r}, u$ is

$$p(x, \vec{r}, u) \propto \mathbb{1}_{[0, exp\{\mathcal{L}(x) - \frac{1}{2}\vec{r}\vec{r}\}]}.$$

Observe that integrating over u the joint probability $p(x, \vec{r}, u)$ we obtain $p(x, \vec{r})$.
To simplify the notation, let us define as B the set containing all the candidate states, and $C \subseteq B$ the set containing all the candidate states which satisfy the detailed balance equation.

As mentioned when explaining the idea of the method, NUTS generates a bunch of candidates $(x, \vec{r})$ and then chooses the best one. The subset B contains all these candidates. So, given certain values for $x, \vec{r}, u$ and a maximum step size (which we will call ϵ) there is a random process that generates B. As C has absolute dependence of B, it also depends on this random process.

Therefore we are interested in the following probability:

$$p(B, C \mid x, \vec{r}, u, \epsilon)$$

In order to give a mathematical sense to this probability, some supositions are necessary:

S1. Any step in the process to obtain a candidate $(x', \vec{r}') \in C$ from $(x, \vec{r}) \in C$ has to satisfy that the determinant of its Jacobian is equal to 1.

S2. $p((x, \vec{r}) \in C \mid x, \vec{r}, u, \epsilon) = 1$

S3. $p(u \leq exp\{\mathcal{L}(x) - \frac{1}{2}\vec{r}\vec{r}\} \mid (x', \vec{r}') \in C) = 1$

S4. $(x', \vec{r}') \in C\ ,\ (x, \vec{r}) \in C \ \Rightarrow\ p(B, C \mid x, \vec{r}, u, \epsilon) = p(B, C \mid x', \vec{r}', u, \epsilon)$

Once considered the prior assumptions, the steps for building the sampling are the following:

1. Sample $\vec{r} \sim N(\vec{0}_n, Id_n)$, where $\vec{0}_n$ is the n-dimensional zero, and Id_k the identity in the space of square matrices of dimension n.

2. Sample $u \sim \mathcal{U}_{[0, exp\{\mathcal{L}(x^{(k)}) - \frac{1}{2}\vec{r}\vec{r}\}]}$.

3. Sample B and C from the joint probability $p(B, C \mid x^{(k)}, \vec{r}, u, \epsilon)$.

4. Sample $(x^{(k+1)}, \vec{r}) \sim \mathcal{T}(x^{(k)}, \vec{r}, C)$.
   $\mathcal{T}$ is the kernel distribution defined such as it leaves the uniform distribution over C invariant:
   $$\sum_{(x,\vec{r}) \in C} \mathcal{T}(x', \vec{r}' \mid x, \vec{r}, u) = \mathbb{1}_{(x', \vec{r}') \in C}$$

Once the new element in the chain $x^{(k+1)}$ is obtained, we test if $\mid x^{(k+1)} - x^{(k)} \mid$ increases its value or not. If it does increase its value significantly, then we iterate the process again. If not, the method has already converged and then we stop sampling; we have reached the optimal number of elements in the sampled chain.

Remark 3.12. The stopping criterion is much more complex than as it has been explained in the lines above. Going in-depth into the in and outs of this process would mean both extending too much in the explanation and going beyond the complexity expected for this project.

To sum up, the No U-Turn Sampler follows a process of generating samples that makes the probability
$$p(x, \vec{r}, u, B, C \mid \epsilon)$$
remain invariant; that means, that given the maximum step size ϵ the algorithm is well-defined at any time in the sampled chain because the distribution of θ, which is the parameter we want to model, remains also invariant. In fact, considering that we have defined $\mathcal{L}$ as the logarithm of the density of x, we have that
$$p(x) = exp\{\mathcal{L}(x)\}$$

Remark 3.13. The equality above holds module constant, given that $\mathcal{L}(x)$ is not normalised.

The formal demonstration of the convergence of the method will not be given because it goes beyond the complexity of this project. For further detail, see [18] in the bibliography.

3.8 Langevin algorithm

Langevin algorithm, as a non-random walk MCMC method, aims to optimize the sampling procedure by tuning the parameters that affect the algorithm. It is an upgrade of the Metropolis-Hastings algorithm; Langevin algorithm follows the same sampling rules than MH algorithm but with a faster convergence.

Idea of the method

Consider $x = (x_1, x_2, \dots, x_n)$ an n-dimensional parameter, π a differenciable density on the n-dimensional space where x lives, and D the drift function (defined in A.10 in the Appendix) of the algorithm, defined as

$$D(x) = \frac{x}{max(x, \mid \bigtriangledown \mathcal{L}(x)) \mid)} \bigtriangledown \mathcal{L}(x),$$

where $\delta \in \mathbb{R}$, $\bigtriangledown$ is the gradient and $\mathcal{L}$ is the logarithm of the density function π.

Remark 3.14. The case with $\delta = 0$, and therefore $D = 0$, is the random walk Metropolis-Hastings algorithm.

The two parameters that determine the convergence speed of the algorithm are a $n \times n$ positive definite matrix Λ, and a positive scale parameter σ.
Let $q(x, x')$ be a density function of a Normal distribution with expected value

$$\mu_q = x + \frac{\sigma^2}{2} \Lambda D(x),$$

and covariance

$$\sigma_q = \sigma^2 \Lambda.$$

The idea of the method is that given $x^{(i)}$ the i-th element of the generated Markov chain, an element x_* is proposed such that

$$x_* \sim N(\mu_q, \sigma_q).$$

As in Metropolis-Hastings algorithm, the proposal x_* is accepted with probability $p_A(x^{(i)}, x_*$, where

$$p_A(x^{(i)}, x_*) = min\left(1, \frac{\pi(x_*)q(x_*, x^{(i)})}{\pi(x^{(i)})q(x^{(i)}, x_*)}\right)$$

and therefore rejected with probability $1 - p_A(x^{(i)}, x_*)$. The transition kernel $\mathcal{T}$ of the algorithm is

$$\mathcal{T}(x, x') = \int_C p_A(x, x')q(x, x')dx' + \mathbb{1} \int (1 - p_A(x, x'))q(x, x')dx',$$

where $C \in \mathcal{B}^k$ is a set in the Borel subsets.

Remark 3.15. Note that the density function q depends on Λ and σ. Langevin algorithm finds the optimal values for Λ and σ are found, while when applying Metropolis-Hastings algorithm, both values have to be chosen, which is a difficult task.

The Langevin algorithm generates a well-defined Markov chain as it leaves the density π invariant for every element $x^{(i)}$, $\forall i \in \{0, 1, 2, ...\}$.

Steps in the algorithm

Consider $\epsilon_1, \epsilon_2, \alpha_1 \in \mathbb{R}^+$ such that $0 < \epsilon_1 < \alpha_1$, $I_\sigma = [\epsilon_1, \alpha_1]$ an interval, and $B_r(\vec{0}_n)$ the ball of radius r and center the n-dimensional zero.
Consider also M_Γ the set defined as

$$M_\Gamma = \left\{\Gamma \in \mathcal{M}_{n\times n} \; ; \; \Gamma \text{ semidefinite positive }, \mid \Gamma \mid_F \leq \alpha_1 \right\},$$

where $| \cdot |_F$ is the Frobenius norm (defined in A.11 in the Appendix).
Then, we define the space

$$\Theta = B_r(\vec{0}_n) \times M_\Gamma \times I_\sigma,$$

where the process of optimizing the parameters takes place. This process of optimizing is done by creating an element $(\mu_i, \Gamma_i, \sigma_i) \in \Theta$ for every iteration i in the process. This element is called the adaptation process of the method.

Generation of the adaptation process

Generating the values μ_i, Γ_i, and σ_i requires three independent endomorphisms f_{μ_*}, f_{Γ_*} and f_{σ_*} defined as follows:

- For a n-dimensional parameter μ, we define f_{μ_*} as

$$f_{\mu_*} = \begin{cases} \mu & , \mid \mu \mid \leq \alpha_1 \\ \frac{\alpha_1}{|\mu|}\mu & , \mid \mu \mid > \alpha_1 \end{cases},$$

  i.e., $f_{\mu_*}(\mu)$ is the closest point to μ in $B_{\alpha_1}(\vec{0}_n)$.
  Therefore, $f_{\mu_*}(\mu)$ satisfies

$$\mid \mu' - f_{\mu_*}(\mu) \mid \leq \mid \mu' - \mu \mid,$$

  where μ' is a n-dimensional value such that $\mid \mu' \mid \in B_{\alpha_1}(\vec{0}_n)$.

- For a semidefinite positive matrix Γ, we define f_{Γ_*} as

$$f_{\Gamma_*} = \begin{cases} \Gamma & , \mid \Gamma \mid \leq \alpha_1 \\ \frac{\alpha_1}{|\Gamma|}\Gamma & , \mid \Gamma \mid > \alpha_1 \end{cases}.$$

  As in the case above, $f_{\Gamma_*}(\Gamma)$ is the closest point to Γ in M_Γ, and therefore satisfies

$$\mid \Gamma' - f_{\Gamma_*}(\Gamma) \mid \leq \mid \Gamma' - \Gamma \mid,$$

  where $\Gamma' \in M_\Gamma$.

- For $\sigma \in \mathbb{R}$, we define f_{σ_*} as

$$f_{\sigma_*} = \begin{cases} \sigma & , \sigma \in I_\sigma := [\epsilon_1, \alpha_1] \\ \epsilon_1 & , \sigma < \epsilon_1 \\ \alpha_1 & , \sigma > \alpha_1 \end{cases}$$

Therefore, $f_{\sigma_*}(\sigma)$ is the closest point to σ in I_σ, and satisfies

$$| \sigma' - f_{\sigma_*}(\sigma) | \leq | \sigma' - \sigma |,$$

where $\sigma' \in I_\sigma$.

The algorithm uses these three functions to perform as follows:

1. Begin from a starting point $x^{(0)}$, and a point $(\mu_0, \Gamma_0, \sigma_0) \in \Theta \equiv B_r(\vec{0}_n) \times M_\Gamma \times I_\sigma$.
2. Having generated the first i elements, set
$$\Lambda_i = \Gamma_i + \epsilon_2 I_n.$$
3. Sample x_* the proposed value for the following element $x^{(i+1)}$ and u_i,
$$\begin{aligned} x_* &\sim N(x^{(i)} + \frac{\sigma_i^2}{2}\Lambda_i D(x^{(i)}, \sigma_i^2\Lambda_i) \\ u_i &\sim \mathcal{U}_{[0,1]} \end{aligned}$$
4. Then, we define the $(i+1)$-th element of the Markov chain as
$$x^{(i+1)} = \begin{cases} x_* & , \ u \leq p_A(x^{(i)}, x_*) \\ x^{(i)} & , \ u > p_A(x^{(i)}, x_*) \end{cases}.$$
5. Finally, new values $(\mu_{i+1}, \Gamma_{i+1}, \sigma_{i+1})$ are calculated as
$$\begin{aligned} \mu_{i+1} &= f_{\mu_*}(\mu_i + \tau_i(x^{(i+1)} - \mu_i)) \\ \Gamma_{i+1} &= f_{\Gamma_*}(\Gamma_i + \tau_i(x^{(i+1)-\mu_i})(x^{(i+1)} - \mu_i)^t - \Gamma_i) \\ \sigma_{i+1} &= f_{\sigma_*}(\sigma_i + \tau_i(p_A(x^{(i)}, x^{(i+1)}) - \tilde{\tau}) \end{aligned}$$
where $\{\tau_i\}_i$ is a sequence of real positive values, and $\tilde{\tau}$ the estimated optimal acceptance rate for the algorithm.

Remark 3.16. The acceptance rate τ is defined as

$$\tau = \int \pi(dx) \int p_A(x, x') q(x, x') dx'.$$

The calculation of the optimal acceptance rate is a long process which goes beyond the purpose of this project.

The proof of the convergence of the Langevin algorithm to the desired distribution is done by showing that the generated Markov chain is ergodic. We will not give details of the demonstration as it goes beyond the complexity of this project. For further information of the demonstration and the calculation of the optimal acceptance rate, see [19] in the Bibliography.

Chapter 4

Probabilistic Programming and Bayesian inference

The first section of the chapter will provide an explanation of Bayesian inference. The second section explains how MCMC methods bring Probabilistic Programming to the next level, and at the same time, how Probabilistic Programming exploits all possibilities of MCMC methods. Finally, in the third section some key aspects for the optimization of the Probabilistic Programming models are discussed.

4.1 Bayesian inference

Bayesian inference is a term tied to the probabilistic field. It consists on updating the probabilities assigned to an event after observing new evidence. This procedure makes sense when probability is interpreted as a measure of belief of an event ocurring; and that's exactly how Bayesians understand the notion of probability.
Opposite to the Bayesian perspective, classical (or Frequentist) point of view understands probability as the long-run frequency of events. Even though both perspectives share some similarities, there are some nuances between them:

- The biggest difference is that Bayesian inference preserves uncertainty in its results by returning the distribution. This will be shown and clarified further in this chapter.
- Considering probability as the long-run frequency of an events, as from the Frequentist point of view, lacks of sense when an event does not have long-term frequency of occurrence. However, Bayesian perspective of understanding the probability as believability of success still makes sense when events are punctual.
- Understanding probability as a measure of belief opens the chance of having different beliefs, which means assigning different probabilities to the same event. Clearly, if two subjects have different information (or different opinions) for a same event, they will assign different probabilities of success to this event.

Prior and posterior probabilities

Consider an event A. The belief of success of the event A is called the prior probability. We will denote prior probability as $P(A)$. Now assume we have some evidence X. The updated belief is denoted by $P(A \mid X)$, understood as the probability of success of the event A having observed X, and it is called the posterior probability.
The Bayesian philosophy consists basically in the following two steps:

1. Build a prior belief $P(A)$ using the information available.

   Usually, this prior probability is a known distribution.

2. Observe evidence X. Once observed, update the prior belief $P(A)$ to the posterior belief $P(A \mid X)$. The way of updating our belief is by using the Bayes Theorem:

$$P(A \mid X) = \frac{P(X \mid A)P(A)}{P(X)}$$

Remark 4.1. The more pieces of evidence are observed, the more the prior belief will be blurred in benefit of the observations.

Bayesian inference's due respect to the Law of Large Numbers

The Law of Large numbers is present in any procedure involving convergence. The idea of the Law of Large numbers is that the mean of a sample of N observations of a certain random variable will converge to its expected value when $N \to \infty$.
This is a very powerful and useful statement for modeling purposes. But there is a must condition; the Law of Large Numbers is applicable only when $N \to \infty$.
And Bayesian inference is absolutely conscious about this sacred rule. By returning the probability distribution, Bayesian inference reflects perfectly the shape of the distribution, and therefore the uncertainty involved. When the posterior has a short amount of data, Bayesian inference shows higher uncertainty in the estimation. The fact that Bayesian inference returns the graphic of the distribution is one of the main advantages of its main advantages over Frequentist statistics, because looking at the shape of the distribution many information can be extracted.
Figure 4.1 is a simple demonstration of how Bayesian inference reflects uncertainty. The graphic in the left shows a model with a very low uncertainty (clearly, x_* is the most probable value), while the model in the right has a very large uncertainty. For both models, the most probable value is x_*, but with different degrees of certainty.

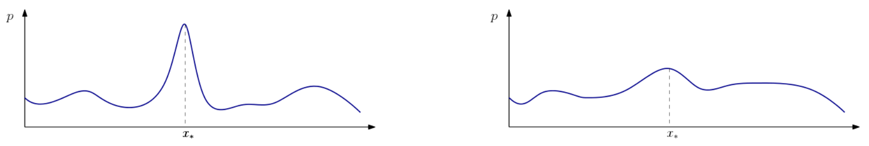

Figure 4.1: Comparison of different uncertainty levels.

To close this section, a simple example will illustrate the importance of Law of Large Numbers.

Example 4.2. As we have been talking about Monte Carlo during the whole project, using a gambling example is adequate.
Consider that we want to calculate the probability of obtaining a 13 in a non-tampered casino roulette; i.e., where all the numbers have the same probability.
It is widely known that a casino roulette has 37 numbers, from 0 to 36, therefore the probability for every number in the roulette is 1 over 37.

$$P(X = k) = \frac{1}{37}, \quad k \in \{0, 1, ..., 36\}$$

Using a Python code, 10000 roulette games have been simulated (section A.2.3 in the Appendix includes all the Python code employed for the example). Figure 4.2 below shows the results of the first 200 observations of the simulation, where ones mean a 13 in the roulette, and zeros any other number from 0 to 36 excluding 13. Only 200 observations are displayed in the graphic to make it clear; trying to plot all the 10000 observations resulted in an illegible plot.

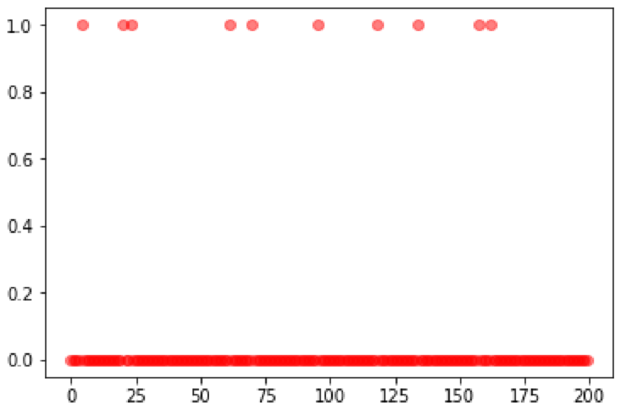

Figure 4.2: Results of the 200 first observations of the sampling.

We can see that the number 13 has been the winning number 10 times in these 200 simulations. Then, for a naive observer who has seen this simulation, the probability of getting a 13 in the roulette is

$$p = \frac{10}{200} = 0.05$$

when in reality it is $\frac{1}{37} \approx 0.027027027$.
Denoting $A = \{\text{Probability of a 13 in the roulette}\}$, the blue line on Figure 4.3 shows the probability of the event A for a naive observer; i.e., the probability calculated as number of successes over total number of observations:

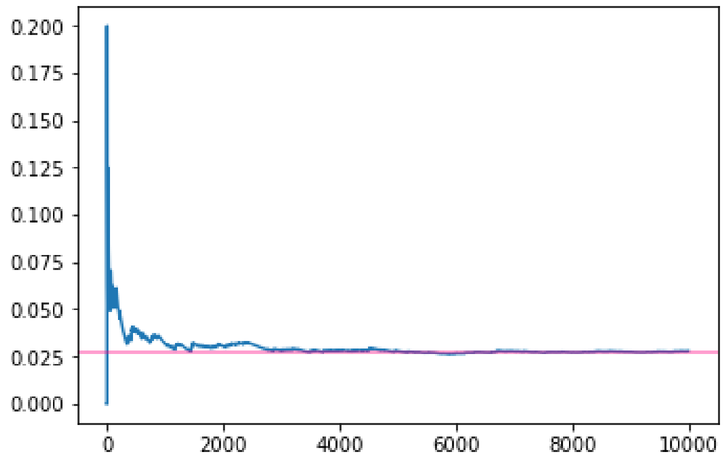

Figure 4.3: Theoretical and empirical probabilities for obtaining a 13 in a non-tampered roulette.

The red line is the theoretical probability of A. As we can see, it is not until $N = 5000$ approximately that the blue line begins to converge in a stable way to the red line.

Remark 4.3. In the example 4.2, note that when N is a low value, the accuracy of the models is very low.
Therefore, the evidence observed does not have much reliability when the number of observations is not large enough. For this reason, when the number of samples is small the prior probability is slightly modified. As N gets larger, the observed data changes the model to a greater extent.

4.2 Synergy between Probabilistic Programming and MCMC

After discussing Bayesian inference purposes and procedures, this section links Bayesian inference with the Monte Carlo Markov chain methods. MCMC methods are employed in Probabilistic Programming to estimate the parameters of the model.

Parameter estimation

When modeling the behaviour of a certain event, the main difficulty falls on parameter estimation. Once decided the key parameters of our model, the next step is determining the distribution that each of them may follow.
Consider Y a random variable, and $\alpha_1, \alpha_2, ..., \alpha_n$ the parameters involved in its distribution. Some of this parameters α_i might be determined with a certain value, but some other might be unknown parameters.
Thus, thinking in a Bayesian way, we can estimate the parameters α_i using Bayesian inference. That means that, at the same time, these α_i's can depend on some other

parameters which model their behaviour:

$$\alpha_i \sim D_i(\beta_{i1}, ..., \beta_{im}),$$

where D_i is a probability distribution with parameters β_{ij}, which are called *hyper-parameters*. Hyper-parameters are those parameters which have impact on other parameters.

The following list shows the questions we have to ask ourselves when facing Bayesian modeling:

I) Are we modeling the behaviour of a discrete, continuous or mixed random variable?

II) Which distribution follows the random variable and which parameters α_i are involved?

III) Can we make an accurate guess of the values for the parameters α_i or we need to find a distribution for them?

IV) If we need to find out the distribution for our parameters α_i, which distribution do these parameters follow and which parameters β_{ij} are involved?

V) Can we make a guess of the values β_{ij}? The process of finding hyper-parameters can be repeated as much as needed, i.e., these β_{ij} can depend at the same time on some other parameters γ_{ijk}, but is recommendable to stop iterating once we have gone too far in the hyper-parameter depth.

Once all the parameters are defined, our model is created.

Impact of MCMC

Probabilistic Programming is such a powerful tool on account of MCMC methods. Consider we want to model an event with k parameters. As we have seen before in this section, setting a prior distribution means modeling each parameter with a certain behaviour; this behaviour can be as naive as we want, but we have to assign each parameter a certain distribution.
Then, we can think about a $(k+1)$-dimensional surface where the first k dimensions are the k parameters, and the $(k+1)^{th}$ dimension is the probability assigned to every k-dimensional point.
We will illustrate this with a short example to make this notion clear before advancing on the explanation.

Example 4.4. Let's think about a simple example. Assume our model depends on two parameters α_1 and α_2. To make it even more simple, consider both parameters follow the Normal distributions below:

$$\alpha_1 \sim N(0,1),$$
$$\alpha_2 \sim N(2,1).$$

Then, we can build a surface

$$\begin{aligned} \mathcal{S}\colon \mathbb{R}^2 &\to \mathbb{R}^3 \\ (\alpha_1, \alpha_2) &\mapsto (\alpha_1, \alpha_2, P(\alpha_1, \alpha_2)) \end{aligned}$$

which shows the probability of every point in $\mathbb{D}^2 := \mathbb{D}_1 \times \mathbb{D}_2$, where $\mathbb{D}_i$ is the domain of the parameter α_i.

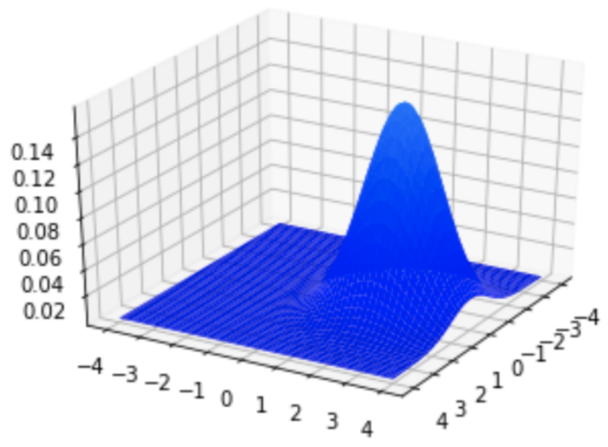

Figure 4.4: Surface representing the probabilities for each point in $\mathbb{R}^2$.

The vertical axis in Figure 4.4 (section A.2.4 in the Appendix includes the Python code employed to generate the plot) shows the probability. Clearly, the probability of $(\alpha_1 = 0, \alpha_2 = 2)$ is much higher than the probability of, for example, $(\alpha_1 = -3, \alpha_2 = 0)$.

Returning to the explanation, once we have the prior distribution, we sample data by useing an MCMC method. This will change the shape of the surface by giving a higher probability to some points and a lower probability to some others. The surface obtained after considering the data sampled is the shape of our posterior probability.
This way of considering Bayesian inference remarks that even when getting two identical sampled observations, if we apply them to two different prior beliefs, the resultant posterior beliefs will be different.
Using MCMC methods and Bayesian inference can seem a high-complexity implementation procedure, but the fact that it returns the shape of the probability distribution means being one step ahead other possible ways of presenting the results of the model:

- Only knowing the k-dimensional points with a higher probability is a much more simple option which would give information about the most probable values for the k parameters, but would not give any information about the shape of the surface, and therefore information about all the other points would be non-existent.
- Modeling the values of the k parameters with a formula would mean not losing any information, but it would involve creating a function $f : \mathbb{D}^k \longrightarrow \mathbb{R}$ which can be very complex to deal with and to build ($\mathbb{D}^k = \mathbb{D}_1 \times \mathbb{D}_2 \times \ldots \times \mathbb{D}_k$ is the product of the domains for each parameter).

As it has been mentioned in the preface of Chapter 3, MCMC methods converge to a distribution of possible points. Thus, when modeling a parameter by using MCMC, we are not only interested in the most probable value for the parameter but also in the uncertainty of the prediction and the other values probabilities. This is the main reason why there is a such strong link between MCMC methods and Probabilistic Programming, and at the same time the differentiating attribute that the first bring to the latter.

4.3 Model optimization

We have seen in Chapter 3 that all MCMC methods are well-defined and therefore all of them converge to the desired distribution. As stated in the Law of Large Numbers, convergence is demonstrated when $N \to \infty$.
But, in practice, we can not sample data with infinite observations. The faster our method converges, the shorter it will take to find an accurate solution to our problem. This section will discuss some practices which can optimize our models and increase their accuracy.

Burn-in period

Let x_0 be the initial point for starting our Markov chain.
The burn-in period is the set of the first m sampled observations $(x_0, x_1, ..., x_{m-1})$ which are removed from the sampling used for the model. Then, the sampled data used for building posterior probabilities begins in the m^{th} element of the Markov chain.
Using a burn-in period in our sampled data eliminates the first observations which, if the initial point is not very good, can distort our model.

Remark 4.5. Instead of sampling 10000 observations, it is recommended to sample for example 12000 observations and use the first 2000 observations as the burn-in period. The accuracy of the model will be higher if we use the sample with the burn-in period.

Good initial values

Another way of speeding up the convergence of the model is by choosing good initial values. What do we mean by good initial values? If we choose a starting point with a very low probability in our prior, the model will need a very large number of samples to converge. So, if we know the behaviour of the random variables we are modeling, we can choose a starting value that helps our model to converge faster.

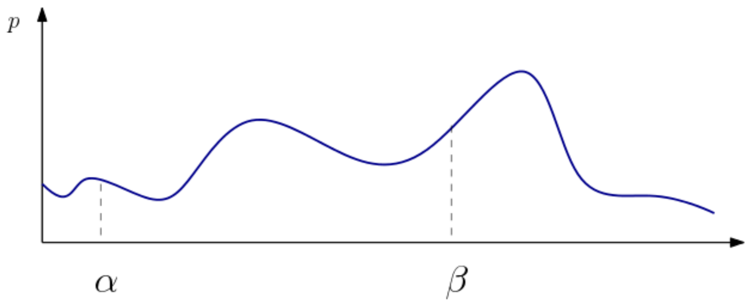

Figure 4.5: Choice of initial values.

In Figure 4.5 above, α is an example of a bad initial value, while β would be a good initial value, as it probability is high enough, and it will take short to get to the values with higher probability.

In fact, choosing a good initial value is equivalent to using a burn-in period. When we use a burn-in period what we are doing is to turn down the first observations and begin using data when it is more representative. Hence, if the choice of the starting point is appropriate is not necessary to consider a burn-in period.

Remark 4.6. It is known as MAP (maximum a posteriori) the point with a higher probability in the domain of our space of parameters. Beginning our sampling from the MAP is the ideal situation, since it is the "best" point from where we can begin our sampling.
If we were only interested in the most likely value for our parameters, approximating them by the MAP would be enough. But recall that with Bayesian inference we do obtain information of the expected value for our parameters but also of its uncertainty; so only considering the MAP would mean losing some useful information of the fitting capacity of our model.

Use confidence intervals and eliminate outliers from our data

Sampling is a process that collects data from a set of possible values (some of them are more probable and some are less probable). That means that we can get some outliers that can distort our data, especially if our sampling does not have a large amount of data. For this reason, it is recommendable to use confidence intervals in our sampled data; the most common is 95% confidence interval. By doing this, we improve the accuracy and uncertainty of the model.

Loss functions

Loss functions measure how good an estimation is by comparing a parameter with its estimation; $L(\theta, \hat{\theta})$.
There are some established loss functions, for example the absolute-loss function

$$L_{|\cdot|}(\theta, \hat{\theta}) = \mid \theta - \hat{\theta} \mid .$$

But, in some specific cases, we may want to create our own loss function. The following example is a particular case.

Example 4.7. Consider a mobile app that estimates the time of arrival from one place to another taking into account traffic, accidents, works on the road and any other possible relevant variables. It is much worse to fall short in the estimation than to do a cautious estimation. Therefore, programmers will that take into account and, when programming the computing algorithms to minimize the loss function, they will apply different criteria for the estimations which go beyond the real time than for the ones which estimate a shorter time. The loss function of the estimation could be, for example,

$$L_\tau = \begin{cases} \mid \tau - \hat{\tau} \mid & , \ \hat{\tau} <= \tau \\ 3 \mid \tau - \hat{\tau} \mid & , \ \hat{\tau} > \tau \end{cases} .$$

Loss functions are very useful for valuating estimations, but only when the parameter we want to estimate is unknown (if we do know the parameter, then there is no point in estimating it).
The way of computing the loss in the estimation is by using the posterior distribution of the parameter. Consider the parameter θ and some evidence X, from which the posterior for θ can be build. Then, the expected loss of the estimation $\hat{\theta}$ to estimate θ is

$$E_\theta(L(\theta, \hat{\theta}))$$

Now, we apply the Law of Large Numbers; considering N large enough, we can compute this expected value as

$$E_\theta(L(\theta, \hat{\theta})) \approx \frac{1}{N} \sum_1^N L(\theta_i, \hat{\theta})$$

Since the purpose is to minimize our loss function, we will try to find the estimator $\hat{\theta}_*$ such that

$$L(\theta, \hat{\theta}_*) = min_{\hat{\theta}} L(\theta, \hat{\theta})$$

Objective vs subjective priors

Objective priors are those prior probabilities that assign equal probability for every value in the domain of the parameter. That means that when setting the prior we show no preference for any value; every parameter in the model follows a uniform distribution. On the contrary, if we give different probabilities to the elements in the domain of the parameters, we are setting a subjective prior. Assigning any distribution different to the uniform as the prior probability is a subjective prior.
Figure 4.6 shows a very simple example of a objective and a posterior prior (see section A.2.5 in the Appendix for detail of the piece of Python code employed for generating both graphs).

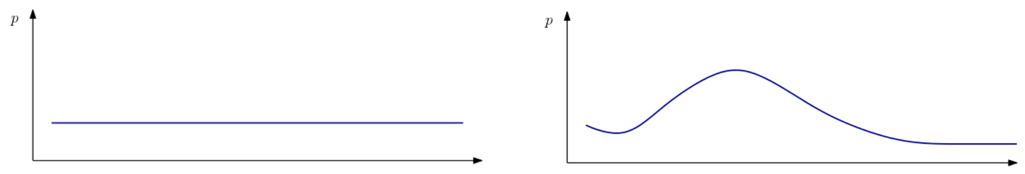

Figure 4.6: Objective and subjective prior distributions.

Therefore, choosing between using an objective or a subjective prior is a trade-off between not trying to biase our model and using some information to help our model converge faster and be more reliable.
Taking the option of an objective prior gives all the responsability of the posterior to the sampled data. This can be a good choice when we are not sure about the behaviour of the parameters. Setting a wrong prior is a critical mistake that can drive us to inaccurate results, so in those cases where we are not sure whether to choose a prior or another, an objective prior is a wise alternative.
On the other hand, when we choose a subjective prior we are giving our model some previous knowledge which, if precise, will help our model converge faster. Not only will imply a more accurate inference, since MCMC will move faster to the values with higher probabilities without scattering to little probable values, but will also restrict the domain of the parameter, and hence the model will reflect uncertainty better.

Chapter 5

Application of Probabilistic Programming

Once having analyzed Probabilistic Programming from a theoretical perspective, we will use it to study the performance of a professional basketball player, Roy Hibbert.
First, the case will be introduced, and the objectives and hypothesis will be set up. Following that, the modeling procedure will be detailed and studied step by step.

Statistics in sports, and especially in basketball, have an important role. Looking at the statistics of a player, we can guess what kind of player he is, if he performed well or not, and the fields where he can improve. The improvement of statistical models has also been implemented in the sports field, driving to more exhaustive statistical analyses which lead to more accurate and relevant results. Here we show an example of analyzing the performance of a player through his statistics during his professional career.

Roy Hibbert...Indiana man?

Roy Hibbert is a Jamaican-American professional NBA player. He was born in Queens, New York, in December 11^{th}, 1986. He plays as a center, he's 2.18m tall and weighs 122kg. He was selected in the 17th position of the 2008 NBA Draft by the Toronto Raptors. The Raptors traded his rights to the Indiana Pacers before even playing, where he made his NBA debut in the 2008-2009 season.
He played for Indiana Pacers from 2008-09 season to 2014-2015 season, where he performed at a very high level being considered one of the best centers in the NBA, even recording two appearances in the All-Star Game.
After seven seasons in Indiana, he was traded to the Los Angeles Lakers, who expected a lot from Roy Hibbert. His performance in L.A. was very poor, even playing the same minutes than in Indiana. After his first season as a Laker (the 2015-16 season), he was traded to Charlotte Hornets, where after his irregular performance, as well as having some troubles with teammates, was traded again to the Denver Nuggets in February 2017. In Denver, his contribution has been almost nonexistent.

Objectives of our model and hypothesis

Roy Hibbert's career has a clear downward trend. We will use Probabilistic Programming to test if the drop in his performance took place when he was traded to Los Angeles, or if it was previous or posterior to that.
Our hypothesis is, in fact, that the beginning of his lower performance is strictly related to the moment when he was traded to Los Angeles.
He had a very important role in Indiana Pacers, a team who has a very European basketball style and where centers have high importance. Roy Hibbert, as a center, was a key player in the team and he felt very comfortable in Indiana. Then, when traded to the Lakers, Roy found himself in a very different situation. He landed in a team where there was one player, Kobe Bryant, who the team played for, and where centers have a very different role. The structure of the Lakers roster in the 2015-16 season required a very physical center, who could run the court and be energetic all the minutes on the court. Roy Hibbert, however, is not that type of center. He feels comfortable with slow pace and he is a real threat when playing very controlled basketball.
He arrived to Los Angeles being a high reputation player, and when he saw that Kobe Bryant was the only star in the team, he began to feel uncomfortable. Once the season finished, he requested the trade to another team, but he still did not find his place neither in Charlotte nor in Denver.

Our model will look for the moment in Roy Hibbert's career when his performance noticed a change of pattern. Therefore, we will use the model to either accept or reject our hypothesis. If the model detects a decrease on Hibbert's performance in the moment he left Indiana, then our hypothesis will be accepted. Otherwise, we will reject it.
With the purpose of adding strength to our model, we will consider two variables to evaluate Roy Hibbert's performance; points and rebounds. The same procedure will be employed for these two variables.

Application of the model

The explanation of the procedure followed to build our model is structured in the following parts:

I) Initial data

II) Random variables of the model and selection of its priors

III) Results analysis and hypothesis evaluation

Remark 5.1. All the code employed to carry out the modeling process for this application case is in the section A.2.6 in the Appendix.

I) Initial data

When facing a modeling problem it is essential to make sure that our data is reliable. Therefore, it is fundamental to carry out a proper selection of the data taking into account that there can be some outliers that can distort our model.
In this particular case, we want to analyze the points and rebounds of Roy Hibbert in every game since season. Since we don't want any data that can drive us to confusing results, we have chosen only the games where he played at least one minute. It may seem obvious, but all those games where Roy Hibbert played less than one minute or did not play have been eliminating from our data.
We have also performed another filter to our data. We have eliminated the data from the first season of Roy Hibbert in the NBA, where he only played . Normally, rookies (names with which are known the players of first year in the NBA) need a period of time to get used to the level of exigence of the NBA season and also they have to make a space in the team. For this reasons, players usually play less minutes in their first year. As we don't want the rookie year of Roy Hibbert to distort our results, we have eliminated it from the data.
Once deleting these observations (both all the games of his first season and those games where he played less than one minute), we still have 587 observations; i.e., 587 games where Roy Hibbert played at least one minute.

Remark 5.2. Data has been obtained from the NBA stats section in the official website of ESPN [22].

Figures 5.1 and 5.2 below show the points and rebounds of Roy Hibbert in each game of his career. Taking a first look at these graphics, we can observe that both points and rebounds are considerably lower in his last games compared to the first half of his career until now. The graphic in blue corresponds to points, and the red one to rebounds.

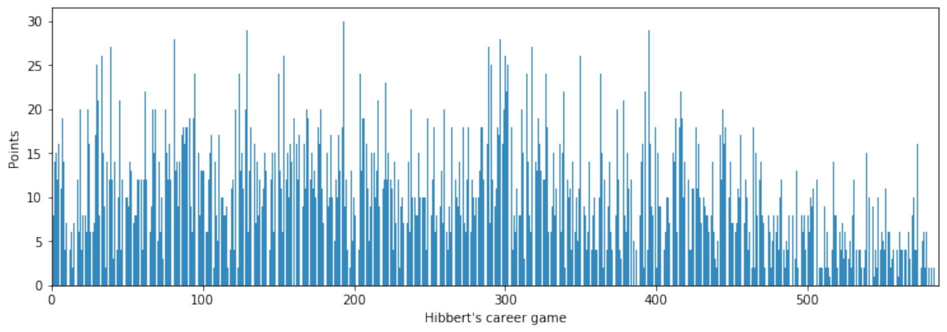

Figure 5.1: Points scored by Roy Hibbert in each game.

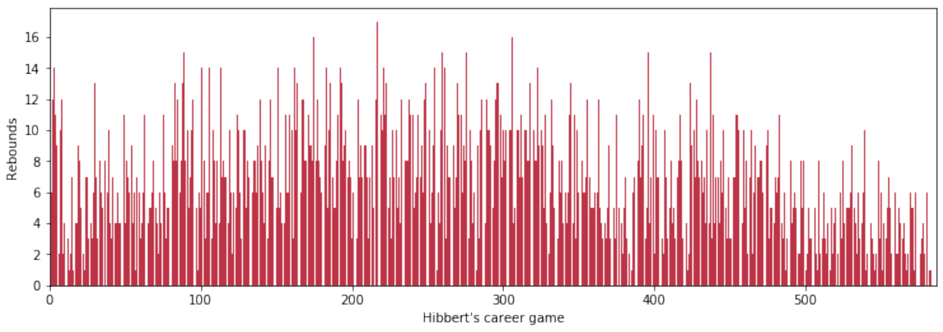

Figure 5.2: Rebounds grabbed by Roy Hibbert in each game.

II) Random variables of the model and selection of its priors

Once our data has been cleansed, a prior distribution for each random variable has to be chosen as a first approximation to our model. Therefore, first of all we have to know which are the random variables in our model.

As our model analyzes the behaviour of both points and rebounds, we will consider sepparately the random variables of our model needed to study Roy Hibbert points and rebounds behaviour. In our hypothesis, we assume there is a moment in Roy Hibbert's career where his performance declined. Therefore, we will need a random variable G_P to find in which game of his career the change of behaviour of Roy Hibbert points takes place. Then, we will have two different distributions for Roy Hibbert points, one before the game G_P and one after, determined by some random variables. The same reasoning applies to study the behaviour of Roy Hibbert rebounds, where we will need a random variable G_R plus the ones to build the distributions before and after G_R.

The priors will be known probability distributions which can be a good initial fit to our data. To find these distributions, we have to analyze our data. Clearly, for both points and rebounds, we are looking for discrete distributions, as they always take integer (and positive) values. If we take a quick look to the two graphics in part I), we can see that extreme observations are much less frequent than the ones closer to the mean (even we do not know the exact value of the mean), therefore we are looking for a bell-shaped distribution.
Putting together all these considerations, we consider a Poisson distribution to be an appropriate fitting for both points and rebounds. Besides, several studies have demonstrated that scoring in basketball fits a Poisson distribution. Figure 5.3 shows the Poisson probability function for different values of λ.

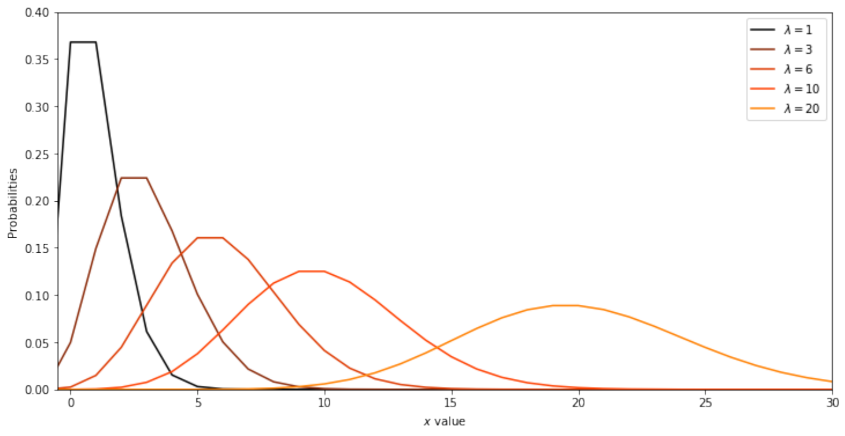

Figure 5.3: Poisson distributions modifying the value of λ.

Remark 5.3. In fact, as the Poisson is a discrete probability distribution, it only takes values with probability higher than 0 for integer values, but drawing it as a continuous variable gives a more visual idea of how the distribution changes when varying the parameter λ.

Then, for every $i \in \{1, 2, ..., 587\}$,

$$\begin{aligned} Points_i &\sim Poisson(\lambda_P) \\ Rebounds_i &\sim Poisson(\lambda_R) \end{aligned}$$

where $\lambda_P, \lambda_R \in \mathbb{R}^+$ are the parameters of the Poisson distribution for points and rebounds respectively.

As we need two different distributions to study the behaviour of points, one before G_P and another one after G_P, we will consider that, for any i lower than G_P, the observations follow a Poisson distribution with parameter λ_{P1}, and for i greater than or equal to G_P, the observations follow a Poisson distribution with parameter λ_{P2}. The same applies for modeling rebounds.

$$\begin{aligned} Pts_1 &\sim Poisson(\lambda_{P1}), & Rebs_1 &\sim Poisson(\lambda_{R1}), \\ Pts_2 &\sim Poisson(\lambda_{P2}), & Rebs_2 &\sim Poisson(\lambda_{R2}) \end{aligned}$$

Therefore, our model has the following random variables: λ_{P1}, λ_{P2}, λ_{R1}, λ_{R2}, G_P, G_R.

We assign to G_P and G_R uniform prior distributions to avoid adding subjectivity to our model, even though we suspect when the change in his performance may take place.

We will consider that the parameters λ_{P1}, λ_{P2}, λ_{R1}, λ_{R2} follow an exponential distribution, as all of these parameters are continuous random variables which only take positive values. Again, to avoid distorting our model, we will give the same exponential parameter β_P for λ_{P1} and λ_{P2}, and the same exponential parameter β_R for λ_{R1} and λ_{R2}; i.e.,

$$\lambda_{P1} \sim Exp(\beta_P), \qquad Rebs_1 \sim Poisson(\beta_R),$$
$$\lambda_{P2} \sim Exp(\beta_P), \qquad Rebs_2 \sim Poisson(\beta_R).$$

The parameter of an exponential distribution is the inverse of its mean. Therefore, to give an appropriate value to the parameters β_P, β_R, and denoting ppg and rpg as the points per game and rebounds per game respectively,

$$\beta_P = \frac{1}{ppg}\,,\ \beta_R = \frac{1}{rpg}$$

By way of summary, the random variables of the model are the following:

$$\begin{aligned} \lambda_{P1} &\sim Exp\left(\frac{1}{ppg}\right), & \lambda_{R1} &\sim Exp\left(\frac{1}{rpg}\right), \\ \lambda_{P2} &\sim Exp\left(\frac{1}{ppg}\right), & \lambda_{R2} &\sim Exp\left(\frac{1}{rpg}\right), \\ G_P &\sim \mathcal{U}_{[1,587]}, & G_R &\sim \mathcal{U}_{[1,587]} \end{aligned}$$

Remark 5.4. This model considers that there is just one change of behaviour in the data. More sophisticated MCMC methods offer the possibility of considering the number of changes also as a random variable, but their implementation is much more complex.

III) Results analysis and hypothesis evaluation

The random variables of the model are already defined, so we only need to run the model to obtain the results and check if our beliefs were right.
Using PyMC3, a probabilistic modeling Python package for running Monte Carlo methods, the implementation of the model is very simple. To run our model we have used Metropolis algorithm for sampling 20,000 observations for each of the parameters in the model.

As mentioned several times in the project, Probabilistic Programming returns the distribution for the variables in the model. Figures from 5.4 to 5.9 show the posterior distribution for each of the random variables and their interpretation. We sepparately discuss points and rebounds models:

Points model:

- Posterior distribution for λ_{P1}:

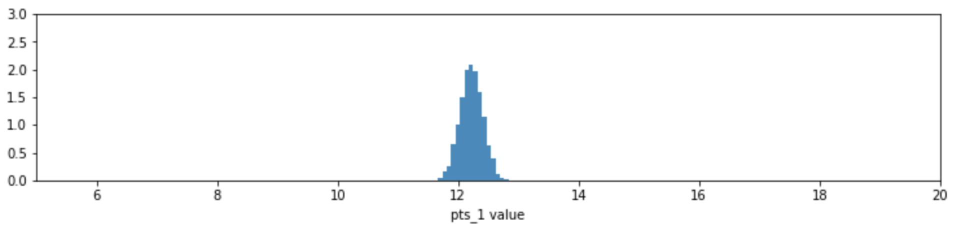

Figure 5.4: Results for λ_{P1}.

- Posterior distribution for λ_{P2}:

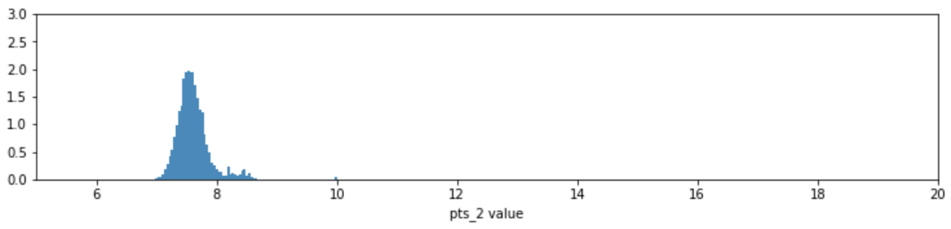

Figure 5.5: Results for λ_{P2}.

- Posterior distribution for G_P:

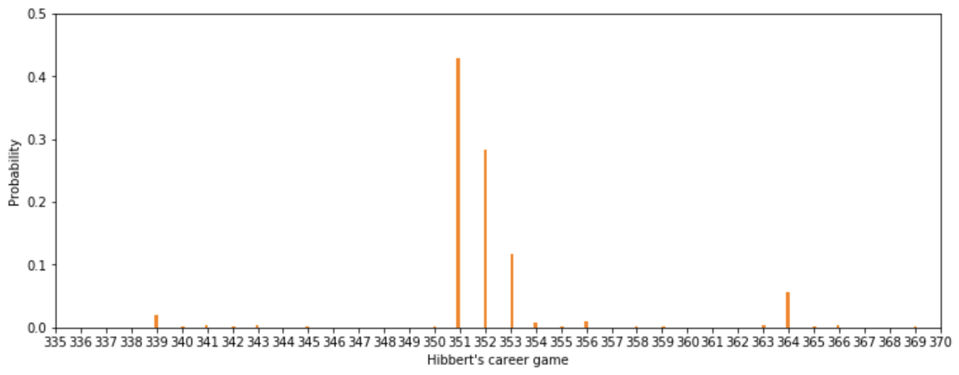

Figure 5.6: Results for G_P.

Interpretation of the results

First of all, we focus our analysis on the distribution for G_P. The graphic clearly shows that the change in the behaviour of the points scored by Roy Hibbert takes place between his 351-th and his 353-th game. There are some other games with a probability reasonably higher than zero, but we can consider them almost irrelevant.

On the other hand, we can see that both λ_{P1} and λ_{P2} have a bell-shaped distribution; in the case of λ_{P1}, centered around a value slightly greater than 12, and λ_{P2} is centered around a value lower than 8. This means that from game 1 to G_P, the points scored by Hibbert follow a Poisson distribution with parameter λ_{P1} close to 12, and from G_P to his last game, they follow a Poisson distribution with parameter λ_{P2} lower than 8.

Therefore, there is a sharp decrease in Roy Hibbert's scoring performance since the interval comprising his 351-th and his 353-th game.

Rebounds model:

- Posterior distribution for λ_{R1}:

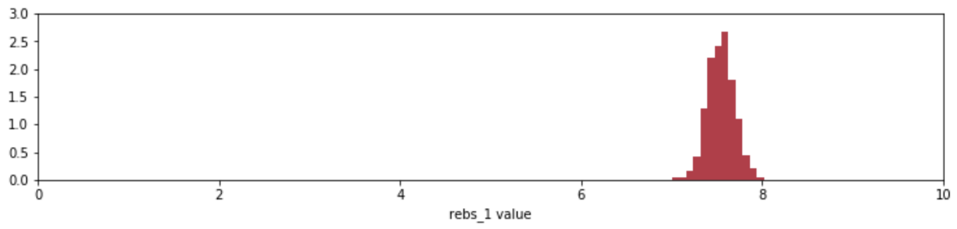

Figure 5.7: Results for λ_{R1}.

- Posterior distribution for λ_{R2}:

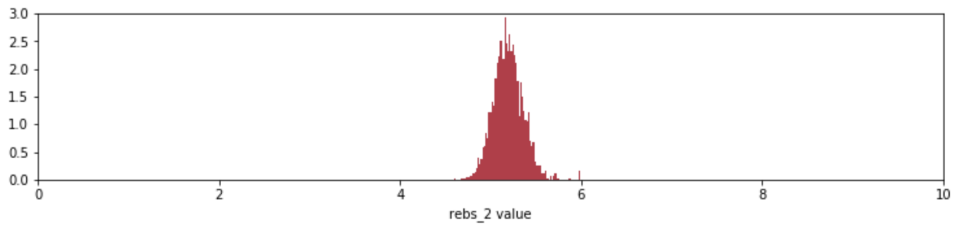

Figure 5.8: Results for λ_{R2}.

- Posterior distribution for G_R:

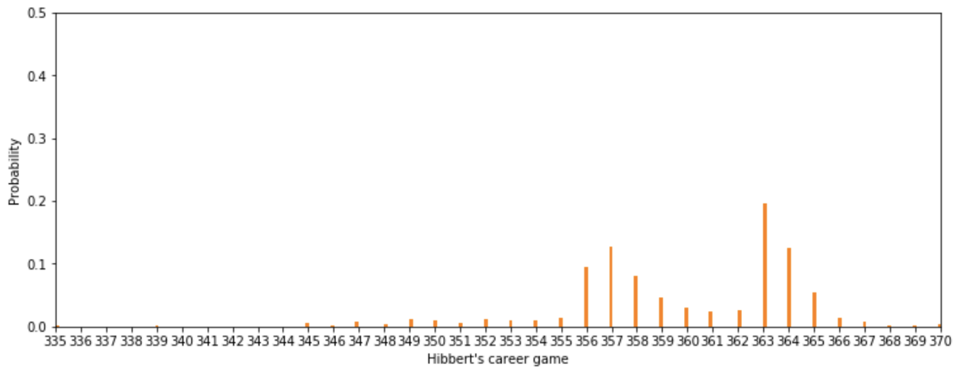

Figure 5.9: Results for G_R.

Interpretation of the results
Analogously to the analysis for points model, first we can see that the change of pattern in Roy Hibbert's rebounds takes place between his 356-th and his 366-th game.
λ_{R1} and λ_{R2} are respectively centered around a value close to 8 and a value between 5 and 5.5.
Again, as we have seen in the points model, there is a relevant fall in Roy Hibbert's rebounding performance which occurs between the games 356 and 366.

Summing up the results obtained, Roy Hibbert performance has clearly become worse both in scoring and rebounding. The moments where these falls have taken place are close from each other; the pattern change for points takes place between games 351 and 353, while the change for rebounds happens between games 356 and 366.
Therefore, we can guarantee that there is an obvious fall in Hibbert's general performance which begins around games 351-366. Now, we only need to compare this interval of games to the moment when Hibbert leaves Indiana to see if our hypothesis was right. Figure 5.10 shows a timeline which gives a clear representation of the events:

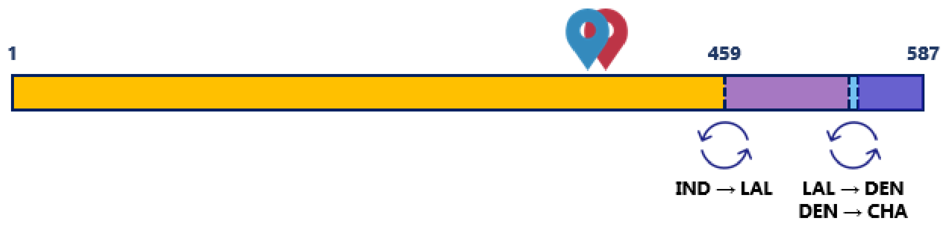

Figure 5.10: Timeline summing up the results of the model.

It is clear enough, just taking a look at the timeline above, that the decrease in Hibbert's performance took place a long time before he was traded to Los Angeles, therefore, we have to reject our hypothesis. Then, reconsidering our opinion, probably his poor performance was one of the reasons why the Indiana Pacers decided to trade him.

Remark 5.5. One of the advantages of Probabilistic Programming, and particularly *PyMC3*, is the easy implementation of the models. The core of this model has been implemented with the following few code lines:

```
with pymc.Model():
        alpha_pts = 1.0/mean_pts

        pts_1 = pymc.Exponential("pts_1",alpha_pts)
        pts_2 = pymc.Exponential("pts_2",alpha_pts)
        change_pts = pymc.DiscreteUniform("change_pts",
                           lower=0,upper=n_games)

        pts = pymc.math.switch(change_pts>=np.arange(n_games),
                pts_1,pts_2)

        sampling_pts = pymc.Poisson("sampling_pts",pts,
                             observed=Hibbert_pts)

        step_pts = pymc.Metropolis()
        trace_pts = pymc.sample(20000,tune=5000,step=step_pts)

        pts_1_samples = trace_pts["pts_1"]
        pts_2_samples = trace_pts["pts_2"]
        change_pts_samples = trace_pts["change_pts"]
```

Conclusions

Despite my great interest in the statistical field, I started from scratch in this topic. Bayesian statistics has been a completely new statistical perspective for me, as all I had seen until the realization of this project was from the Frequentist point of view. Understanding the Bayesian philosophy and comparing the two different points of view, Bayesian and Frequentist, has been very challenging and, at the same time, has given me a wider perspective of Statistics.
Statistical modeling, and particularly Probabilistic Programming, is at its peak. This creates a trade-off between the advantage of having countless information sources and the danger of selecting unreliable ones. It has been a laborious task to choose an accurate bibliography to develop the project from a solid theoretical basis. In relation to Probabilistic Programming implementation, having the *PyMC3* Python package, together with a reference work such as *Probabilistic Programming and Bayesian Methods for Hackers* [20], have paved my way.

Conceptually, the most remarkable conclusion of this project is how Probabilistic Programming deals with uncertainty. Probabilistic Programming returns the probability distribution of the model, which not only shows the values with higher probability, but also reflects uncertainty. By showing uncertainty in the results, the model gives a vast amount of information which can be lost when giving, for example, a formula or just the most probable values. Moreover, we do not confine our models to known distributions. By using inference, the resultant model might have an unrecognizable shape, which brings flexibility to the models and a closer fit to the desired model.

Our attention has been focused on MCMC methods and demonstrating their convergence. Following the lines of this project, further studies can be undertaken to analyze more complex MCMC methods such as the Multiple-try Metropolis algorithm or the reversible-jump algorithm. Being both very recent, the Multiple-try Metropolis is an upgrade of the Metropolis algorithm and achieves convergence faster by tuning the parameters involved in the algorithm, while the reversible-jump algorithm allows the simulation of problems on spaces with variable dimensions; therefore, the number of parameters involved in the modeling is not fixed. The implementation and comprehension of these methods is far more complicated than the ones studied in this project, but they have a much higher modeling capability. This step forward in the power of the methods opens new horizons and allows for the modeling of more complex problems.

Bibliography

[1] C. Robert and G. Casella, *Statistical Science, Vol. 26*, Institute of Mathematical Statistics, (2011), 102-115.

[2] E. Siniksaran, *Throwing Buffon's Needle with Mathematica*, Mathematica Journal, (2008), 73-84.

[3] N. Metropolis, *Los Alamos Science Special; The beginning of the Monte Carlo method*, Los Alamos National Laboratory, (1987), 126-130.

[4] A.W. Burks, *Electronic computing circuits of the ENIAC*, Institute of Radio Engineers, (August, 1947), 756-760.

[5] Official website of Los Alamos National Laboratory. [Consulted in March 2017]. Link: *http://www.lanl.gov*

[6] R. Rumi Rodriguez, *Modelos de redes bayesianas con variables discretas y continuas*, Universidad de Almería, (2004), 41-46.

[7] Alan E. Gelfland, Adrian F.M. Smith, *Journal of the American Statistical Association*, American Statistical Association, (1990), 399-409.

[8] L. Tierney, *The Annals of Statistics, Vol. 22, No. 4*, Institute of Mathematical Statistics, (December, 1994), 1701-1728.

[9] K.L. Mengersen, R.L. Tweedie, *Rates of convergence of the Hastings and Metropolis algorithms*, Queensland University of Technology and Colorado State University, (December, 1994).

[10] K. Sigman, *Discrete-time Markov Chains*, Columbia University, (2009).

[11] F.E. Beichelt, L.P. Fatti, *Stochastic Processes And Their Applications*, Taylor and Francis, (2002).

[12] S. Chib, E. Greenberg, *The American Statistician, Vol. 49, No. 4*, Taylor and Francis, (November, 1995).

[13] W. Hastings, *Monte Carlo sampling methods using Markov chains and their applications*, Biometrika, (1970), 97-109.

[14] S.M. Lynch, *Introduction to Applied Bayesian Statistics and Estimation for Social Scientists*, New York Springer, (2007).

[15] R.M. Neal, *The Annals of Statistics, Vol 31., No.3*, Institute of Mathematical Statistics, (2003), 705-721.

[16] L. Chen, Z. Qin, J.S. Liu, *Exploring Hybrid Monte Carlo in Bayesian Computation*, Springer Science, (1996), 705-721.

[17] R.M. Neal, *Bayesian Learning for Neural Networks*, University of Harvard, (2000), 55-97.

[18] M.D. Hoffmann, A. Gelman, *Journal of Machine Learning, No. 15*, University of Columbia, (2014), 1351-1381.

[19] Y.F. Atchadé, *An adaptative version for the Metropolis adjusted Langevin algorithm with a truncated drift*, University of Ottawa, (2005), 1-9.

[20] C. Davidson-Pilon, *Bayesian Methods for Hackers: Probabilistic Programming and Bayesian Inference*, Addison-Wesley, (2015).

[21] O. Ibe, *Markov Processes for Stochastic Modeling*, Elsevier Insights, (2008), 434-448.

[22] Official website of ESPN. *ESPN.com*. [Consulted in May 2017].
Link: *http://www.espn.com/nba/player/stats/_/id/3436/roy-hibbert*

[23] J. Alison, *Proof of Bayes Theorem*, University of Pennsylvania.

[24] M. Lugo, *A proof of the Central Limit Theorem*, University of California, Berkeley, (2011).

[25] C.M. Grinstead, J.L. Snell, *Law of Large Numbers*, Dartmouth College, (2011), 305-324.

Appendix

A.1 Useful theorems and definitions

This section includes three theorems which have a strong tie with the theme of this project, as well as some definitions of concepts mentioned tangentially during the project. The three theorems presented and demonstrated in the following pages are:

1. Bayes Theorem
2. Law of Large Numbers
3. Central Limit Theorem

A.1.1 Bayes Theorem

Theorem A.1. (Bayes Theorem) *Let A and B be two events such that $P(B) \neq 0$. Then,*

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

Proof. The probability of two events A and B happening can be defined as

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

or, equivalently,

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

Then,

$$P(A|B)P(B) = P(B|A)P(A)$$

Isolating the probability $P(A|B)$ the Bayes Theorem equation is obtained

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

□

A.1.2 Law of Large Numbers

Theorem A.2. (Law of Large Numbers) *Let* $X_1, X_2, ..., X_n$ *be independent random variables with expected value* $E(X_i) = \mu$ *and variance* $0 < \sigma^2 < \infty$*, and* $S_n = X_1 + X_2 + ... + X_n$*. Then, as* $n \to \infty$ *and for any* $\epsilon > 0$*,*

$$P\left(|\frac{S_n}{n} - \mu| \geq \epsilon\right) \to 0$$

Proof. $X_1, X_2, ..., X_n$ are independent variables, then, as $Var(X_i) = \sigma^2$ for every i,

$$Var(S_n) = Var(X_1 + X_2 + ... + X_n) = n\,Var(X_1) = n\,\sigma^2$$

$$Var\left(\frac{S_n}{n}\right) = \frac{n\,\sigma^2}{n^2} = \frac{\sigma^2}{n}$$

Also,

$$E\left(\frac{S_n}{n}\right) = \frac{1}{n}E(X_1 + X_2 + ... + X_n) = \frac{1}{n}(E(X_1) + E(X_2) + ... + E(X_n)) = \frac{1}{n}n\mu = \mu$$

Using Chebyshev's inequality (appendix), for any $\epsilon > 0$,

$$P\left(|\frac{S_n}{n} - \mu| \geq \epsilon\right) \leq \frac{Var\left(\frac{S_n}{n}\right)}{\epsilon^2} = \frac{\sigma^2}{n\epsilon^2}$$

For a fixed $\epsilon > 0$ and $n \to \infty$,

$$P\left(|\frac{S_n}{n} - \mu| \geq \epsilon\right) \to 0$$

□

Corollary A.3. *A trivial implication of the Law of Large Numbers is that*

$$P(|\frac{S_n}{n} - \mu| > \epsilon) \to 1$$

A.1.3 Central Limit Theorem

Theorem A.4. (Central Limit Theorem) *Let* $X_1, X_2, ..., X_n$ *be independent identically distributed random variables with expected value* $E(X_i) = \mu$ *and variance* $0 < \sigma^2 < \infty$*. Then, the random variable*

$$Z_n = \frac{\sqrt{n}(\bar{X} - \mu)}{\sigma}$$

converges in distribution to the standard normal variable, i.e.,

$$lim_{n\to\infty}P(Z_n \leq x) = \Phi(x)$$

where $\Phi(x)$ *is the standard normal cumulative distribution function.*

Proof. It suffices to proof the theorem for $\mu = 0$. In the case where $\mu \neq 0$ the same proof is valid for $Y_i = X_i - \mu$ instead of X_i. Let $M(t) = E(e^{tX_i})$ be the moment generating function (mgf) of X_i. The following steps will show that the mgf of Z_n tends to the mgf of the standard normal distribution. Defining $S_n = X_1 + ... + X_n$ as the sum of independent and identically distributed random variables X_i,

$$M_{S_n}(t) = E(e^{t\,\sum_{i=1}^{n} X_i}) = [E(e^{tX_1})]^n = [M(t)]^n$$

Proposition A.5. *If X and $Y = a + bX$ are random variables with mgf M_X and M_Y respectively, then $M_Y(t) = e^{at}M_X(bt)$.*

Proof. (Prop.)

$$M_Y(t) = E(e^{tY}) = E(e^{at+btX}) = E(e^{at}e^{btX}) = e^{at}E(e^{btX}) = M_X(bt)$$

□

Using the proposition above,

$$M_{S_n}(t) = [M(t)]^n \;\Rightarrow\; M_{Z_n}(t) = [M(\frac{t}{\sigma\sqrt{n}})]^n$$

Next lines will be in order to demonstrate that the following equality holds:

$$L := lim_{n\to\infty} n\, log\, M(\frac{t}{\sigma\sqrt{(n)}}) = \frac{t^2}{2}$$

Calling $x = \frac{1}{\sqrt{n}}$,

$$L = lim_{x\to 0}\frac{log\, M(tx/\sigma)}{x^2}$$

This limit results to be an indetermination $\frac{0}{0}$ given that $M(0) = 1$. Applying l'Hôpital's rule,

$$L = lim_{x\to 0}\frac{\frac{t\,M'(tx/\sigma)}{\sigma\,M(tx/\sigma)}}{2x}$$

Again, this limit is indeterminate, $\frac{0}{0}$. Differentiating again,

$$L = \frac{t}{2\sigma} lim_{x\to 0}\frac{M''(tx/\sigma)\frac{t}{\sigma}}{M(tx/\sigma) + xM'(tx/\sigma)\frac{t}{\sigma}} \;=\; \frac{t^2}{2\sigma^2}\frac{lim_{x\to 0}M''(tx/\sigma)}{lim_{x\to 0}M(tx/\sigma) + \frac{t}{\sigma}lim_{x\to 0}xM'(tx/\sigma)} =$$
$$= \frac{t^2}{2\sigma^2}\frac{M''(0)}{M(0) + \frac{t}{\sigma}0\,M'(0)}$$

Now, taking into account that $M(0) = 1$, $M'(0) = E(X) = \mu = 0$ and $M''(0) = E(X^2) = E(X)^2 + Var(X) = Var(X)$, the limit results to be

$$L = \frac{t}{2\sigma^2}\frac{\sigma^2}{1} = \frac{t^2}{2},$$

what was to be demonstrated. □

A.1.4 Definitions

Definition A.6. *Consider $\mathbb{S} = \{x_1, x_2, ..., x_n\}$ the state space of a Markov chain. Let p_i be the probability of the Markov chain being in the state i, and $q_{i \to j}$ the probability of the Markov chain going from the state i to the state j. We define the* ***detailed balance equation*** *as*

$$p_i q_{i \to j} = p_j q_{j \to i}$$

Remark A.7. A Markov chain is reversible if it satisfies the detailed balance equation for all states in $\mathbb{S}$.

Definition A.8. *Consider $\mathcal{M}$ a statistical modeling process. A* ***slice variable*** *of $\mathcal{M}$ is a random variable of the model which induces intervals whose endpoints become the sufficient statistics for the parameters posteriors.*

Definition A.9. *Let Θ be the sample space of a parameter, and $\mathcal{A}$ the sigma-algebra on Θ. A* ***transition kernel*** *is a function K defined on $\Theta \times \mathcal{A}$ such that, $\forall \theta \in \Theta, \forall A \in \mathcal{A}$,*

I. *$K(\theta, \cdot)$ is a probability measure.*

II. *$K(\cdot, A)$ is a measurable function.*

Definition A.10. *The* ***drift function*** *$\mathcal{D} : \mathcal{R}^n \to \mathcal{R}^m$ of a modeling process is a function such that*

I) *$\mathcal{D}(\vec{0}_n) = \vec{0}_n$,*

II) *$\mid \mathcal{D}(x) \mid \geq 1, \forall x \in \mathcal{R}^n$,*

and which is used for optimization purposes.

Definition A.11. *Let $M = (m_{ij})$ be a $n_r \times n_c$ matrix. We define the* ***Frobenius norm*** *of M as*

$$\mid M \mid_F := \sqrt{\sum_{i=1}^{n_r} \sum_{j=1}^{n_c} \mid m_{ij} \mid^2}$$

A.2 Programming codes

Probabilistic Programming is strictly related to Python, as Python includes a very powerful package for building and implementing Probabilistic Programming models; PyMC3. Therefore, all the programming-related work in this project has been created in Python.

This section includes all the pieces of code employed in the project.

A.2.1 Buffon's needle graph

The graph in Buffon's needle experiment explanation have been generated with this piece of code:

```
#Buffon's needle graph

%matplotlib inline
from matplotlib import pyplot as p
import numpy as np

theta = np.arange(0,np.pi,0.01)
f = 0.5*sin(theta)

p.xlim(0,np.pi)
p.ylim(0,0.5)
p.fill_between(theta,f,color="red")
```

A.2.2 Example 2.10.

The stationary distribution in the example at the end of the second chapter have been calculated employing these lines of code:

```
#Convergence example

from math import fabs

#tolerance
tol = 1
#vector of probabilities
p = [0.6,0.2,0.2]
#matrix of transition probabilities
M = [[0.6,0.3,0.1],
    [0.4,0.5,0.1],
    [0.3,0,0.7]]
#vector resultant of the product of p and M
pM = [0,0,0]

#product loop
while tol>1e-5:
    for i in range(len(pM)):
        for j in range(len(pM)):
            pM[i] += p[j] * M[j][i]

    #print vector of probabilities
    print("Vector of probabilities:")
    for value in pM:
        print(value)

    #calculation of the difference between p and pM
    tol=0
    for i in range(len(pM)):
        tol += tol + fabs(p[i]-pM[i])

    for i in range(len(pM)):
        p[i] = pM[i]
        pM[i] = 0
```

A.2.3 Example 4.2.

The three pieces of code below have been created to generate the graphs shown in the example 4.2., which intention is to demonstrate the importance of the Law of Large Numbers.
First, the declaration of the necessary packages and variables:

```
%matplotlib inline
import numpy as np
import pandas as pd
import statsmodels.api as sm
import sympy as sp
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from scipy import stats

n = 1
p = 1/37
obs = 10000
simulations = np.random.binomial(n,p,obs)

means = [10,50,100,500,1000,3000,7000,10000]
```

To plot the graph showing the 200 first observations of the sampling, the following code has been employed:

```
plt.plot(simulations[:200],"ro",alpha=0.5)
print("First 200 observations:", simulations[:200])
```

Finally, the following code lines create the graph showing how probabilities converge when the number of samples N increases.

```
vmeans = np.zeros(len(simulations))

for i in range(1,len(simulations)):
    vmeans[i] = np.mean(simulations[:i])

plt.plot(vmeans)
plt.axhline(y=0.027027027,color="#FF0080",alpha=0.5)
```

A.2.4 Example 4.4.

This piece of code plots a three-dimensional graph which shows the prior distribution of a model which has two random variables, and both of them follow a Normal distribution:

```
%matplotlib inline
import scipy.stats as s
from IPython.core.pylabtools import figsize
import numpy as np
import matplotlib.pyplot as p
from mpl_toolkits.mplot3d import Axes3D

figsize(12.5,4)

pp = p.figure()
mu_1 = 0
mu_2 = 2
var = 1

jet = p.cm.jet
x = np.linspace(-4,4,100)
y = np.linspace(-4,4,100)
X,Y = np.meshgrid(x,y)
xnorm = s.norm.pdf(x,mu_1,var)
ynorm = s.norm.pdf(y,mu_2,var)
M = np.dot(ynorm[:,None],xnorm[None,:])

threed = pp.add_subplot(122,projection="3d")
threed.plot_surface(X,Y,M,cmap=p.cm.jet,vmax=1,vmin=-0.1)
threed.view_init(azim=390)
```

A.2.5 Objective vs subjective priors

The graphs used in section 4.3., employed for the comparison between objective and subjective priors have been generated with the code below:

- Objective prior:

```
%matplotlib inline
import numpy as np
from scipy.stats import uniform
import matplotlib.pyplot as p

y_axis = np.array((0,2))
obj = np.linspace(uniform.ppf(0),uniform.ppf(1),100)

p.ylim(0,3)
p.title("Objective prior")
p.plot(obj,uniform.pdf(obj),alpha=1.5)
```

- Subjective prior:

```
%matplotlib inline
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as p
import matplotlib.mlab as m

y_axis = np.array((0,2))
subj = np.linspace(0,1,100)

p.ylim(0,3)
p.title("Subjective prior")
p.plot(subj,m.normpdf(subj,0.5,0.2),alpha=1.5)
```

A.2.6 Application case

Poisson distribution

The graph for representing the Poisson distributions with different parameters has been generated with the following code:

```
%matplotlib inline
from scipy.stats import poisson
from IPython.core.pylabtools import figsize
import patsy as patsy
import pandas as pd
import numpy as np
import pylab as plab
import matplotlib.pyplot as p

poisson_1 = poisson(1)
poisson_3 = poisson(3)
poisson_6 = poisson(6)
poisson_10 = poisson(10)
poisson_20 = poisson(20)

figsize(12,6)
x = np.arange(-1,200)

p.plot(x, poisson_1.pmf(x), color="#000000",
             label='$\lambda=1$')
p.plot(x, poisson_3.pmf(x), color="#8A2908",
             label='$\lambda=3$')
p.plot(x, poisson_6.pmf(x), color="#DF3A01",
             label='$\lambda=6$')
p.plot(x, poisson_10.pmf(x), color="#FF4000",
             label='$\lambda=10$')
p.plot(x, poisson_20.pmf(x), color="#FF8000",
             label='$\lambda=20$')

p.xlim(-0.5, 30)
p.ylim(0, 0.4)

p.xlabel('$x$ value')
p.ylabel("Probabilities")
p.legend(loc="upper right")
```

Libraries and function declarations

This piece of code includes the declaration of all the necessary functions and packages to run the Probabilistic Programming model, as well as the lecture of files containing the data:

```
%matplotlib inline
import scipy.stats as s
from IPython.core.pylabtools import figsize
import patsy as patsy
import pandas as pd
import numpy as np
import pylab as plab
import pymc3 as pymc
import theano.tensor as th
import matplotlib.pyplot as p

#Lecture of files containing Roy Hibbert points and rebounds#
Hibbert_pts = np.loadtxt("Roy_Hibbert_Pts.csv")
Hibbert_rebs = np.loadtxt("Roy_Hibbert_Rebs.csv")
n_games = len(Hibbert_pts)
```

Roy Hibbert data for points and rebounds

The following pieces of code have been used to plot a graphical representation of Roy Hibbert points and rebounds during his career:

- Points:

```
#Roy Hibbert points in each game
figsize(12.5,4)

p.bar(np.arange(n_games),Hibbert_pts,color="#348ABD")
p.xlabel("Hibbert's career game")
p.ylabel("Points")
p.xlim(0,n_games)
p.ylim(0,Hibbert_pts.max()+2)
```

- Rebounds:

```
#Roy Hibbert rebounds in each game
figsize(12.5,4)

p.bar(np.arange(n_games),Hibbert_rebs,color="#BD3446")
p.xlabel("Hibbert's career game")
p.ylabel("Rebounds")
p.xlim(0,n_games)
p.ylim(0,Hibbert_rebs.max()+2)
```

Points and rebounds models

Below, the code lines which implement both the points and rebounds models. In these lines, we define the distributions followed by the random variables in the model, as well as the instructions for executing the Metropolis algorithm to sample 20,000 observations for each variable.

1. Points model:

```
#Roy Hibbert points per game.
mean_pts = Hibbert_pts.mean()

#Definition of our model.
with pymc.Model():

    alpha_pts = 1.0/mean_pts

    pts_1 = pymc.Exponential("pts_1",alpha)
    pts_2 = pymc.Exponential("pts_2",alpha)
    change_pts = pymc.DiscreteUniform("change_pts",lower=0,upper=n_games)

    pts = pymc.math.switch(change_pts>=np.arange(n_games),pts_1,pts_2)

    sampling_pts = pymc.Poisson("sampling_pts",pts,observed=Hibbert_pts)

    step_pts = pymc.Metropolis()
    trace_pts = pymc.sample(20000,tune=5000,step=step_pts)

    pts_1_samples = trace_pts["pts_1"]
    pts_2_samples = trace_pts["pts_2"]
    change_pts_samples = trace_pts["change_pts"]
```

```
100%|██████████| 20000/20000 [00:11<00:00, 1740.70it/s]
```

2. Rebounds model:

```
#Roy Hibbert rebounds per game.
mean_rebs = Hibbert_rebs.mean()

#Definition of our model.
with pymc.Model():

    alpha_rebs = 1.0/mean_rebs

    rebs_1 = pymc.Exponential("rebs_1",alpha_rebs)
    rebs_2 = pymc.Exponential("rebs_2",alpha_rebs)
    change_rebs = pymc.DiscreteUniform("change_rebs",lower=0,upper=n_games)

    rebs = pymc.math.switch(change_rebs>=np.arange(n_games),rebs_1,rebs_2)

    sampling_rebs = pymc.Poisson("sampling_rebs",rebs,observed=Hibbert_rebs)

    step_rebs = pymc.Metropolis()
    trace_rebs = pymc.sample(20000,tune=5000,step=step_rebs)

    rebs_1_samples = trace_rebs["rebs_1"]
    rebs_2_samples = trace_rebs["rebs_2"]
    change_rebs_samples = trace_rebs["change_rebs"]
```

```
100%|██████████| 20000/20000 [00:11<00:00, 1696.28it/s]
```

Posterior distributions for Roy Hibbert points and rebounds

The following codes have been employed to generate the graphs which show the posterior distribution for each random variable in the model. For a clearer display, we have created separated codes for each variable:

- Points 1, λ_{P1}:

```
#Plot for pts_1
figsize(12.5,8)

ax = p.subplot(311)
p.hist(pts_1_samples,bins=80,alpha=1,color = "#348ABD",normed=True)
p.xlim([5,20])
p.ylim([0,3])
p.xlabel("pts_1 value")
```

- Points 2, λ_{P2}:

```
#Plot for pts_2
figsize(12.5,8)

ax = p.subplot(311)
p.hist(pts_2_samples,bins=80,alpha=1,color = "#348ABD",normed=True)
p.xlim([5,20])
p.ylim([0,3])
p.xlabel("pts_2 value")
```

- Change points, G_P:

```
#Plot change_pts
figsize(12.5,10)

ax = p.subplot(211)
pond_pts = 1.0/change_pts_samples.shape[0]*np.ones_like(change_pts_samples)
p.hist(change_pts_samples,bins=n_games,alpha=1,color = "#FF8000",weights=pond_pts,rwidth=4)
p.xticks(np.arange(n_games))
p.xlim([335,370])
p.ylim([0,0.5])
p.xlabel("Hibbert's career game")
p.ylabel("Probability")
```

- Rebounds 1, λ_{R1}:

```
#Plot for rebs_1
figsize(12.5,8)

ax = p.subplot(311)
p.hist(rebs_1_samples,bins=80,alpha=1,color = "#BD3446",normed=True)
p.xlim([0,10])
p.ylim([0,3])
p.xlabel("rebs_1 value")
```

- Rebounds 2, λ_{R2}:

```
#Plot for rebs_2
figsize(12.5,8)

ax = p.subplot(311)
p.hist(rebs_2_samples,bins=80,alpha=1,color = "#BD3446",normed=True)
p.xlim([0,10])
p.ylim([0,3])
p.xlabel("rebs_2 value")
```

- Change rebounds, G_R:

```
#Plot change_rebs
figsize(12.5,10)

ax = p.subplot(211)
pond_rebs = 1.0/change_rebs_samples.shape[0]*np.ones_like(change_rebs_samples)
p.hist(change_rebs_samples,bins=n_games,alpha=1,color = "#FF8000",weights=pond_rebs,rwidth=4)
p.xticks(np.arange(n_games))
p.xlim([335,370])
p.ylim([0,0.5])
p.xlabel("Hibbert's career game")
p.ylabel("Probability")
```