University of Innsbruck
Department of Mathematics

Bachelor Thesis

Monte Carlo methods

Author:
Lukas Einkemmer

Advisor:
Dr. Alexander Ostermann

Contents

1 Introduction **1**

2 Prerequisites from probability theory **2**

3 Monte Carlo integration **4**
3.1 Plain Monte Carlo integration 4
3.2 Importance sampling 5
3.3 The VEGAS algorithm 7
3.4 Applications 8

4 Markov chain Monte Carlo methods **10**
4.1 Introduction to continuous-state Markov chains 10
4.2 The Metropolis-Hastings algorithm 14
4.3 Applications 16

5 Numerical implementation **22**
5.1 Optimization 22
5.2 Tree summation 23
5.3 Random number generators 23

6 Summary **24**

References **24**

A Source code of algorithms and methods **26**

B Source code of examples **29**

March 7, 2010

1 Introduction

In this section we will discuss the shortfalls of classical methods of numerical integration in multidimensional space (usually $\mathbb{R}^d$, $d \geq 4$). Thus, we motivate the introduction of Monte Carlo integration which will be the main focus of this bachelor thesis (and is formally introduced in section 3).

For simplicity we limit ourselves to the composite trapezoidal rule (as a method of classical numerical integration). For our purpose, it contains all the features necessary to demonstrate the need for alternative multidimensional integration techniques, but eliminates a number of technical complications (unequally spaced points, for example). For a twice continuously differentiable function $f : [a,b] \subset \mathbb{R} \to \mathbb{R}$ the composite trapezoidal rule is given by (see e.g. [5, p. 428-435])

$$\begin{aligned}\int_a^b f(x)\,\mathrm{d}x &= \frac{b-a}{n}\sum_{i=0}^{\overset{\star}{n}} f(x_i) - \frac{(b-a)^3}{12n^2} f''(\xi)\\ &= \frac{b-a}{n}\sum_{i=0}^{\overset{\star}{n}} f(x_i) + \mathcal{O}(n^{-2}),\end{aligned}$$

for some $\xi \in (a,b)$ and where $n+1$ is the number of function evaluations (with $n \in \mathbb{N}_{\geq 0}$). In addition

$$x_i = a + \frac{i}{n}(b-a), \qquad 0 \leq i \leq n$$

and

$$\sum_{i=0}^{\overset{\star}{n}} f(x_i) := \frac{1}{2}f(x_0) + \sum_{i=1}^{n-1} f(x_i) + \frac{1}{2}f(x_n).$$

This method can be extended (in an obvious way) to integration of $f : J \subset \mathbb{R}^d \to \mathbb{R}$ by using Fubini's theorem (see e.g. [17])

$$\begin{aligned}\int_J f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} &= \int_{J_1} \cdots \int_{J_d} f(\boldsymbol{x})\,\mathrm{d}x_1 \ldots \mathrm{d}x_d\\ &= \frac{\mathrm{vol}J}{n^d}\sum_{i_1=0}^{\overset{\star}{n}} \cdots \sum_{i_d=0}^{\overset{\star}{n}} f(\boldsymbol{x_i}) + \mathcal{O}(n^{-2})\\ &= \frac{\mathrm{vol}J}{n^d}\sum_{i_1=0}^{\overset{\star}{n}} \cdots \sum_{i_d=0}^{\overset{\star}{n}} f(\boldsymbol{x_i}) + \mathcal{O}(N^{-2/d})\end{aligned}$$

with multi-index $\boldsymbol{i} = (i_1, i_2, \ldots, i_d)$ and where $J = J_1 \times \cdots \times J_d$ is the Cartesian product of the intervals $J_1, \ldots, J_d \subset \mathbb{R}$. The volume of J is denoted by vol$J$. In addition, $N = (n+1)^d = \mathcal{O}(n^d)$ is the number of function evaluations we need to perform.

In light of this result we can characterize our dilemma as follows:

1. The number of function evaluations $N = (n+1)^d$ increases exponentially as the number of dimensions increases.
2. The error bound $\mathcal{O}(N^{-2/d})$ becomes worse as the number of dimensions increases.

Collectively this is referred to as the **curse of dimensionality** (the term is due to R. Bellman, see [21]). For instance, for $d = 10$ the error bound is $\mathcal{O}(N^{-1/5})$. Therefore, we must increase the number of function evaluations by a factor of 32 to double the accuracy.

In the following section we will develop a method which (independently of dimension) scales as $\mathcal{O}(N^{-1/2})$. This is accomplished by evaluating the function on randomly determined points. For an illustration of some typical convergence rates see Figure 1.1.

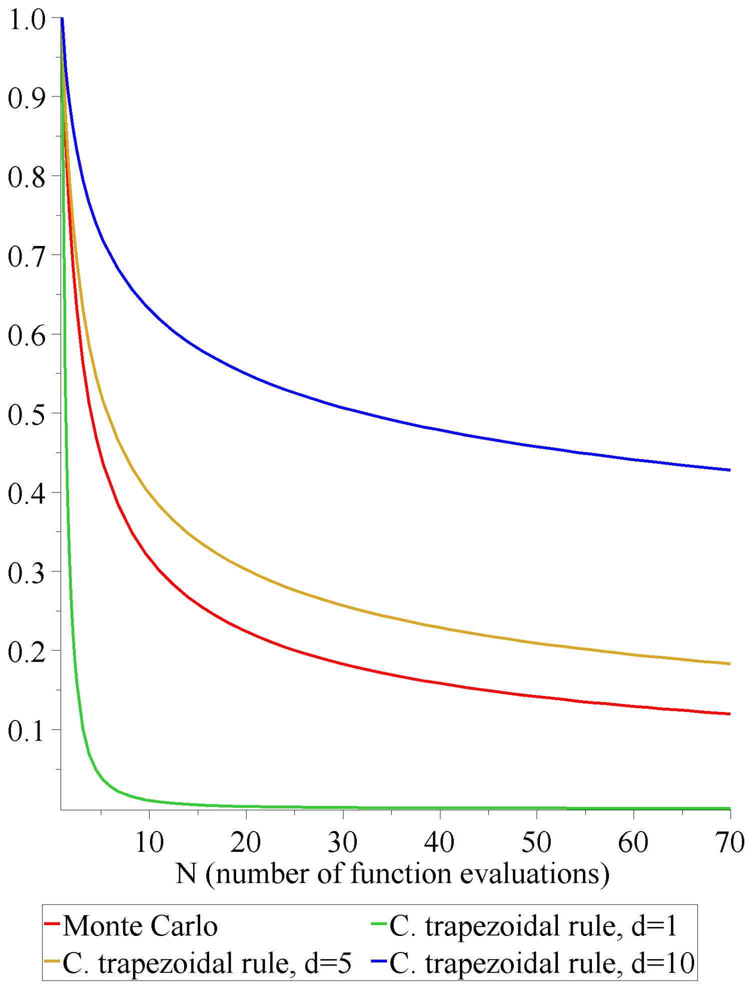

Figure 1.1: Typical convergence rates.

Furthermore, we should remark at this point that if we integrate f over a set V which can't be written as a product of intervals it is necessary to integrate $f \cdot \chi_V$ over J where $V \subset J$. However, in general $f \cdot \chi_V \notin \mathcal{C}^2(J)$ even if $f \in \mathcal{C}^2(J)$; therefore, the error bound of the composite trapezoidal rule as derived above is no longer valid.

As will be shown in section 3, the error bounds for Monte Carlo integration requires only (the much weaker) assumption of integrability. Additionally, we can integrate over V directly, if we are able to generate uniformly distributed random numbers on V and know the value of vol$V$.

2 Prerequisites from probability theory

In this section we will briefly review the concepts from probability theory which are needed in section 3 and 4. Most of the definitions and theorems in this section are taken from [20] and [26] respectively.

Definition 2.1. We call $(\Omega, \mathcal{A}, P)$ a **probability space** if

1. $\Omega \neq \emptyset$ (Ω is called the set of all **outcomes**).
2. $\mathcal{A} \subset \mathcal{P}(\Omega)$ is a σ-algebra ($\mathcal{A}$ is called the set of all **events**).
3. P is a probability measure on $\mathcal{A}$.

Definition 2.2. Let $(\Omega, \mathcal{A}, P)$ be a probability space. We call $X : \Omega \to \mathbb{R}$ a **random variable** if $\{\omega \in \Omega : X(\omega) < x\} \in \mathcal{A}$ for every $x \in \mathbb{R}$. A finite collection of random variables is called a **random vector**.

Definition 2.3. We denote a **uniformly distributed** random vector over $V \subset \mathbb{R}^d$ by writing $\boldsymbol{X} \sim \mathcal{U}(V)$. We denote a n**ormally distributed** random variable with mean μ and standard deviation σ by writing $X \sim \mathcal{N}(\mu, \sigma)$.

It should be noted that in the previous definition no explicit reference is made with respect to the probability space that is attached to each random variable. This space is to be understood implicitly (for more information see a text on probability theory, e.g. [26]). At this point it seems prudent to explain our notation. We denote random variables by capital letters and vectors by bold letters, i.e. x, $\boldsymbol{x}$, X, and $\boldsymbol{X}$ would denote a variable, a vector, a random variable, and a random vector respectively.

Definition 2.4. We call a collection of random variables/vectors $X_1, \dots, X_i$ **i.i.d (independent and identically distributed)** if their distributions are equal and every two of them are pairwise independent.

Therefore, we require not only that the random vectors are pairwise independent and identically distributed, but in addition that the random variables in every random vector are i.i.d, as a collection of random variables.

Definition 2.5. The probability distribution of a random variable X is denoted by $\mathcal{L}(X)$.

The next theorem introduces the two most commonly employed notions of convergence of a sequence of random variables.

Definition 2.6. Let $(X_i)_{i\geq 1}$ be a sequence of random variables and suppose X is a random variable. Then

$$X_n \xrightarrow{\mathrm{D}} X :\Longleftrightarrow \lim_{n\to\infty} \|\mathcal{L}(X_n) - \mathcal{L}(X)\| = 0$$
$$X_n \xrightarrow{\mathrm{a.s.}} X :\Longleftrightarrow P\{\omega \in \Omega : |X_n(\omega) - X(\omega)| \to 0\} = 1$$

is called convergence in **distribution** and convergence **almost surely** respectively. The **total variation norm** of the probability distribution μ is denoted by $\|\mu\| := \sup_{A\in\mathcal{A}} |\mu(A)|$ (see [22, p. 109]).

If $\Omega \subset \mathbb{R}$ the following **equivalent** definition of convergence in distribution is more commonly employed.

Definition 2.7. Let $\Omega \subset \mathbb{R}$. Then

$$X_n \xrightarrow{\mathrm{D}} X :\Longleftrightarrow \forall x \in \mathbb{R} : F(x) \to F_X(x),$$

where F_X denotes the **cumulative distribution function (cdf)** of X.

The next two theorems establish the limiting behavior of a sum of random variables and will therefore be vital in the derivation of Monte Carlo integration methods.

Theorem 2.8. ***(Strong law of large numbers).*** *Let $(X_i)_{i\geq 1}$ be an i.i.d sequence of random variables where $\mathrm{E}(X_i) = \mu$, $\mu \in \mathbb{R}$. Then*

$$\frac{1}{N}\sum_{i=1}^{N} X_i \xrightarrow{a.s.} \mu, \qquad \text{as } N \to \infty.$$

Proof. See e.g. [26, p. 408-410]. □

Theorem 2.9. ***(Central limit theorem).*** *Let $(X_i)_{i\geq 1}$ be an i.i.d. sequence of random variables where $\mathrm{E}(X_i) = \mu$, $\mu \in \mathbb{R}$ and $\mathrm{Var}(X_i) = \sigma^2$, $\sigma^2 \in \mathbb{R}$. Then*

$$\frac{\frac{1}{N}\sum_{i=1}^{N} X_i - \mu}{\sigma/\sqrt{N}} \xrightarrow{D} \mathcal{N}(0,1), \qquad \text{as } N \to \infty.$$

Proof. See e.g. [26, p. 399-401]. □

3 Monte Carlo integration

3.1 Plain Monte Carlo integration

This section is mainly based on [30, Chap. 3] and [6]. Consider the following integral

$$I := \int_V f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}$$

which needs to be computed numerically and where $V \subset \mathbb{R}^d$ with vol$V < \infty$. The idea of Monte Carlo integration is to randomly sample a number of points from V and evaluate the function f at those points. Similar to classical (numerical) integration the average of those values (up to multiplication with vol$V$) converges to I. The next theorem provides the mathematical justification of this statement.

Theorem 3.1. ***(Monte Carlo integration).*** *Let $f \in L^1(V)$ and suppose $(\boldsymbol{X}_i)_{i=1}^N$ is an i.i.d. collection of random vectors, where $\boldsymbol{X}_i \sim U(V)$. Then*

$$I_N := \frac{\mathrm{vol}V}{N}\sum_{i=1}^{N} f(\boldsymbol{X}_i) \xrightarrow{a.s.} I, \qquad \textit{as } N \to \infty.$$

Proof. Clearly $f(\boldsymbol{X}_i)$ is a random variable. Furthermore,

$$\begin{aligned}
\mathrm{E}(f(\boldsymbol{X}_i)) &= \int_V f(\boldsymbol{X}_i)\,dU(V) \\
&= \frac{1}{\mathrm{vol}V}\int_V f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}.
\end{aligned}$$

Since the integral is finite, our result follows from the strong law of large numbers (Theorem 2.8). □

Thus, we have established that it is possible to find a sequence of random variables that converges to the desired integral. The next step is to determine the rate of convergence. Remember, that I_N is a random variable, i.e. we can only find a probabilistic error bound. The next theorem computes the variance of I_N. In order to get a probabilistic error bound, the central limit theorem (Theorem 2.9) will then be employed.

Theorem 3.2. *Let $f \in L^1(V)$ and suppose $(\boldsymbol{X}_i)_{i=1}^N$ is an i.i.d collection of random vectors, where $\boldsymbol{X}_i \sim U(V)$. Then*

$$\mathrm{Var}\,(I_N) = \frac{(\mathrm{vol}V)^2}{N}\mathrm{Var}(f(\boldsymbol{X}_1)).$$

Proof. Since the independence assumption implies $\mathrm{Cov}(f(\boldsymbol{X}_i), f(\boldsymbol{X}_j)) = 0$ for $i \neq j$, it follows that

$$\begin{aligned}
\mathrm{Var}\left(\frac{\mathrm{vol}V}{N}\sum_{i=1}^{N} f(\boldsymbol{X}_i)\right) &= \frac{(\mathrm{vol}V)^2}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\mathrm{Cov}\,(f(\boldsymbol{X}_i), f(\boldsymbol{X}_j)) \\
&= \frac{(\mathrm{vol}V)^2}{N^2}\sum_{i=1}^{N}\mathrm{Var}(f(\boldsymbol{X}_i)) \\
&= \frac{(\mathrm{vol}V)^2}{N}\mathrm{Var}\,(f(\boldsymbol{X}_1))\,.
\end{aligned}$$

It should be noted that we used the fact that $f \in L^2(V)$ since

$$\begin{aligned}
\mathrm{Var}\,(f(\boldsymbol{X}_i)) &= \int_V \left[f(\boldsymbol{X}_i) - \mathrm{E}\,(f(\boldsymbol{X}_i))\right]^2\,\mathrm{d}U(V) \\
&= \frac{1}{\mathrm{vol}V}\int_V \left[f(\boldsymbol{x}) - \frac{I}{\mathrm{vol}V}\right]^2\,\mathrm{d}\boldsymbol{x}
\end{aligned}$$

requires the square integrability of f. However, since vol$V$ is finite $f \in L^1(V)$ implies $f \in L^2(V)$ (by the Cauchy-Schwarz inequality). $\square$

Theorem 3.3. *Let $f \in L^1(V)$ and suppose $(\boldsymbol{X}_i)_{i=1}^N$ is an i.i.d collection of random vectors, where $\boldsymbol{X}_i \sim U(V)$. Furthermore, $s \in \mathbb{R}_{\geq 0}$. Then (for large enough N)*

$$P\left\{|I_N - I| \geq s \cdot \frac{\text{vol}V\sigma(f(\boldsymbol{X}_1))}{\sqrt{N}}\right\} \approx 2\Phi(-s),$$

where Φ is the cdf of $\mathcal{N}(0,1)$.

Proof. Since $(\boldsymbol{X}_i)_{i=1}^N$ is i.i.d., so is $(f(\boldsymbol{X}_i))_{i=1}^N$ and we can invoke the central limit theorem (Theorem 2.9) to obtain

$$\frac{I_N - I}{\text{vol}V\sigma\left(f(\boldsymbol{X}_1)\right)/\sqrt{N}} = \frac{\frac{1}{N}\sum_{i=1}^N \text{vol}V f(\boldsymbol{X}_i) - I}{\text{vol}V\sigma\left(f(\boldsymbol{X}_1)\right)/\sqrt{N}} \xrightarrow{D} \mathcal{N}(0,1).$$

If N is large enough, we get an approximation to the limiting behavior, from which our result follows. $\square$

Continuing the discussion started in the introduction, we note that the requirements of the above error bound is merely $f \in L^1(V)$ (compared to $f \in \mathcal{C}^2(V)$ as in the classical method). That is, if we are unable to sample uniformly from V (or if we are unable to compute vol$V$) it is still possible to use our error bound as long as we are able to find a product of intervals J such that $V \subset J$ and $f \cdot \chi_I \in L^1(J)$. We will revisit this approach at the end of this section (e.g. in Example 3.6).

In a computer program we output the standard deviation of our final approximation (The standard deviation is preferred over the variance since it is measured in the same units as our random variable $f(\boldsymbol{X}_i)$). However, since $\sigma(f(\boldsymbol{X}_i))$ is usually not known, we substitute the following (well known) approximation

$$\text{Var}(I_N) \approx S_N^2 := \frac{1}{N-1}\sum_{i=1}^N (f(\boldsymbol{x}_i) - I_N)^2 = \frac{1}{N-1}\sum_{i=1}^N f^2(\boldsymbol{x}_i) - \frac{N}{N-1}I_N^2,$$

where $\boldsymbol{x}_i$ are the points which have been sampled from the random variables $\boldsymbol{X}_i$ (i.e. $\boldsymbol{x}_i$ is a realization of the random variable $\boldsymbol{X}_i$).

An implementation of the plain Monte Carlo method can be found in Listing 2.

3.2 Importance sampling

The method introduced in the previous section forms the basis of all Monte Carlo integration methods. However, we can improve this method by reducing the variance of I_N that is given in Theorem 3.2. In this section we will introduce a method called importance sampling which uses a non-uniform sampling distribution to focus sampled points in areas where $|f|$ is large (i.e. f has a large contribution to the integral).

Other methods to reduce the variance of I_N have also been used. The exploitation of correlated variables (the method of antithetic variables as described, e.g., in [6, p. 14] or [30, p. 15]) and the division of V into subsets with different sample sizes in each subset (the method of stratification as described, e.g., in [6, p. 16-19] or [30, p. 13-14]) being the most common. In this bachelor thesis, however, we will focus on importance sampling, which, in addition to being a valuable technique on its own, forms the basis of the VEGAS algorithm as described in section 3.3.

Suppose $p : V \to \mathbb{R}$ is a **probability density function (pdf)** that is approximately proportional to $|f|$. In this case, the following theorem provides the mathematical formulation of importance sampling.

Theorem 3.4. *Let $f \in L^1(V)$, $p : V \to \mathbb{R}$ be a pdf such that $f^2/p \in L^1(V)$, and suppose $(\boldsymbol{X}_i)_{i=1}^N$ is an i.i.d collection of random vectors. Then*

$$I_{N;p} := \frac{1}{N}\sum_{i=1}^N \frac{f(\boldsymbol{X}_i)}{p(\boldsymbol{X}_i)} \xrightarrow{a.s.} I, \qquad \text{as } N \to \infty \tag{3.1}$$

and

$$\operatorname{Var}(I_{N;p}) = \frac{1}{N}\int_V \left(\frac{f(\boldsymbol{x})}{p(\boldsymbol{x})} - I\right)^2 p(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}.$$

Proof. Consider,

$$\begin{aligned}
\operatorname{E}\left(\frac{f(\boldsymbol{X}_i)}{p(\boldsymbol{X}_i)}\right) &= \int_V \frac{f(\boldsymbol{X}_i)}{p(\boldsymbol{X}_i)}\,\mathrm{d}p \\
&= \int_V \frac{f(\boldsymbol{x})}{p(\boldsymbol{x})} p(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \\
&= \int_V f(\boldsymbol{x})\,d\boldsymbol{x}.
\end{aligned}$$

Therefore, the first result follows from the law of large numbers. Furthermore, the second result follows from

$$\begin{aligned}
\operatorname{Var}(I_{N;p}) &= \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} \operatorname{Cov}\left(\frac{f(\boldsymbol{X}_i)}{p(\boldsymbol{X}_i)}, \frac{f(\boldsymbol{X}_j)}{p(\boldsymbol{X}_j)}\right) \\
&= \frac{1}{N^2}\sum_{i=1}^{N} \operatorname{Var}\left(\frac{f(\boldsymbol{X}_i)}{p(\boldsymbol{X}_i)}\right) \\
&= \frac{1}{N}\operatorname{Var}\left(\frac{f(\boldsymbol{X}_1)}{p(\boldsymbol{X}_1)}\right) \\
&= \frac{1}{N}\int_V \left(\frac{f(\boldsymbol{X}_1)}{p(\boldsymbol{X}_1)} - I\right)^2 \mathrm{d}p \\
&= \frac{1}{N}\int_V \left(\frac{f(\boldsymbol{x})}{p(\boldsymbol{x})} - I\right)^2 p(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}.
\end{aligned}$$

□

As already mentioned, if we choose the pdf p such that $f/p \approx \text{const}$; then, $\operatorname{Var}(I_{N;p}) \approx 0$ which has the desired effect of minimizing the variance of $I_{N;p}$. Now, naively we might choose

$$p(\boldsymbol{x}) = \frac{|f(\boldsymbol{x})|}{\int_V |f(\boldsymbol{x})|\,\mathrm{d}\boldsymbol{x}},$$

which clearly is a probability density function. This, however, isn't feasible, since we must know the normalization constant and therefore have to solve a problem equally complicated as our original task to compute I. Even if we could sample from p without computing the normalization constant (such an algorithm will be introduced in section 4.2) it is still necessary to compute $p(\boldsymbol{x}_i)$ which once again would require the computation of the normalization constant. Therefore, we must find a function p that (up to a normalization constant which we are able to compute) approximates $|f|$ as closely as possible.

As in the previous section, we often need the following (well known) estimate of $\operatorname{Var}(I_{N;p})$.

$$\begin{aligned}
\operatorname{Var}(I_{N;p}) \approx S_{N;p}^2 := \frac{1}{N}\frac{1}{N-1}\sum_{i=1}^{N}\left(\frac{f(\boldsymbol{x}_i)}{p(\boldsymbol{x}_i)} - I_{N;p}\right)^2 &= \frac{1}{N}\left[\frac{1}{N-1}\sum_{i=1}^{N}\frac{f^2(\boldsymbol{x}_i)}{p^2(\boldsymbol{x}_i)} - \frac{N}{N-1}I_{N;p}^2\right] \\
&= \frac{1}{N(N-1)}\sum_{i=1}^{N}\frac{f^2(\boldsymbol{x}_i)}{p^2(\boldsymbol{x}_i)} - \frac{1}{N-1}I_{N;p}^2.
\end{aligned}$$

3.3 The VEGAS algorithm

The approach to variance reduction discussed in the previous section (importance sampling) requires a good estimate of the function to be integrated beforehand. Often, however, such knowledge can't be assumed and we need an algorithm that adaptively learns about the function and chooses the appropriate sampling distribution as it proceeds. In this section we will introduce one such method which is referred to as the VEGAS algorithm (as described in [18]).

The basic idea is to start with a uniform distribution in the first iteration of the algorithm and then adjust the probability distribution at the end of every iteration in accordance with the previously sampled points. The different iterations are then (weighted by the estimated variance) combined to a single solution. It is also possible to use distinct sample sizes in different iterations (e.g. we often use a couple of iterations to get a good probability distribution and then a much larger sample to compute the integral).

However, the problem of approximating the probability distribution by a number of sampled values remains. The naive method would include the construction of a grid. However, suppose we have d dimension and n bins per dimension; then, our grid would contain

$$n^d$$

elements; however, usually $N \ll n^d$. Clearly, it is not feasible to estimate the values of this grid. If we suppose that p can be separated such that

$$p(\boldsymbol{x}) = \prod_{i=1}^{d} p_i(x_i),$$

where p_i are probability density functions of a single variable. Then there are only $d \cdot m$ elements in our grid (every dimension is treated separately, disregarding all other dimensions). The validity of this assumption, clearly, influences the efficiency of the VEGAS algorithm. However, the worst case scenario is a return to plain Monte Carlo integration with the additional computing cost which is necessary to determine the grid.

In the first iteration we start with a uniform grid and adjust the bins (separately per dimension) at every iteration such that the size of a bin is inversely proportional to $|f|$. This leads to the VEGAS algorithm.

Algorithm 3.5. ***(VEGAS).***

1. Setup a matrix $P \in \mathbb{R}^{d \times n}$ where $n \in \mathbb{N}$ is arbitrary and $P_{ij} := \frac{j}{n}$.
2. Set $\alpha := 0$.
3. Set $\alpha := \alpha + 1$.
4. Sample $j_i \in \mathbb{N}$ uniformly over $1 \leq j \leq n$, where $1 \leq i \leq d$.
5. Sample $\boldsymbol{x}_1$ uniformly from the volume $V_{\boldsymbol{x}_1} := \prod_{i=1}^{d}(a_k + (b_k - a_k)(P_{ij_{i-1}}, P_{ij_i}))$, where $P_{ij_0} = 0$ for all $1 \leq i \leq d$.
6. Repeat step (5) for $\boldsymbol{x}_2, \ldots, \boldsymbol{x}_{N_\alpha}$.
7. Calculate
$$S_\alpha := \frac{1}{N_\alpha} \sum_{k=1}^{N_\alpha} \frac{f(\boldsymbol{x}_k)}{p(\boldsymbol{x}_k)}$$
and
$$\sigma_\alpha^2 := \frac{1}{N_\alpha - 1} \frac{1}{N_\alpha} \sum_{k=1}^{N_\alpha} \frac{f^2(\boldsymbol{x}_k)}{p^2(\boldsymbol{x}_k)} - \frac{1}{N_\alpha - 1} S_\alpha^2,$$
where $p(\boldsymbol{x}_i) = \left(n^d \mathrm{vol} V_{\boldsymbol{x}_i}\right)^{-1}$.

8. Adjust P_{ij} (for every i) such that $S_{\alpha_j} := \frac{1}{N_\alpha}\sum_{\boldsymbol{x}\in V_j}|f(\boldsymbol{x})| \approx \frac{1}{n}\frac{1}{N_\alpha}\sum_{k=1}^{N_\alpha}|f(\boldsymbol{x}_k)|$, where $V_j = I_1 \times \cdots \times I_{i-1} \times (a_i + (b_i - a_i)(P_{i(j-1)}, P_{ij})) \times I_{i+1} \times I_d$ and $P_{i0} = 0$.
9. if $\alpha < m$ goto step (3).
10. Calculate

$$\bar{\sigma}^2 := \left(\sum_{\alpha=1}^{m} \frac{1}{\sigma_\alpha^2}\right)^{-1},$$

$$I := \bar{\sigma}^2 \sum_{\alpha=1}^{m} \frac{S_\alpha}{\sigma_\alpha^2}.$$

We can use $\bar{\sigma}$ as an estimate of the overall standard deviation of I. The VEGAS algorithm is implemented in Listing 3.

3.4 Applications

Example 3.6. (Decay rate of the muon).
In the field of particle physics, the decay rates of elementary particles are of interest. In this example we will calculate the mean lifetime of the following muon decay channel

$$\mu \to e + \nu_\mu + \overline{\nu}_e$$

i.e., the muon decays into an electron (e), a muon neutrino (ν_μ) and an electron antineutrino ($\overline{\nu}_e$). Since this bachelor thesis is a mathematical treatment of Monte Carlo integration, we will omit most of the derivation that results in an integral suitable for Monte Carlo integration (The interested reader is referred to the more detailed discussion found in [12, p. 304-309]). In what follows, we need the numerical values of the physical constants listed in Table 3.1 (all digits shown are experimentally significant).

Description	Symbol	Value
Mass of the muon	m_μ	0.105 GeV
Mass of the electron	m_e	0.510 MeV
Mass of the W boson	m_W	80.4 GeV
Weak coupling constant	g_w	0.66
Experimental value of the mean muon lifetime	τ_E	2.197 ns

Table 3.1: Physical constants (taken from [4]).

Except for the mean lifetime of the muon (which is measured in seconds), we used in Table 3.1 and will continue to use natural units. In those units energy and mass is measured in GeV (Giga electronvolt). In addition, $\hbar = 1$ and $c = 1$ (i.e. the reduced Planck constant and the speed of light are one unit of action and speed respectively).

Now, in our example Fermi's golden rule for single particle decay states that to first order in g_w we have (see [12, p. 195])

$$\Gamma = \int \frac{|\mathcal{M}|^2(2\pi)^4}{2m_\mu}\left[\frac{\mathrm{d}\boldsymbol{p}_1}{(2\pi)^3 2E_1}\frac{\mathrm{d}\boldsymbol{p}_2}{(2\pi)^3 2E_2}\frac{\mathrm{d}\boldsymbol{p}_3}{(2\pi)^3 2E_3}\right]\delta\left(\boldsymbol{p}_A - \sum_{i=1}^{3}\boldsymbol{p}_i\right)\delta\left(E_A - \sum_{i=1}^{3}E_i\right)\mathrm{d}\boldsymbol{p}_1\mathrm{d}\boldsymbol{p}_2\mathrm{d}\boldsymbol{p}_3, \tag{3.2}$$

where $E_i,\ i \in \{1, 2, 3, A\}$ are the (relativistic) energies of the particles (e, ν_μ, $\overline{\nu}_e$ and μ), $\boldsymbol{p}_A$ is the classical momentum (three momentum) of the decaying particle and $\boldsymbol{p}_i,\ 1 \le i \le 3$ is the classical momentum of the decay products.

The value of the modulus-squared of the invariant amplitude $|\mathcal{M}|^2$ is dependent on the force that mediates the interaction. In our example, as well as in most particle decays, the weak force is responsible. If we assume that $m_e = 0.510\,\text{MeV} \approx 0\,\text{GeV}$ we get (see [12, p. 305])

$$|\mathcal{M}|^2 = \left(\frac{g_w}{m_W}\right)^4 m_\mu^2 E_2(m_\mu - 2E_2). \tag{3.3}$$

By inserting equation 3.3 into Fermi's golden rule (equation 3.2) and by selecting as our coordinate system the rest frame of the muon (i.e. $\boldsymbol{p}_A = 0$) we get

$$\Gamma = \int \frac{|\mathcal{M}|^2}{2^4 m_\mu (2\pi)^5} \frac{1}{E_1 E_2 E_3} \delta(\boldsymbol{p}_1 + \boldsymbol{p}_2 + \boldsymbol{p}_3) \delta(E_A - E_1 - E_2 - E_3)\, \mathrm{d}\boldsymbol{p}_1 \mathrm{d}\boldsymbol{p}_2 \mathrm{d}\boldsymbol{p}_2.$$

However, in its present form the integral is not suitable for numerical evaluation (due to the Dirac delta distributions). Therefore, we need to perform the necessary steps to remove them. By converting the integral to spherical coordinates we get (see [12, p. 305-306])

$$\Gamma = \int_0^{\frac{1}{2}m_\mu} \int_{-E_2+\frac{1}{2}m_\mu}^{\frac{1}{2}m_\mu} \int_0^{\pi} \int_0^{2\pi} \frac{|\mathcal{M}|^2}{(4\pi)^4 m_\mu} \sin\theta_2 \, \mathrm{d}\phi_2 \mathrm{d}\theta_2 \mathrm{d}E_4 \mathrm{d}E_2. \tag{3.4}$$

We could perform the integral with respect to θ_2 and ϕ_2 quite easily. However, since we are interested in a numerical solution we stop here. Now, let us compute the integral by the plain Monte Carlo integration method (as described in section 3.1) as well as by using the VEGAS algorithm (as described in section 3.3). First, let us rewrite the integral in equation 3.4 as

$$\Gamma = \int_0^{\frac{1}{2}m_\mu} \int_0^{\frac{1}{2}m_\mu} \int_0^{\pi} \int_0^{2\pi} \chi_{[-E2+\frac{1}{2}m_\mu, \frac{1}{2}m_\mu]}(E_4) \cdot \frac{|\mathcal{M}|^2}{(4\pi)^4 m_\mu} \sin\theta_2 \, \mathrm{d}\phi_2 \mathrm{d}\theta_2 \mathrm{d}E_4 \mathrm{d}E_2, \tag{3.5}$$

which is simpler to handle numerically, since the integration boundaries form a 4-dimensional rectangle. To compare the result of different Monte Carlo integration methods we note that equation 3.4 has a closed-form solution, computed in [12, p. 307], which is given by

$$\Gamma = \left(\frac{m_\mu g_w}{m_W}\right)^4 \frac{m_\mu}{12(8\pi)^3} \approx 3.04226623514192 \cdot 10^{-19}\,\mathrm{GeV}.$$

The implementation (see Listing 5) computes the integral in equation 3.5 by using the following values of N

$$1000,\ 3981,\ 15848,\ 63095,\ 251188,\ 1000000.$$

This corresponds to a uniform distribution of points on a (base 10) logarithmic scale. The VEGAS algorithm uses two iterations of sample size $\lfloor \frac{N}{10} \rfloor$ and one iteration of sample size N. Additionally, $n = 10$ (i.e. 10 bins are used per dimension). The result is shown in Figure 3.1.

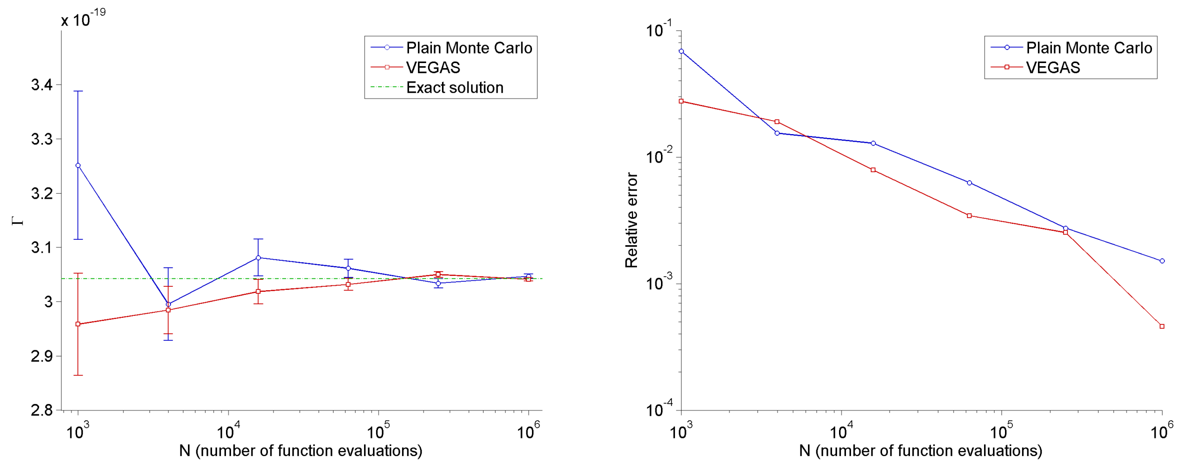

Figure 3.1: Plot of standard deviation (left) and logarithmic error plot (right) of muon decay.

We are (for obvious reasons) especially interested in the behavior of the algorithm for $N = 10^6$. Therefore, results and standard deviations are listed in Table 3.2, whereas the P matrix (matrix of bin arrangement) is listed in Figure 3.2.

Algorithm	Γ	Standard deviation	Relative error
Plain Monte Carlo	$0.3047 \cdot 10^{-18}$	$0.4263 \cdot 10^{-21}$	0.001510
VEGAS	$0.3041 \cdot 10^{-18}$	$0.2794 \cdot 10^{-21}$	0.0004597

Table 3.2: Output of the algorithm in the case $N = 10^6$.

$$P = \begin{bmatrix} 0.17 & 0.26 & 0.34 & 0.42 & 0.49 & 0.57 & 0.65 & 0.73 & 0.83 & 1.00 \\ 0.10 & 0.20 & 0.30 & 0.40 & 0.50 & 0.60 & 0.70 & 0.80 & 0.90 & 1.00 \\ 0.17 & 0.26 & 0.34 & 0.42 & 0.50 & 0.58 & 0.66 & 0.74 & 0.83 & 1.00 \\ 0.21 & 0.32 & 0.41 & 0.50 & 0.58 & 0.67 & 0.75 & 0.83 & 0.92 & 1.00 \end{bmatrix}$$

Figure 3.2: P matrix (matrix of bin arrangement) in the last iteration ($N = 10^6$).

We conclude, that the result is accurate up to three digits (compared to the closed-form solution **not** the experimental result) and that the VEGAS algorithm has a standard deviation (as well as a relative error) that is about half that of the plain Monte Carlo method. Furthermore, the bins are approximately of equal width, i.e. a more narrow peaked function would probably result in an even lower standard deviation for the VEGAS algorithm.

For the sake of completeness let us calculate the mean life time that corresponds to Γ

$$\tau = \frac{\hbar}{\Gamma} = \frac{6.582 \cdot 10^{-25}\,\text{GeV s}}{3.042 \cdot 10^{-19}\,\text{GeV}} = 2.164\,\text{ns}.$$

Compared to the experimental value of 2.197 ns this is in excellent agreement considering the fact that the computation is correct to first order in g_w only. Higher, order calculations would yield more accurate results at the price of greatly increased computational cost.

4 Markov chain Monte Carlo methods

4.1 Introduction to continuous-state Markov chains

In this section the most basic definitions and some useful theorems about continuous-state Markov chains are introduced. Many books/papers on this subject, especially in the more applied fields such as statistics, introduce discrete-state Markov chains and then use some argument to postulate that similar considerations also apply in the continuous case (see e.g. [9] or [11]). This argumentation, for numerical purposes, isn't easily dismissed since it can be argued that no such construct as a continuous state-space exists on a computer anyway. However, since this is a bachelor thesis in the field of mathematics it is my hope that this introduction at least can serve as a starting point for a reader who wants to study this subject in more detail. Many definitions and theorems in this section are taken from [22] (A similar discussion that is more centered on the application to Markov chain Monte Carlo methods can be found in [24]).

At a first step let us define absolute continuity of a measure with respect to another measure.

Definition 4.1. Let $(\Omega, \mathcal{A})$ be a measure space and suppose ν, μ are (positive) measures on $(\Omega, \mathcal{A})$. We say ν is **absolutely continuous with respect to** μ (denoted by $\nu \ll \mu$), if

$$\forall A \in \mathcal{A} \,:\, (\mu(A) = 0) \Longrightarrow (\nu(A) = 0)\,.$$

Additionally, define **equivalence of measures** by $\nu \sim \mu :\Longleftrightarrow (\nu \ll \mu) \wedge (\mu \ll \nu)$.

Theorem 4.2. *Let $\sim$ be defined as above, absolute continuity is a partial order of all measures on $(\Omega, \mathcal{A})$.*

Proof. Antisymmetry is satisfied by construction. To proof reflexivity we note that

$$\forall A \in \mathcal{A} \,:\, (\mu(A) = 0) \Longrightarrow (\mu(A) = 0)\,,$$

which implies $\mu \ll \mu$ as desired. To proof transitivity suppose that $\nu \ll \mu$ and $\mu \ll \eta$. Then

$$\forall A \in \mathcal{A} \, : \, (\eta(A) = 0) \Longrightarrow (\mu(A) = 0) \Longrightarrow (\nu(A) = 0) \, ,$$

which implies $\nu \ll \eta$, as desired. $\square$

Note that for signed measures the definition is slightly more complicated (see e.g. [14, p. 122-124]). However, since we are exclusively interested in probability measures (which are per definition positive) this is no restriction. Next, let us introduce the concept of a Markov chain.

Definition 4.3. Suppose $(\Omega, \mathcal{A}, K)$ is a probability space and let $(\boldsymbol{X}_i)_{i=0}^{\infty}$ be a sequence of random vectors. Let $P : \Omega \times \mathcal{A} \to [0,1]$ be a function such that $P(\boldsymbol{x}, \cdot)$ is a probability distribution for all $\boldsymbol{x} \in \Omega$. If

$$\mathcal{L}(\boldsymbol{X}_{n+1} | \boldsymbol{X}_n = \boldsymbol{x}_n, \dots, \boldsymbol{X}_0 = \boldsymbol{x}_0) = \mathcal{L}(\boldsymbol{X}_{n+1} | \boldsymbol{X}_n = x_n) = P(\boldsymbol{x}_n, \cdot)$$

for all $n \in \mathbb{N}_{\geq 0}$, we call $(\boldsymbol{X}_i)_{i=0}^{\infty}$ a **Markov chain** with **transition probability** P. We set

$$L(\boldsymbol{x}, A) := P \left\{ \exists n \in \mathbb{N}_{\geq 1} \, : \, \boldsymbol{X}_n \in A, \, \boldsymbol{X}_0 = \boldsymbol{x} \right\}$$

for $A \in \mathcal{A}$. Our ultimate goal is to construct a Markov chain that converges towards a specific distribution. If a Markov chain converges at all (under some conditions) its limit is a so-called invariant measure.

Definition 4.4. A measure $\pi : \mathcal{A} \to \mathbb{R}$ is called **invariant** if

$$\pi = \pi P.$$

If π is a probability measure we refer to it as a **steady-state distribution**.

However, to give necessary conditions for convergence towards a unique steady-state distribution we first need to develop the concept of irreducibility.

Definition 4.5. A transition probability P is called **φ-irreducible** if φ is a non-zero measure and

$$\forall \boldsymbol{x} \in \Omega \, \forall A \in \mathcal{A}, \varphi(A) > 0 \, : \, L(\boldsymbol{x}, A) > 0.$$

That the definition of irreducibility depends on the measure φ is not a desirable attribute. Therefore, we introduce the concept of a maximal irreducibility measure (with respect to the partial ordering $\ll$). This (unique) maximal irreducibility measure is then used in the following definitions and theorems.

Lemma 4.6. *Suppose that P is φ-irreducible. Then, there exists a unique maximal irreducibility measure (maximal with respect to the partial ordering $\ll$).*

Proof. The following proof is based on [23]. We proof the statement by transfinite induction. Suppose $(\mu_i)_{i=1}^{\infty}$ is a totally ordered subset (chain) of all irreducible measures. Then, we define the upper bound μ by

$$\forall A \in \mathcal{A} \, : \, \mu(A) := \sup_{i \in \mathbb{N}} \mu_i(A).$$

If μ is a measure, $\mu(A) = 0$ implies $\mu_i(A) = 0$, $\forall i \in \mathbb{N}$. Therefore, $\mu_1 \ll \mu_2 \ll \cdots \ll \mu$. That is, μ is a maximal irreducibility measure by Zorn's lemma.

We still have to proof that μ is in fact a measure. First,

$$\mu(\emptyset) = \sup_{i \in \mathbb{N}} \mu_i(\emptyset) = 0.$$

Second, suppose $A \subset B$ where $A, B \in \mathcal{A}$, then (since $\mu_i(A) \leq \mu_i(B)$, $\forall i \in \mathbb{N}$)

$$\mu(A) \leq \mu(B).$$

Third, $(A_j)_{j=1}^{\infty}$ where $A_j \in \mathcal{A}$ and $A_j \cap A_k = \emptyset$ for $j \neq k$. Then

$$\mu\left(\bigcup_{j=1}^{\infty} A_j\right) = \sup_{i\in\mathbb{N}} \mu_i\left(\bigcup_{j=1}^{\infty} A_j\right) \leq \sup_{i\in\mathbb{N}} \sum_{j=1}^{\infty} \mu_i(A_j) \leq \sum_{j=1}^{\infty} \sup_{i\in\mathbb{N}} \mu_i(A_j) = \sum_{j=1}^{\infty} \mu(A_j).$$

Therefore μ is a measure, as desired.

To show uniqueness we assume that ν and μ are maximal irreducibility measures. Then $\nu \ll \mu$ and $\mu \ll \nu$ which implies $\mu \sim \nu$. □

In the case of a discrete state-space Markov chain the above discussion can be omitted. It is possible to define recurrence directly by stating that if the Markov chain is in state i the probability to return to that same state in finite time must be 1 (Often, this is expressed by saying that the state i has a finite hitting time with probability 1). If this is the case for all states we say that the Markov chain is recurrent.

However, in the continuous case we need the concept of a maximal irreducibility measure to define Harris recurrence (which can be seen as the continuous equivalent to recurrence).

Definition 4.7. A φ-irreducible Markov chain (for some φ) is called **Harris recurrent**, if

$$\forall A \in \mathcal{A}, \psi(A) > 0 \,:\, P\{\boldsymbol{X}_n \in A, \text{ infinitely often} \,|\, \boldsymbol{X}_0 = \boldsymbol{x}\} = 1,$$

where ψ is the unique maximal irreducibility measure.

In many instances, it can easily be shown that a Markov chain is π-irreducible (i.e., irreducible with respect to the invariant measure). In this case we can formulate an equivalent condition for Harris recurrent that is commonly used in the literature (e.g. [27, 25]).

Theorem 4.8. *A π-irreducible Markov chain is Harris recurrent iff*

$$\forall A \in \mathcal{A}, \pi(A) > 0 \,:\, P\{\boldsymbol{X}_n \in A, \text{ infinitely often} \,|\, \boldsymbol{X}_0 = \boldsymbol{x}\} = 1.$$

Proof. We prove that if a Markov chain is π-irreducible it holds that $\psi \sim \pi$, where ψ is the maximal irreducibility measure. Let us assume $\psi \not\sim \pi$. Then, there exists $A \in \mathcal{A}$ such that $\psi(A) > 0$ but $\pi(A) = 0$. The irreducibility of ψ implies that $L(\boldsymbol{x}, A) > 0, \forall \boldsymbol{x} \in \Omega$ and therefore that $L(\pi, A) := P\{\exists n \in \mathbb{N}_{\geq 1} : \boldsymbol{X}_n \in A, \boldsymbol{X}_0 \sim \pi\} > 0$. However, since $\pi(A) = \pi P^n(A), \forall n \in \mathbb{N}$ we have $L(\pi, A) = 0$ which is a contradiction. □

We will need one additional theorem that gives conditions for Harris recurrence that are easily verified in many applications.

Theorem 4.9. *Let $(\boldsymbol{X}_i)_{i=0}^{\infty}$ be a π-irreducible Markov chain with transition probability P and assume $P(\boldsymbol{x}, \cdot) \ll \pi$ for all $\boldsymbol{x} \in \Omega$. Then $(\boldsymbol{X}_i)_{i=0}^{\infty}$ is a Harris recurrent Markov chain.*

Proof. See e.g. [27, p. 1712-1713]. □

The next definition is a strengthening of Harris recurrence which clearly is a necessary condition if our Markov chain converges to a probability distribution. If the condition $\pi(\Omega) < \infty$ is not met, there is no way to normalize the invariant measure to a probability distribution.

Definition 4.10. An irreducible Markov chain is called **positive Harris recurrent**, if it is Harris recurrent and there exists an invariant measure π such that

$$\pi(\Omega) < \infty.$$

The next condition is the obvious extension from the discrete case. To ensure convergence it is necessary to exclude Markov chains with periodic states (i.e. a Markov chain where if we start from a state in $\mathcal{A}$ we can only return to the same state after k steps where $k > 1$).

Definition 4.11. The **period** of an irreducible Markov chain with transition probability P is the largest $d \in \mathbb{N}$ such that there exist disjoint sets $\mathcal{X}_0, \dots, \mathcal{X}_{d-1} \in \mathcal{A}$, $\psi(\mathcal{X}_i) > 0$ for which

$$\forall \boldsymbol{x} \in \mathcal{X}_i \, \forall i \in \{0, \dots, d-1\} \, : \, P(\boldsymbol{x}, \mathcal{X}_{i+1 \, (\mathrm{mod} \ d)}) = 1,$$

where ψ is the maximal irreducibility measure. If $d = 1$ the Markov chain is called **aperiodic**.

Theorem 4.12. *An irreducible Markov chain is* ***aperiodic*** *if*

$$\forall A \in \mathcal{A} \, \forall \boldsymbol{x} \in \Omega \, \forall n \in \mathbb{N} \, : \, \psi(A) > 0 \Longrightarrow P\left\{\boldsymbol{X}_n \in A \,|\, \boldsymbol{X}_0 = \boldsymbol{x}\right\} > 0,$$

where ψ is the maximal irreducibility measure.

Proof. Assume $d > 1$ and $\mathcal{X}_1, \dots, \mathcal{X}_{d-1}$ as in Definition 4.11. Then for $\boldsymbol{x} \in \mathcal{X}_1$ we have $P(\boldsymbol{x}, \mathcal{X}_2) \leq 1 - P\left\{\boldsymbol{X}_1 \in \mathcal{X}_1 \,|\, \boldsymbol{X}_0 = \boldsymbol{x}\right\} < 1$ which is a contradiction. □

Now, we are finally in the position to state the theorem about the convergence of Markov chains that we need to develop the Metropolis-Hastings algorithm in the next section. Note, that the requirements are exactly those we developed in this section, namely positive Harris recurrence and aperiodicity. Since these conditions imply the convergence to a unique steady-state distribution, we refer to it as an ergodic Markov chain (or Harris ergodic if we want to emphasize the continuous state-space).

Theorem 4.13. *Let* $(\boldsymbol{X}_i)_{i=0}^{\infty}$ be an aperiodic and positive Harris recurrent Markov chain (also called a **(Harris) ergodic** Markov chain) and suppose λ, μ are probability distributions. Then

$$\lim_{n \to \infty} \|\lambda P^n - \mu P^n\| = 0.$$

Proof. See [22, p. 112-113]. □

Corollary 4.14. *Let* $(\boldsymbol{X}_i)_{i=0}^{\infty}$ be a (Harris) ergodic Markov chain. Then for any initial distribution λ and a steady-state distribution π it holds that

$$\lambda P^n \xrightarrow{\mathrm{D}} \pi, \qquad \text{as } n \to \infty$$

where π is unique.

Proof. Choose π as an initial distribution. Then from Theorem 4.13 we have

$$0 = \lim_{n \to \infty} \|\lambda P^n - \pi P^n\| = \lim_{n \to \infty} \|\lambda P^n - \pi\|,$$

as desired. To prove uniqueness suppose π and ϱ are steady-state distributions of our Markov chain. Then by Theorem 4.13

$$0 = \lim_{n \to \infty} \|\pi P^n - \varrho P^n\| = \lim_{n \to \infty} \|\pi - \rho\| = \|\pi - \varrho\|,$$

as desired. □

From a mathematical standpoint it is quite noteworthy that the reverse of this result also holds. However, since this is of no relevance to the development in the next section the interested reader is referred to [22, p. 114].

What is still missing from our picture are sufficient conditions for a Markov chain to have a specific steady-state distribution π. The next theorem states the condition of detailed balance which is used in the Metropolis-Hastings algorithm (to be developed in the next section).

Theorem 4.15. Let $(\boldsymbol{X}_i)_{i=0}^{\infty}$ be a Markov chain with transition probability P and suppose $\psi(\boldsymbol{x}, \cdot)$ is the probability density function of $P(\boldsymbol{x}, \cdot)$. If χ is a probability density function and

$$\forall \boldsymbol{x}, \boldsymbol{y} \in \Omega \, : \, \psi(\boldsymbol{x}, \boldsymbol{y})\chi(\boldsymbol{x}) = \psi(\boldsymbol{y}, \boldsymbol{x})\chi(\boldsymbol{y}),$$

then χ is the pdf of a steady-state distribution (This condition is referred to as **detailed balance**).

Proof.

$$\begin{aligned}\int_\Omega \psi(\boldsymbol{x},\boldsymbol{y})\chi(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} &= \chi(\boldsymbol{y})\int_\Omega \psi(\boldsymbol{y},\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}\\ &= \chi(\boldsymbol{y}),\end{aligned}$$

as desired, since if π is the probability distribution attached to the pdf χ our result implies $\pi P = \pi$. □

4.2 The Metropolis-Hastings algorithm

The Metropolis-Hastings algorithm is an application of the principles developed in the previous section; i.e., a Markov chain is constructed that has a specific probability distribution as its steady-state distribution. This section is mainly based on [15] and [9].

Algorithm 4.16. ***(Metropolis-Hastings).*** *Let $(\boldsymbol{X}_i)_{i=0}^\infty$ be a Markov chain with transition probability Q and suppose f is a pdf. Let $\psi(\boldsymbol{x},\cdot)$ be the probability density function of $Q(\boldsymbol{x},\cdot)$. Choose an initial value $\boldsymbol{y}_0$ and proceed as follows*

1. Generate a proposal value $\boldsymbol{x}'$ from the probability distribution $Q(\boldsymbol{y}_{i-1},\cdot)$.
2. Calculate $r := \min\left\{1, \frac{\psi(\boldsymbol{x}',\boldsymbol{y}_{i-1})f(\boldsymbol{x}')}{\psi(\boldsymbol{y}_{i-1},\boldsymbol{x}')f(\boldsymbol{y}_{i-1})}\right\}$,
3. Set
$$\boldsymbol{y}_i := \begin{cases}\boldsymbol{x}' & \text{with probability } r\\ \boldsymbol{y}_{i-1} & \text{with probability } 1-r\ ,\end{cases}$$

Then the probability distribution attached to f is a steady-state distribution of $(\boldsymbol{Y}_i)_{i=0}^\infty$.

Proof. The pdf attached to Q is given by

$$p(\boldsymbol{x},\boldsymbol{y}) = r(\boldsymbol{x},\boldsymbol{y})\psi(\boldsymbol{x},\boldsymbol{y}) + \left(1 - \int_\Omega r(\boldsymbol{x},\boldsymbol{y})\psi(\boldsymbol{x},\boldsymbol{y})\,\mathrm{d}\boldsymbol{y}\right)\delta(\boldsymbol{y}-\boldsymbol{x}).$$

To verify that f is a steady-state distribution we confirm the detailed balance condition. First,

$$\begin{aligned}r(\boldsymbol{x},\boldsymbol{y})\psi(\boldsymbol{x},\boldsymbol{y})f(\boldsymbol{x}) &= \min\left\{1, \frac{\psi(\boldsymbol{y},\boldsymbol{x})f(\boldsymbol{y})}{\psi(\boldsymbol{x},\boldsymbol{y})f(\boldsymbol{x})}\right\}\psi(\boldsymbol{x},\boldsymbol{y})f(\boldsymbol{x})\\ &= \min\left\{\psi(\boldsymbol{x},\boldsymbol{y})f(\boldsymbol{x}), \psi(\boldsymbol{y},\boldsymbol{x})f(\boldsymbol{y})\right\}\\ &= \min\left\{\psi(\boldsymbol{y},\boldsymbol{x})f(\boldsymbol{y}), \psi(\boldsymbol{x},\boldsymbol{y})f(\boldsymbol{x})\right\}\\ &= \min\left\{1, \frac{\psi(\boldsymbol{x},\boldsymbol{y})f(\boldsymbol{x})}{\psi(\boldsymbol{y},\boldsymbol{x})f(\boldsymbol{y})}\right\}\psi(\boldsymbol{y},\boldsymbol{x})f(\boldsymbol{y})\\ &= r(\boldsymbol{y},\boldsymbol{x})\psi(\boldsymbol{y},\boldsymbol{x})f(\boldsymbol{y}).\end{aligned}$$

Second,

$$\left(1 - \int_\Omega r(\boldsymbol{x},\boldsymbol{z})\psi(\boldsymbol{x},\boldsymbol{z})\,\mathrm{d}\boldsymbol{z}\right)\delta(\boldsymbol{y}-\boldsymbol{x})f(\boldsymbol{x}) = \left(1 - \int_\Omega r(\boldsymbol{y},\boldsymbol{z})\psi(\boldsymbol{y},\boldsymbol{z})\,\mathrm{d}\boldsymbol{x}\right)\delta(\boldsymbol{x}-\boldsymbol{y})f(\boldsymbol{y}),$$

since by the definition of a (mathematical) distribution (if $\Phi \in \mathcal{C}_0^\infty$ is a test function with compact support) we have

$$\begin{aligned}&\int_\Omega\int_\Omega \left(1 - \int_\Omega r(\boldsymbol{x},\boldsymbol{z})\psi(\boldsymbol{x},\boldsymbol{z})\,\mathrm{d}\boldsymbol{z}\right)\delta(\boldsymbol{y}-\boldsymbol{x})f(\boldsymbol{x})\Phi(\boldsymbol{x},\boldsymbol{y})\,\mathrm{d}\boldsymbol{x}\mathrm{d}\boldsymbol{y}\\ &= \int_\Omega \left(1 - \int_\Omega r(\boldsymbol{y},\boldsymbol{z})\psi(\boldsymbol{y},\boldsymbol{z})\,\mathrm{d}\boldsymbol{z}\right)f(\boldsymbol{y})\Phi(\boldsymbol{y},\boldsymbol{y})\,\mathrm{d}\boldsymbol{y}\\ &= \int_\Omega \left(1 - \int_\Omega r(\boldsymbol{x},\boldsymbol{z})\psi(\boldsymbol{x},\boldsymbol{z})\,\mathrm{d}\boldsymbol{z}\right)f(\boldsymbol{x})\Phi(\boldsymbol{x},\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}\\ &= \int_\Omega \left(1 - \int_\Omega r(\boldsymbol{y},\boldsymbol{z})\psi(\boldsymbol{y},\boldsymbol{z})\,\mathrm{d}\boldsymbol{x}\right)\delta(\boldsymbol{x}-\boldsymbol{y})f(\boldsymbol{y})\Phi(\boldsymbol{x},\boldsymbol{y})\,\mathrm{d}\boldsymbol{x}\mathrm{d}\boldsymbol{y},\end{aligned}$$

as desired. □

It should be noted that in order to compute r we only need to know the ratio $f(\boldsymbol{x}')/f(\boldsymbol{x})$. Therefore, it is sufficient to know a function $g(\boldsymbol{x})$ that satisfies $g(\boldsymbol{x}) = Cf(\boldsymbol{x})$. That is, we don't need to compute the normalization constant of a probability density function to sample from it. This fact is employed in most applications of the Metropolis-Hastings algorithm (see Example 4.21 or 4.22).

In the Metropolis-Hastings algorithm we left the transition probability (up to the requirement that the Markov Chain is (Harris) ergodic) completely undetermined. Theoretically every transition probability that satisfies this requirement would suffice. However, in application the following two methods (due to Metropolis and Hastings respectively) are the most common. It is clear that those two algorithms are a special case of the more general Metropolis-Hastings algorithm.

First, let us discuss the **Metropolis algorithm**. In this case we impose a symmetry condition on ψ. That is

$$\forall \boldsymbol{x}, \boldsymbol{x}' \in \Omega \, : \, \psi(\boldsymbol{x}, \boldsymbol{x}') = \psi(\boldsymbol{x}', \boldsymbol{x}).$$

Then

$$r(\boldsymbol{x}, \boldsymbol{x}') = \min\left\{1, \frac{f(\boldsymbol{x}')}{f(\boldsymbol{x})}\right\}.$$

Often, a symmetric random walk is used, i.e.

$$\psi(\boldsymbol{x}, \boldsymbol{x}') = p_D(\boldsymbol{x} - \boldsymbol{x}'),$$

where $D \sim \mathcal{N}(0, \Sigma)$ and Σ is a diagonal matrix with $\operatorname{diag}\Sigma = (\sigma_1, \dots, \sigma_d)$ (we consider an uncorrelated multivariate normal distribution). In addition, p_D denotes the pdf of the random variable D.

The parameters σ_i need to be determined. If we choose σ_i too large we have a very low acceptance rate. On the other hand, if σ is too small we have a very high acceptance rate. It has been shown in [10] that for a symmetric random walk the optimal acceptance rate in the asymptotic case (i.e. $d \to \infty$) is 0.234. However, from the examples in this bachelor thesis we infer that an acceptance rate between 0.2 and 0.8 leads to acceptable results.

In some implementations the algorithm uses the so called burn-in phase to adjust this parameter. A burn-in phase, a number of steps in which we generate new values from the Markov chain but do not sample from it, is necessary since the Markov chain converges to the desired distribution. Thus, some time is needed (from an arbitrary starting point) until the Markov chain approximates the steady-state distribution to some accuracy.

Second, let us discuss the **Hastings algorithm**, which lifts the restriction of symmetry and imposes instead

$$\psi(\boldsymbol{x}, \boldsymbol{x}') = \psi(\boldsymbol{x}').$$

Then

$$r(\boldsymbol{x}, \boldsymbol{x}') = \min\left\{1, \frac{\psi(\boldsymbol{x})f(\boldsymbol{x}')}{\psi(\boldsymbol{x}')f(\boldsymbol{x})}\right\}.$$

This method is also referred to as **independence sampling**, since the proposal value is generated independently of $\boldsymbol{y}_{n-1}$. However, it should be **emphasized** that both algorithms produce a Markov chain with **dependent** samples. If it is desirable to sample independently from the probability distribution of f we sample only the values $\boldsymbol{y}_i, \boldsymbol{y}_{i+K}, \boldsymbol{y}_{i+2K}, \dots$. If $K \in \mathbb{N}$ is chosen large enough the sample is (to an excellent approximation) independent.

In the examples that follow we will exclusively use the Metropolis algorithm as described above. An implementation of the Metropolis algorithm can be found in Listing 4. Therefore, all that remains is to proof convergence of the Metropolis algorithm.

Theorem 4.17. *The Metropolis algorithm (as described above) converges (in distribution) to the probability distribution attached to f if $f(\boldsymbol{x}) > 0$ for every $\boldsymbol{x} \in \Omega \subset \mathbb{R}^n$.*

Proof. Since , $f(\boldsymbol{x}) > 0$ for all $\boldsymbol{x} \in \Omega$

$$\left(\pi(A) := \int_A f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} = 0\right) \Longleftrightarrow (\mu(A) = 0)\,,$$

where μ denotes the Lebesgue measure. Then

$$r(\boldsymbol{x}, \boldsymbol{x}') = \min\left\{1, \frac{f(\boldsymbol{x}')}{f(\boldsymbol{x})}\right\} > 0,$$

since $f(\boldsymbol{x}) > 0$ for all $\boldsymbol{x} \in \Omega$ and therefore

$$\forall \boldsymbol{x} \in \Omega,\, \forall A \in \mathcal{A},\, \mu(A) > 0 \,:\, P(\boldsymbol{x}, A) = \int_A p(\boldsymbol{x}, \boldsymbol{y})\,\mathrm{d}\boldsymbol{y} \geq \int_A r(\boldsymbol{x}, \boldsymbol{y})\psi(\boldsymbol{x}, \boldsymbol{y})\,\mathrm{d}\boldsymbol{y} > 0,$$

since by assumption $\psi(\boldsymbol{x}, \boldsymbol{y}) > 0$. This implies the π-irreducibility of $(\boldsymbol{Y}_i)_{i=0}^{\infty}$. Next,

$$\forall \boldsymbol{x} \in \Omega \,:\, (\mu(A) = 0) \Longrightarrow \left(\int_A p_D(\boldsymbol{x} - \boldsymbol{x}')\,\mathrm{d}\boldsymbol{x}'\right) = 0,$$

where $p_D(\boldsymbol{x} - \boldsymbol{x}')$ is the pdf of a multivariate normal distribution with mean 0 and VC matrix Σ. This implies $P(\boldsymbol{x}, \cdot) \ll \pi$. Thus by Theorem 4.9 $(\boldsymbol{Y}_i)_{i=0}^{\infty}$ is Harris recurrent. Now, since f is a probability density function $\pi(\Omega) = 1 < \infty$; therefore, $(\boldsymbol{Y}_i)_{i=0}^{\infty}$ is positive Harris recurrent.

Since

$$\forall \boldsymbol{x} \in \Omega,\, \forall A \in \mathcal{A},\, \mu(A) > 0 \,:\, P\{\boldsymbol{X}_n \in A \,|\, \boldsymbol{X}_{n-1} = \boldsymbol{x}\} > 0,$$

aperiodicity follows. Therefore $(\boldsymbol{Y}_i)_{i=0}^{\infty}$ is (Harris) ergodic and converges in distribution to π (by Theorem 4.13). □

4.3 Applications

An important problem in Bayesian statistics is to compute a posterior distribution by using Bayes' theorem. More formally (see e.g. [16, p. 31])

Theorem 4.18. ***(Bayes' theorem).*** *Let $\boldsymbol{X}$ be a random vector with probability density $p_{\boldsymbol{X}} : \Omega \to \mathbb{R}$* ***(the prior)*** *and suppose $\boldsymbol{Y}$ is a random vector. Let us denote by $p_{\boldsymbol{Y}}(\boldsymbol{y}|\boldsymbol{X} = \boldsymbol{x})$ the probability density of a specific realization $\boldsymbol{y}$ of the random vector $\boldsymbol{Y}$ under the condition that $p_{\boldsymbol{X}}(\boldsymbol{\xi}) = \delta(\boldsymbol{x} - \boldsymbol{\xi})$ ($p_{\boldsymbol{Y}}(\boldsymbol{y}|\boldsymbol{X} = \boldsymbol{x})$ is also referred to as the* ***likelihood function****) . Then*

$$p_{\boldsymbol{X}}(\boldsymbol{x}|\boldsymbol{Y} = \boldsymbol{y}) = \frac{p_{\boldsymbol{Y}}(\boldsymbol{y}|\boldsymbol{X} = \boldsymbol{x})p_{\boldsymbol{X}}(\boldsymbol{x})}{\int_\Omega p_{\boldsymbol{Y}}(\boldsymbol{y}|\boldsymbol{X} = \boldsymbol{x})p_{\boldsymbol{X}}(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}}, \tag{4.1}$$

where $p_{\boldsymbol{X}}(\boldsymbol{x}|\boldsymbol{Y} = \boldsymbol{y})$ is called the ***posterior distribution****.*

Note that Bayesian statistics has a different interpretation of probability than classical statistics. Often, $\boldsymbol{X}$ will be a vector of parameters that have a fixed (but unknown) value. In Bayesian statistics we assign a probability distribution to the parameters as a measure of how certain **we** are that the parameters have a specific value. This is clearly forbidden in classical statistics where one can, at most, construct confidence intervals of parameter values. However, it is important to note that even in Bayesian statistics we don't imply that $\boldsymbol{X}$ is fundamentally random, instead it is implied that we can not be sure about the true value of the parameters.

The dimensionality of the integral in equation 4.1 is equal to the number of parameters in our random vector $\boldsymbol{X}$. However, by using the Metropolis algorithm we can sample from the posterior distribution without carrying out the integration. Once, we have created a large enough sample from the posterior distribution we can estimate it (e.g. by means of histograms) or compute its mean, standard deviation, ...

To exemplify this we introduce the autoregressive model of order 2 (which is, e.g., used in the field of economics, see [8, p. 270-273]).

Definition 4.19. Let $(Y_i)_{i=1}^n$ be a **time series** (or in mathematical terms a finite sequence of random variables). The **autoregressive model** is given by

$$Y_i = A_i(\boldsymbol{x}) := \beta_0 + \beta_1 y_{i-1} + \beta_2 y_{i-2} + E_i,$$

where $E_i \sim \mathcal{N}(0, \sigma)$ i.i.d. and $A_i(\boldsymbol{x}) = A_i([\sigma, \beta_0, \beta_1, \beta_2]^T)$.

That is, we assume that our time series can be modeled as a linear recurrence relation of order 2 with the addition of a normally distributed error term. The model has four parameters $(\sigma, \beta_0, \beta_1, \beta_2)$, which we regard as the components of the random vector $\boldsymbol{X}$.

Since, on many occasions, there is no prior information available (and in particular this will be the assumption we make in the two examples that follow), we choose a uniform distribution as the prior. This uniform distribution, in turn, can be neglected, since we only need to know the probability density function up to a normalization constant.

Therefore, the important ingredient is the likelihood function which is given by

$$p_{\boldsymbol{Y}}\left(\boldsymbol{y} \middle| \boldsymbol{X} = [\sigma, \beta_0, \beta_1, \beta_2]^T\right) = \prod_{i=3}^{n} p_{\mathcal{N}(0,\sigma)}\left(\beta_0 + \beta_1 y_{i-1} + \beta_2 y_{i-2} - y_i\right), \tag{4.2}$$

where $\boldsymbol{y}$ is the realization of our time series and $p_{\mathcal{N}(0,\sigma)}$ is the pdf of a normally distributed random variable with mean 0 and standard deviation σ. To implement the Metropolis algorithm we need to compute

$$r = \frac{p_{\boldsymbol{Y}}\left(\boldsymbol{y} \middle| \boldsymbol{X} = [\sigma^{(1)}, \beta_0^{(1)}, \beta_1^{(1)}, \beta_2^{(1)}]^T\right)}{p_{\boldsymbol{Y}}\left(\mathbf{y} \middle| \boldsymbol{X} = [\sigma^{(2)}, \beta_0^{(2)}, \beta_1^{(2)}, \beta_2^{(2)}]^T\right)}.$$

However, numerically the product in equation 4.2 can become very small or even zero (to the accuracy of the floating point computations) if n is large. To circumvent this problem (which leads to undesirable results in the numerical computation, e.g. $\min(1, 0/0) = 1$) we compute the product term by term, i.e.

$$r = \prod_{i=3}^{n} \frac{p_{\mathcal{N}(0,\sigma^{(1)})}\left(\beta_0^{(1)} + \beta_1^{(1)} y_{i-1} + \beta_2^{(1)} y_{i-2} - y_i\right)}{p_{\mathcal{N}(0,\sigma^{(2)})}\left(\beta_0^{(2)} + \beta_1^{(2)} y_{i-1} + \beta_2^{(2)} y_{i-2} - y_i\right)}.$$

Now we are in a position to sample from the probability distributions of σ, β_0, β_1, β_2. However, our goal is to estimate future values in the time series (i.e. to estimate y_{n+1} or values thereafter). Thus, we integrate over all possible values of $x = [\sigma, \beta_0, \beta_1, \beta_2]^T$ weighted by the posterior probability. Therefore,

$$y_{n+1} = \int_{\Omega} A_n(\boldsymbol{x}) \cdot p_{\boldsymbol{X}}(\boldsymbol{x} | \boldsymbol{Y} = \boldsymbol{y}) \, \mathrm{d}\boldsymbol{x}. \tag{4.3}$$

However, we only have a sample of the distribution attached to the pdf $p_{\boldsymbol{X}}(\boldsymbol{x}|\boldsymbol{Y} = \boldsymbol{y})$. Therefore, we estimate equation 4.3 by

$$y_{n+1} \approx \sum_{i=1}^{N} A(x_i),$$

where x_i, $1 \leq i \leq N$ are the values sampled by means of the Metropolis algorithm. Now, let us turn our attention to two examples that use the autoregressive model of order 2 as described above.

Example 4.20. This example is rather artificial in that we create a time series (with $n = 100$) that exactly satisfies our model.

$$y_i = \begin{cases} 0 & i = 1 \\ 1 & i = 2 \\ 0.2 + 1.1 y_{i-1} - 0.1 y_{i-2} + \epsilon_i & 3 \leq i \leq 100 \end{cases},$$

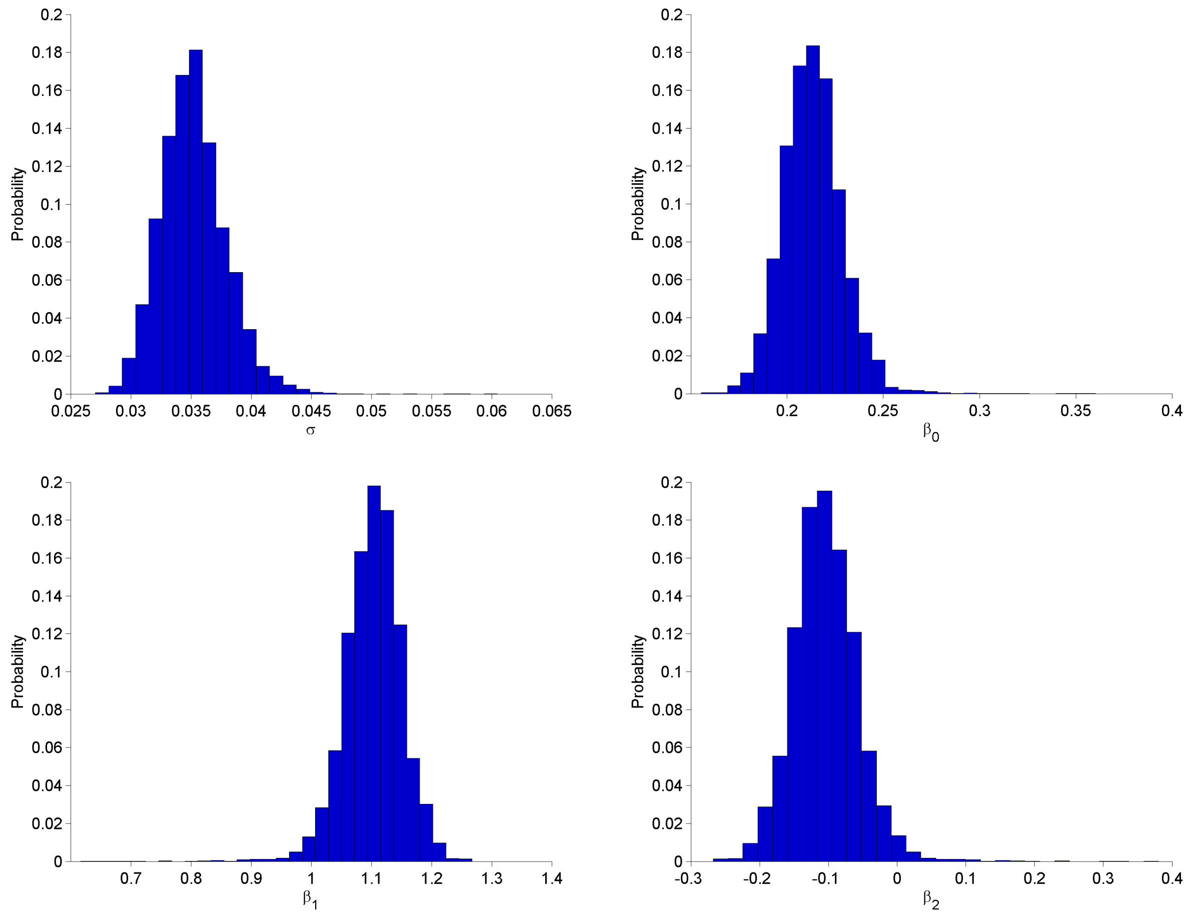

Figure 4.1: Histogram estimation of the parameters.

where ϵ_i is a realization of the random variable $E_i \sim \mathcal{N}(0, 0.03)$. Let us choose a sample size of $N = 5 \cdot 10^5$. The implementation of this procedure can be found in Listing 6.

Figure 4.1 gives an estimation of the distribution of the parameters by means of a histogram. The estimate of mean and standard deviation as well as an estimate of y_{101} can be found in Table 4.1.

This demonstrates the plausibility of our approach. We recovered all parameters to some accuracy and $y_{101} \approx 23.5671$ is well within the theoretical value of

$$\begin{aligned} y_{101} &= \beta_0 + \beta_1 y_{100} + \beta_2 y_{99} + E_{101} \\ &= 0.2 + 1.1 \cdot y_{100} - 0.1 y_{99} + E_{101} \\ &= 23.5718 + E_{101}. \end{aligned}$$

This concludes our (rather artificial) example.

Example 4.21. The analysis done in Example 4.20 can be extended to time series which only approximately satisfy our model assumptions. If we, for example, take the quarterly GDP of the United States in the period between Q1 2004 and Q2 2009 (measured in a system of units where one unit of money is equal to 10^{13}\$). The next value is given by

$$y_{2009Q3} = 1.43015 \cdot 10^{13}.$$

All data have been retrieved from [3]. In this instance it is not clear if the time series can be written as an autoregressive model of order 2. However, we will, in what follows, apply the same analysis as in the

Parameter	Mean/Value	Standard deviation
σ	0.0352	0.002668
β_0	0.2133	0.01525
β_1	1.103	0.0475
β_2	-0.1038	0.04732
y_{101}	23.5671	0.0352

Table 4.1: Parameters and projection of y_{101}.

case of the autoregressive model in the hope that the standard deviation is small enough for the model to be useful (Of course, this formally would invalidate extrapolation but since we are only interested in the next value it is plausible to still get a good approximation).

The implementation of this example can be found in Listing 6 (where once again $N = 5 \cdot 10^5$). The result is given in Figure 4.2 and Table 4.2.

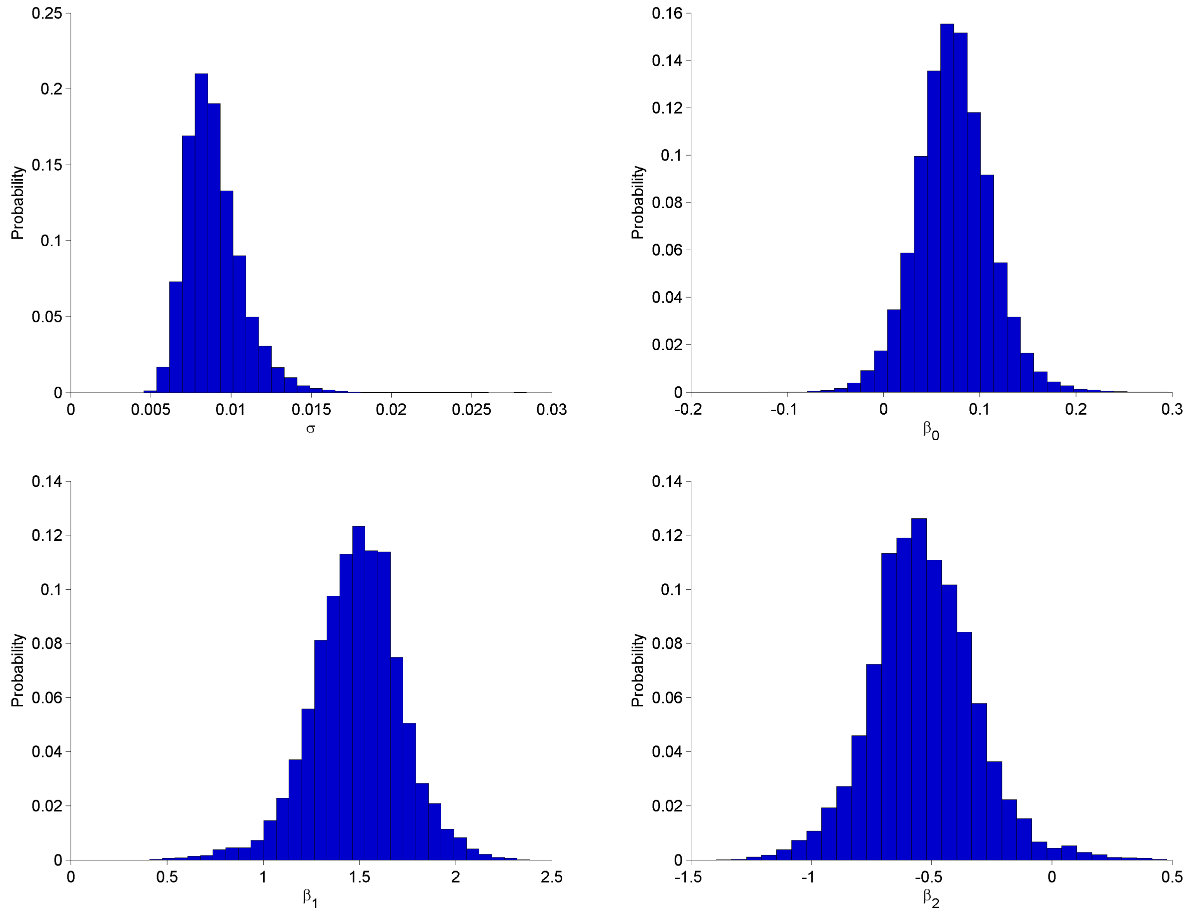

Figure 4.2: Histogram estimation of the parameters.

Comparing this to the experimental value we find an absolute error of approximately 0.015. This (at least) is of the same order of magnitude as the mean of the standard deviation. Depending on the application this might well be an intolerable large error; however, it should be noted that the model used is only of order 2. A model of higher order, say order 10, could be better suited to tackle this problem as could be other models (e.g. moving averages, a combination of moving averages and the autoregressive model, see e.g. [8, Chap. 13]). In conclusion, the Metropolis algorithm can be a viable tool in virtually any application of model fitting that employs Bayes' theorem.

Parameter	Mean/Value	Standard deviation
σ	$0.8920 \cdot 10^{-2}$	$0.1725 \cdot 10^{-2}$
β_0	0.07258	0.03796
β_1	1.4846	0.2317
β_2	-0.5350	0.2158
y_{2009Q3}	1.41496	$0.8920 \cdot 10^{-2}$

Table 4.2: Parameters and projection of y_{2009Q3}.

Example 4.22. (Variational quantum mechanics of the helium atom).

In the field of quantum mechanics (and/or chemistry) an important task is to compute the ground state energy of an atom (i.e. the energy of the state that binds the electrons most tightly). In the case of the hydrogen atom the problem has a closed-form solution and the well-known ground state energy is $-13.6\,\mathrm{eV}$. However, no closed-form solution is known for the helium atom [13, 262] (The same is obviously true for most atoms with more than a single electron). Traditional numerical methods to solve partial differential equations might not be applicable either, since, for instance, the iron atom has 26 electrons which would require us to solve a partial differential equation in $3 \cdot 26 = 78$ dimensions (This is, if at all feasible, certainly computationally extremely expensive).

However, it is possible to compute an excellent approximation to the ground state energy, by means of a method called variational quantum mechanics (or in our case the variational quantum Monte Carlo method), without solving the Schrödinger equation. We will demonstrate this method for the, in comparison, simple case of the helium atom. This treatment is based on the variational method as described in [7, Chap. 16].

As explained in more detail in Example 3.6 we use natural units for the remainder of this example. In what follows, we need the numerical values of the physical constants listed in Table 4.3 (all digits shown are experimentally significant).

Description	Symbol	Value
Mass of the electron	m	$0.511\,\mathrm{MeV}$
fine structure constant (electromagnetic coupling constant)	α	$\frac{1}{137}$
Bohr radius	a_0	$\frac{1}{m\alpha}$
Experimental value of the ground state energy of helium	E_0	$-78.98\,\mathrm{eV}$

Table 4.3: Physical constants (taken from [4]).

The helium atom consist of two protons and two electrons and is, to an excellent approximation, described by the following Schrödinger equation

$$i\frac{\partial \psi}{\partial t} = \frac{-1}{2m}\left(\Delta_{\boldsymbol{r}^{(1)}} + \Delta_{\boldsymbol{r}^{(2)}}\right)\psi - \left(\frac{2\alpha}{\|\boldsymbol{r}^{(1)}\|} + \frac{2\alpha}{\|\boldsymbol{r}^{(2)}\|} - \frac{\alpha}{\|\boldsymbol{r}^{(2)} - \boldsymbol{r}^{(1)}\|}\right)\psi =: H\psi,$$

where H is the Hamiltonian operator, $\psi(\boldsymbol{r}^{(1)}, \boldsymbol{r}^{(2)}, t)$ is the wavefunction and $\boldsymbol{r}^{(1)}, \boldsymbol{r}^{(2)}$ represent the "position" of the first and second electron respectively.

Now, suppose $\psi_G(\boldsymbol{r}) := \psi_G(\boldsymbol{r}^{(1)}, \boldsymbol{r}^{(2)})$ is the wavefunction of the ground state and let $\psi_T(\boldsymbol{r})$ be any (physical) wavefunction. Then

$$E_0 = \frac{\int_{\mathbb{R}^6} \psi_G^*(\boldsymbol{r}) H \psi_G(\boldsymbol{r})\, d\boldsymbol{r}}{\int_{\mathbb{R}^6} \psi_G^*(\boldsymbol{r}) \psi_G(\boldsymbol{r})\, \mathrm{d}\boldsymbol{r}} \leq \frac{\int_{\mathbb{R}^6} \psi_T^*(\boldsymbol{r}) H \psi_T(\boldsymbol{r})\, d\boldsymbol{r}}{\int_{\mathbb{R}^6} \psi_T^*(\boldsymbol{r}) \psi_T(\boldsymbol{r})\, \mathrm{d}\boldsymbol{r}} =: E_T. \tag{4.4}$$

That is, E_T provides an upper bound of the ground states energy E_0. The idea is to choose ψ_T in such a way that we don't merely get an upper bound but an estimate of E_0. For our purpose the following wavefunction will proof sufficient (Note, however, that, in general, to determine a suitable wavefunction is no easy task. see e.g. [28] for a treatment of the commonly used Jastrow-Slater wavefunction)

$$\psi_T(\boldsymbol{r}^{(1)}, \boldsymbol{r}^{(2)}) = \mathrm{e}^{-Z\frac{\|\boldsymbol{r}^{(1)}\| + \|\boldsymbol{r}^{(2)}\|}{a_0}},$$

where by physical reasoning we conclude that $Z \in [1,2]$ (physically Z is a measure of the charge the second electron sees, i.e. how good the first electron is able to shield the charge of the nucleus). To render equation 4.4 into a form suitable for applying importance sampling we rewrite it as

$$E_T = \int_{\mathbb{R}^6} \underbrace{\underbrace{\frac{\psi_T^*(\boldsymbol{r})\psi_T(\boldsymbol{r})}{\int \psi_T^*(\boldsymbol{r})\psi_T(\boldsymbol{r})\,\mathrm{d}\boldsymbol{r}}}_{p(\boldsymbol{r})} \frac{H\psi_T(\boldsymbol{r})}{\psi_T(\boldsymbol{r})}}_{f(\boldsymbol{r})} \mathrm{d}\boldsymbol{r}.$$

Obviously p is a probability density function and we can use importance sampling to estimate the integral. To avoid computing the normalizing integral we use the Metropolis algorithm. Therefore, our approximation E_T is given by (see equation 3.1)

$$E_T \approx \frac{1}{N}\sum_{i=1}^{N} \frac{f(\boldsymbol{r}_i)}{p(\boldsymbol{r}_i)} = \frac{1}{N}\sum_{i=1}^{N} \frac{H\psi_T(\boldsymbol{r}_i)}{\psi_T(\boldsymbol{r}_i)},$$

where $\boldsymbol{r}_i$ are realizations of the probability distribution attached to the pdf p. Our goal is now to compute Z such that E_T is minimized.

Since, we are going to estimate the minimum of Z numerically (and don't want to require the knowledge of specific optimization algorithms) we evaluate it at 20 uniformly spaced points in the interval $[1,2]$. At every point $5 \cdot 10^4$ function evaluations are performed (i.e. $N = 5 \cdot 10^4$). The implementation can be found in Listing 7 whereas Figure 4.3 plots the result.

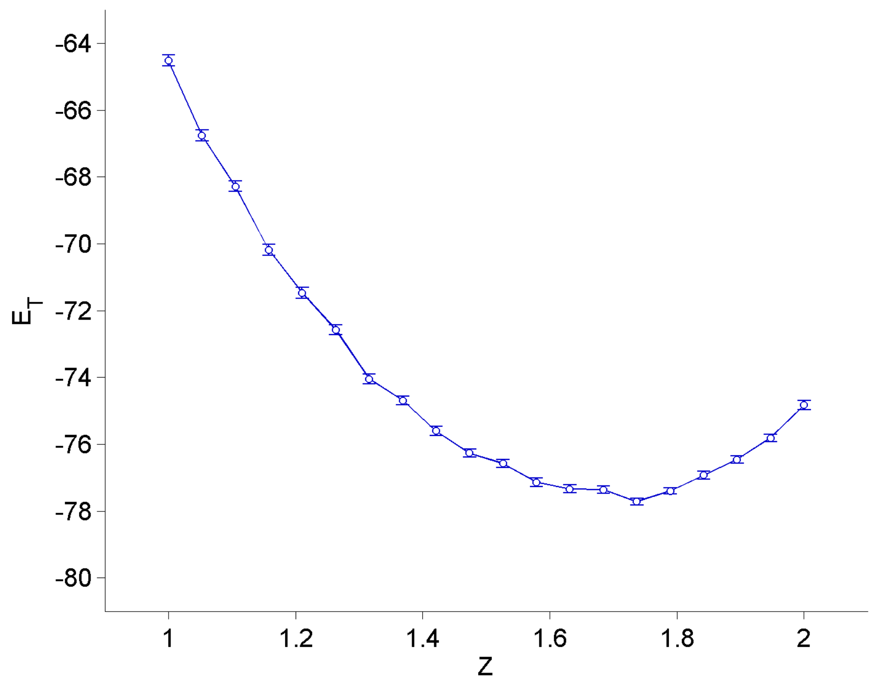

Figure 4.3: Energy levels of the helium atom for $Z \in [1,2]$.

Then $Z \approx 1.7368$ is the value of Z where E_T is minimal (among those computed). The implementation automatically selects this values and computes the integral at this value with $N = 2 \cdot 10^6$ (i.e., we use a larger sample to compute the value once we have determined the minimum with a number of smaller samples). The result of this computation can be found in Table 4.4.

E_T	Standard deviation
-77.4084	0.05534

Table 4.4: Approximation to the ground state energy of helium.

We can compare this to the experimental value of $-78.98\,\mathrm{eV}$ which is off by about $1.6\,\mathrm{eV}$. More exact values can be obtained if we use more complicated wavefunction in place of ψ_T which is also allowed to have more parameters. In this case, however, the use of a minimization algorithm is a necessity.

5 Numerical implementation

5.1 Optimization

In the previous sections we have implemented the Monte Carlo algorithms in matlab, disregarding execution speed for the most part. This, is not always a true limitation since, e.g., the accuracy of Example 4.22 is limited to a greater extend by the approximation made beforehand. Therefore, it is futile to compute the integral to ten significant digits if our approximation is only good to two digits. However, on many occasions a numerical result significant to a large number of digits is necessary and therefore a much larger number of function evaluations has to be performed.

In this section, we will use the C/C++ programming language. However, since this is a bachelor thesis on the topic of Monte Carlo methods we will not discuss any of the common optimization techniques employed in numerical intensive applications. Instead we will focus our development on the fact that the plain Monte Carlo method (and most other Monte Carlo methods) are especially well suited for parallelization; i.e., we can split the numerical intensive part (i.e. evaluating the function to be integrated) to different processing units. Most modern desktop processors (as of 2009) have between two and eight processing units (in this context usually called cores). However, there is a component found in most desktop computers even better suited for massively parallel numerical intensive applications, the GPU (or graphics processing unit). More recent GPUs are fully programmable (a technology called GPGPU or general-purpose computing on graphics processing units) and even includes double precision floating point support. For the purpose of this bachelor thesis however we will use the hardware listed in Table 5.1 (which includes 128 so called stream processors).

Component	Manufacturer	Model
CPU	Intel	i7 920 (2.66 GHz, 4 cores)
GPU	Gainward/Nvidia	GeForce 9800 GTX+, 512 MB (128 stream processors)
RAM	Crucial	4 GB DDR3-1066

Table 5.1: Hardware specifications.

Since, this particular graphic card does support single precision floating point operations only, we will stick to those for the remainder of the discussion. Furthermore, all time measurements in this bachelor thesis are performed using this hardware specifications. To access the GPU a programming platform called CUDA (Compute Unified Device Architecture, for an introduction see [2]) is used. The CPU version is written in C/C++ and compiled using the freely available compiler from Microsoft (see [1]). The CPU implementation does not include multi-core support, i.e. it's performance could be improved by up to four times (this is a theoretical estimate that is unlikely to be achieved in a given implementation). The implementations can be found in Listings 8 and 9 respectively. The performance of these two implementations, is compared in Table 5.2 (for the muon decay process already discussed in Example 3.6).

Platform	N. of function evaluations	Execution time	Compiler options	Relative error
CPU	10^9	60 sec	/Ox	$0.17 \cdot 10^{-4}$
GPU	10^9	0.7 sec	–use_fast_math -O3	$0.76 \cdot 10^{-5}$
GPU	$100 \cdot 10^9$	59 sec	–use_fast_math -O3	$0.30 \cdot 10^{-6}$

Table 5.2: Performance comparison.

Therefore, we conclude that on the GPU the algorithm is about 10^2 times faster than the same algorithm implemented on the CPU. It should also be noted that a relative error of $0.30 \cdot 10^{-7}$ corresponds to a an absolute error of $0.000001 \cdot 10^{-19}$ (shown to the maximum precision of single precision floating point numbers). That is, the approximation is only different from the real value by ± 1 in the last (decimal) digit of the (single precision) floating point number.

5.2 Tree summation

Since the accuracy of the computation done in the previous section is close to the limit of single precision floating point numbers, we have to avoid round-off errors when summing up $1.6 \cdot 10^9$ values. It is well known (see e.g. [19]) that simple summation in a single loop potentially introduces large round-off errors. Therefore, we use a so called **summation tree**. The idea is to compute partial sums

$$S_{1;k} := \sum_{i=kn+1}^{kn+n} f(\boldsymbol{x}_i), \qquad 0 \leq k \leq \frac{N}{n} - 1$$

where n is small compared to $N = n^m$ (for Example $n = 10$ as in Listings 8 and 9). Then, summing up these partials sums we get a new set of partial sums. Continuing until only one partial sum is left we have

$$S_{j;k} := \sum_{i=kn+1}^{kn+n} S_{j-1;i}, \qquad 0 \leq k \leq \frac{N}{n^j} - 1, \; 1 \leq j \leq m$$

where

$$\sum_{i=1}^{N} f(\boldsymbol{x}_i) = S_{m;0}.$$

However, computing $S_{m;0}$ results in significantly lower round-off errors. Therefore, this algorithm is used in the CPU as well as the GPU implementation (see Listings 8 and 9). Obviously, we can let n depend on j and k as well. For our purpose this is not strictly necessary and would complicate the treatment of the topic here as well as the implementation. We refer the interested reader to [19].

5.3 Random number generators

If we consider performance of an Monte Carlo algorithm it is valid to ask if there is a random number generator that, while still generating good random numbers, is fast in execution. For the purpose of this bachelor thesis two different random number generators, namely LCG (linear congruential generator) and the Xor-shift algorithm were implemented. Both of these algorithms are discussed in detail in [29]. We discovered that the LCG generator is only usable if no more than 4 significant digits are required. No such limitations have been observed for the Xor-shift algorithm (in the case of single precision floating point numbers). Therefore, the Xor-shift algorithm is used in all implementations.

Furthermore, the GPU implementation needs a large number of random number generators (basically one for every concurrently running thread). Therefore, we use the rand function from standard C/C++ to seed the Xor-shift generators used on the graphic card.

6 Summary

In summary, we developed Monte Carlo integration along two lines. First, we focused on computing an integral by means of the VEGAS algorithm and, secondly, we avoided computing the integral altogether by using the Metropolis-Hastings algorithm. By illustrating this methods on three simple, yet real world examples, it is our hope to have demonstrated that the algorithms developed here have a number of practical applications.

Obviously, this bachelor thesis by no means gives an exhaustive list of Monte Carlo methods. Instead, it focuses on developing two algorithms that are of vital importance to many applications. The interested reader is referred to [30, 6] (to learn additional variance reduction techniques). Someone more interested in the statistical applications of the Metropolis-Hastings algorithm is referred to [9] which develops and applies the Metropolis-Hastings algorithm to extreme value statistics. Many additional topics can be found in [24]. Although not technically Monte Carlo methods Quasi-Monte Carlo methods share many properties with Monte Carlo methods and are discussed, e.g., in [6].

References

[1] Microsoft Windows SDK for Windows 7 and .NET Framework 3.5 SP1. `http://www.microsoft.com/downloads/details.aspx?FamilyID=c17ba869-9671-4330-a63e-1fd44e0e2505&displaylang=en`.

[2] NVIDIA CUDA, Programming Guide. `http://www.nvidia.com/object/cuda_develop.html`.

[3] FRED (Federal Reserve Economic Data). `http://research.stlouisfed.org/fred2/series/GDP/downloaddata?cid=106`, 2009.

[4] C. Amsler et al. Particle Data Group (2009 partial update for the 2010 edition). *Physics Letters*, B667(1), 2008. `http://pdg.lbl.gov/2009/tables/contents_tables.html`.

[5] Rizwan Butt. *Numerical Analysis using MATLAB.* Infinity Science Press LLC, 2007.

[6] Russel E. Caflisch. Monte Carlo and quasi-Monte Carlo methods. *Acta Numerica*, pages 1–49, 1998. `http://dsec.pku.edu.cn/~tieli/notes/numer_anal/MCQMC_Caflisch.pdf`.

[7] Russell Davidson and James G. MacKinnon. *Principles of Quantum Mechanics.* Plenum Press – New York and London, 2nd edition, 1994.

[8] Russell Davidson and James G. MacKinnon. *Econometric Theory and Methods.* Oxford University Press, USA, 2003.

[9] Ben Downton. Statistical Analysis of Extremes. 2007. `http://fourier.dur.ac.uk/Ug/projects/library/PR4/000406778r.pdf`.

[10] A. Gelman G. O. Roberts and W. R. Gilks. Weak convergence and optimal scaling of random walk Metropolis algorithms. *Ann. Appl. Probab.*, (1):110–120, 1997.

[11] Dani Gamerman and Hedibert F. Lopes. *Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference.* Chapman & Hall/CRC, 2nd edition, 2006.

[12] David Griffiths. *Introduction to elementary particles.* John Wiley & Sons, Inc., 1987.

[13] David Griffiths. *Introduction to Quantum Mechanics.* Prentice Hall, Inc., 1995.

[14] Paul R. Halmos. *Measure Theory.* Springer-Verlag Berlin Heidelberg New York, 1978.

[15] W. K. Hastings. Monte Carlo Sampling Methods Using Markov Chains and Their Applications. *Biometrika*, (1):97–109, 1970.

[16] Karl-Rudolf Koch. *Introduction to Bayesian statistics.* Springer-Verlag Berlin Heidelberg New York, 2nd edition, 2007.

[17] L.D. Kudryavtsev. Encyclopaedia of mathematics. 2002. `http://eom.springer.de/F/f041870.htm`.

[18] G. Peter Lepage. VEGAS - An Adaptive Multi-dimensional Integration Program. *Journal of Computational Physics*, pages 192–203, 1978.

[19] Peter Linz. Accurate Floating-Point Summation. *Coummunications of the ACM*, (13):361–362, 1970.

[20] M. Loeve. *Probability theory I.* Springer-Verlag Berlin Heidelberg New York, 4 edition, 1963.

[21] Erich Novak. Encyclopaedia of mathematics. 2002. `http://eom.springer.de/C/c120310.htm`.

[22] Esa Nummelin. *General irreducible Markov chains and non-negative operators.* Cambridge University Press, 1984.

[23] Sanatan Rai. A non constructive proof of the existence of a maximal irreducibility measure. 2003. `http://arxiv.org/PS_cache/math/pdf/0308/0308108v1.pdf`.

[24] Christian P. Robert and George Casella. *Monte Carlo Statistical Methods.* Springer-Verlag Berlin Heidelberg New York, 2nd edition, 2005.

[25] Gareth O. Roberts and Jeffrey S. Rosenthal. Harris Recurrence of Metropolis-within-Gibbs and Trans-Dimensional Markov Chains. *The Annals of Applied Probability*, 16(4):2123–2139, 2006.

[26] Sheldon Ross. *A First Course in Probability.* Prentice Hall, 5 edition, 1997.

[27] Luke Tierney. Markov Chains for Exploring Posterior Distributions. *Ann. Statist.*, 22(4):1701–1728, 1994.

[28] Julien Toulouse and C. J. Umrigar. Full optimization of Jastrow-Slater wave functions with application to the first-row atoms and homonuclear diatomic molecules. `http://arxiv.org/abs/0803.2905`.

[29] Rebecca Tschenett. Zufallszahlen, University of Innsbruck (bachelor thesis), 2009.

[30] Stefan Weinzierl. Introduction to Monte Carlo methods. 2000. `http://arxiv.org/abs/hep-ph/0006269/`.

A Source code of algorithms and methods

Listing 1: Helper function: randuniform

```
function x = randuniform(N,V)
% Samples uniformly from a rectangular region
% Input:  N, number of values to sample
%         V, 2xd matrix with the boundaries of the rectangle
% Output: x, dxN matrix of sampled values
    x = rand(size(V,2),N);
    for k=1:size(V,2)
        x(k,:) = V(1,k) + (V(2,k)-V(1,k))*x(k,:);
    end
end
```

Listing 2: Plain Monte Carlo method

```
function [I,stddev] = plainmc(N, f, V)
% Implements the Plain Monte Carlo method
% Input:   N, number of function evaluations
%          f, function
%          V, 2xd matrix of integration boundaries
% Output:  I,      approximate value of the integral
%          stddev, approximation of the standard deviation of I
    x = randuniform(N, V);
    I = 0;
    stddev = 0;
    vol = volume(V);
    for k=1:N
        y = feval(f,x(:,k));
        I = I + y;
        stddev = stddev + y^2;
    end
    stddev = sqrt(vol^2/N*(stddev/(N-1) - N/(N-1)*(I/N)^2));
    I = vol*I/N;
end
```

Listing 3: VEGAS algorithm

```
function [I, stddev_bar] = vegas(N, f, V,n)
% Implements the VEGAS algorithm
% Input:  N, number of function evaluations
%         f, function
%         V, 2xd matrix of the integration boundary
%         n, number of bins
% Output: I,      approximate value of the integral
%         stddev, approximation of the standard deviation of I

    % Setup
    d = length(V);
    m = length(N);
    P = zeros(d,n);
    vol = volume(V);
    for i=1:d
        for j=1:n
            P(i,j) = j/n;
        end
    end

    % Main loop
    for a=1:m
        [x, volx] = sample(P,V,N(a));
        S(a) = 0;
        var(a) = 0;
        Sabs = 0;

        % Perform estimation of the integral
        for k=1:N(a)
            y = feval(f,x(:,k));
            S(a) = S(a) + y*volx(k)*n^d;
            Sabs = Sabs + abs(y);
            var(a) = var(a) + (y*volx(k)*n^d)^2;
        end
        S(a)   = 1/N(a) * S(a);
        var(a) = 1/N(a)*(1/((N(a)-1)) * var(a) ...
                 - N(a)/(N(a)-1)*(S(a))^2);

        % If not last iteration rearrange the bins
        if a ~= m
            P = rearrangebins(x,P,Sabs,n,d,f);
        end
    end

    var_bar     = 1/sum(1./var);
    stddev_bar  = sqrt(var_bar);
    I           = var_bar*sum(S./var);
end

function P = rearrangebins(x,P,Sabs,n,d,f)
% Rerranges the bins in the VEGAS algorithm
% Input:   x,    matrix of sampled points
%          P,    current bin matrix
%          Sabs, sum over f(x_i)
%          n,    number of bins
%          d,    number of dimensions
%          f,    function
% Output: P, new bin matrix

    Sb = Sabs/n;
    for i=1:d
        c  = 1;
        Sc = 0;
        % Sorts x by column
        x  = sortrows(x',i)';
        for k=1:length(x)
            Sc = Sc + abs(feval(f,x(:,k)));
            if Sc >= c*Sb
                P(i,c) = k/length(x);
                c = c + 1;
                if c == n
                    break;
                end
            end
        end
    end
end

function [x, vol] = sample(P, V, n)
% Samples according to the distribution specified by the bins
% Input:  P, matrix of bins
%         V, 2xd matrix of the boundaries of integration
%         n, number of bins
% Output: x,   matrix with sampled values
%         vol, vector with the volume of the bin a given sampled
%              vector falls into
    x = zeros(size(V,2),n);
    for i=1:n
        j = randint(1,size(P,1),[1,size(P,2)]);
        Vnew = zeros(size(V));
        for k=1:size(V,2)
                if j(k)-1 == 0
                    Vnew(1,k) = V(1,k);
                    Vnew(2,k) = V(1,k) + (V(2,k)-V(1,k))*P(k,j(k));
                else
                    Vnew(1,k) = V(1,k) + (V(2,k)-V(1,k))*P(k,j(k)-1);
                    Vnew(2,k) = V(1,k) + (V(2,k)-V(1,k))*P(k,j(k));
                end
        end
        x(:,i) = randuniform(1,Vnew);
        vol(i) = volume(Vnew);
    end
end
```

Listing 4: Metropolis algorithm

```
function x = metropolis(N, g, sigma, burn_in_n, step_size, d)
% Implements the Metropolis algorithm
% Input:  N,          number of sampled values
%         g,          function that computes f(x1)/f(x2)
%         sigma,      step size (vector or scalar)
%         burn_in_n,  number of steps discarded before sampling starts
%         step_size,  skipped number of steps between samples
%         d,          number of dimensions
% Output: x, dxN matrix of sampled values
    x = zeros(d,N);
    s = 0;

    % Burn-in period
    for k=1:burn_in_n
        y = x(:,1) + sigma.*randn(d,1);
        r = min(1, feval(g, y, x(:,1)));
        a = rand();
        if a <= r
            x(:,1) = y;
        end
    end

    % Sampling period
    for k=2:N
        x(:,k) = x(:,k-1);
        for j=1:step_size
            y = x(:,k) + sigma.*randn(d,1);
            r = min(1, feval(g,y, x(:,k)));
            a = rand();
            if a <= r
                x(:,k) = y;
                s = s+1;
            end
        end
    end

    % Outputs the acceptance rate
    s/((N-1)*step_size)
end
```

B Source code of examples

Listing 5: Muon decay

```
function [I,stddev] = muondecay()
% computes the decay rate (GAMMA) by means of plain Monte Carlo
% integration and the VEGAS algorithm for different values of N.
% A plot of values and standard deviations as well as an error
% plot is created.

    g      = 0.66;
    M_w    = 80.4;
    m_mu   = 0.105;

    % initializes a constant seed to get reproducable results
    RandStream.setDefaultStream(RandStream('mt19937ar','Seed',0));

    V = [0,0,0,0;1/2*m_mu,2*pi,pi,1/2*m_mu];

    %the actual computation
    x = floor(logspace(3,6,6));
    for k=1:length(x)
        [I1(k), stddev1(k)] = plainmc(x(k), @f, V);
        K = floor(x(k)/10);
        [I2(k), stddev2(k)] = vegas([K;K;x(k)], @f, V, 10);
    end

    tic;
    %plainmc(2*10^6, @f, V);
    toc

    % saves the output values
    I(1) = I1(end);
    I(2) = I2(end);
    stddev(1) = stddev1(end);
    stddev(2) = stddev2(end);
```

```
% create a plot of values and standard deviations
figure('Name', 'Stddev␣plot', 'NumberTitle', 'off');

sol = 3.04226623514192*10^-19;
a = 0.77*x(1);
b = 1.23*x(end);
errorbar(x, I1, stddev1, '-o','Color', [0 0 0.8], ...
         'LineWidth', 1, 'MarkerEdgeColor',[0 0 0.8], ...
         'MarkerFaceColor', [1 1 1], 'MarkerSize', 5);
errorbarlogx(0.01);
hold on;
errorbar(x, I2, stddev2, '-s','Color', [0.8 0 0], ...
        'LineWidth', 1, 'MarkerEdgeColor',[0.8 0 0], ...
        'MarkerFaceColor', [1 1 1], 'MarkerSize', 5);
errorbarlogx(0.01);
plot([a,x,b], sol + zeros(length(x)+2) ,'-.',  ...
     'Color', [0 0.7 0], 'LineWidth', 1);
legend('Plain␣Monte␣Carlo','VEGAS','Exact␣solution', ...
       'Location', 'NorthEast');
ylabel('\Gamma', 'FontSize', 15);
xlabel('N␣(number␣of␣function␣evaluations)', 'FontSize', 15);
set(gca, 'FontSize', 15);

fh = figure(1);
set(fh, 'color', 'white');
set(gca, 'Box', 'off');
axis([a, b, 2.8*10^-19, 3.5*10^-19])
set(gca, 'TickDir', 'out');

print -dpng -r200 muondecay1

% create an error plot
figure('Name', 'Error␣plot', 'NumberTitle', 'off');


loglog(x, abs(I1-sol)./sol, '-o','Color', [0 0 0.8], ...
       'LineWidth', 1, 'MarkerEdgeColor',[0 0 0.8], ...
       'MarkerFaceColor', [1 1 1], 'MarkerSize', 5);
hold on;
loglog(x, abs(I2-sol)./sol, '-s','Color', [0.8 0 0], ...
       'LineWidth', 1, 'MarkerEdgeColor',[0.8 0 0], ...
       'MarkerFaceColor', [1 1 1], 'MarkerSize', 5);
legend('Plain␣Monte␣Carlo','VEGAS','Location','NorthEast');
ylabel('Relative␣error', 'FontSize', 15);
xlabel('N␣(number␣of␣function␣evaluations)', 'FontSize', 15);
set(gca, 'FontSize', 15);

fh = figure(2);
set(fh, 'color', 'white');
set(gca, 'Box', 'off');
set(gca, 'TickDir', 'out');

print -dpng -r200 muondecay2


function Gamma = f(p)
```

```
    % computes the value of the integrant at the point
    % p=(E2,E4,theta2,phi2)^T
        E2   = p(1);
        E4   = p(4);
        the2 = p(3);

        if(E4 < -E2+ 1/2*m_mu)
            Gamma =0;
            return
        end

        AmplitudeSquared = (g/M_w)^4*(m_mu)^2*E2*(m_mu-2*E2);
        Gamma = AmplitudeSquared/((4*pi)^4*m_mu)*sin(the2);
    end
end
```

Listing 6: Autoregression

```
function [S,sigma, b0, b1, b2, ...
          std_sigma, std_b0, std_b1, std_b2] = autoregression()
% Fits an autoregressive model of order 2 by means of Bayes' theorem
% Output: S,          estimate of y_{n+1}
%         sigma,      mean of the sigma paramteter
%         b0,b1,b2,   mean of the beta_0, beta_1, beta_2 parameter
%         std_sigma,  standard deviation of the sigma parameter
%         std_b0, std_b1, std_b2, standard deviation of the
%                                 beta_0, beta_1, beta_2 parameter

    % initializes a constant seed to get reproducable results
    RandStream.setDefaultStream(RandStream('mt19937ar','Seed',0));

    % GDP from Q1 2004 to Q2 2009 in units of 10^13 * $
    name = 'bip';
    y = [11597.2,11778.4,11950.5,12144.9,12379.5,12516.8,12741.6, ...
        12915.6,13183.5,13347.8,13452.9,13611.5,13795.6,13997.2, ...
        14179.9,14337.9,14373.9,14497.8,14546.7,14347.3,14178, ...
        14151.2] / 10000;

    %{
    % Remove the %{ %} to get the "artificial" example
    name = 'artificial';
    y = [0;1];
    n = 100;
    for i=3:n
        y(i) = 0.2 + 1.1*y(i-1) - 0.1*y(i-2) + 0.03*randn();
    end
    %}

    % Sample
    N = 500000;
    x = metropolis(N, @likelihood, [0.01;0.03;0.03;0.03], ...
        5000, 10, 4);

    % Compute mean and standard deviation of the parameters
    sigma      = mean(x(1,:));
    b0         = mean(x(2,:));
```

```
b1          = mean(x(3,:));
b2          = mean(x(4,:));
std_sigma   = std(x(1,:));
std_b0      = std(x(2,:));
std_b1      = std(x(3,:));
std_b2      = std(x(4,:));

% Estimate y_{n+1}
S = 0;
for k=1:N
    S = S + modelfunc(x(2:4,k),y(end),y(end-1));
end
S = S/N;

% Create histograms
createhist(x(1,:), '\sigma', strcat(name,'_sigma'));
createhist(x(2,:), '\beta_0', strcat(name,'_beta0'));
createhist(x(3,:), '\beta_1', strcat(name,'_beta1'));
createhist(x(4,:), '\beta_2', strcat(name,'_beta2'));

function createhist(x, xlab, filename)
% Creates the histogram for different vectors
% and labels
    rhist(x',30);

    xlabel(xlab, 'FontSize', 15);
    ylabel('Probability', 'FontSize', 15);
    set(gca, 'FontSize', 15);
    h = findobj(gca,'Type','patch');
    set(h(1), 'FaceColor', [0 0 0.8]);

    fh = figure(1);
    set(fh, 'color', 'white');
    set(gca, 'Box', 'off');
    set(gca, 'TickDir', 'out');

    print('-dpng','-r200',filename);
end

function P = likelihood(p1, p2)
% Computes the likelihood function

    if p1(1) < 0 || p2(1) < 0
        P = 0;
        return;
    end

    yh1 = modelfunc(p1(2:4), y(2:end-1), y(1:end-2));
    yh2 = modelfunc(p2(2:4), y(2:end-1), y(1:end-2));

    d1 = abs(yh1 - y(3:end));
    d2 = abs(yh2 - y(3:end));

    d1 = normpdf(d1, 0, p1(1));
    d2 = normpdf(d2, 0, p2(1));
```

```
        P = prod(d1 ./ d2);
    end

    function z = modelfunc(p,y1,y2)
    % The model function A(x)
        z = p(1) + p(2)*y1 + p(3)*y2;
    end
end
```

Listing 7: Variational quantum Monte Carlo method

```
function [I,stddev] = vqmc()
% estimates the ground state energy of helium by using
% variational quantum Monte carlo methods

    m      = 511000;
    alpha  = 1/137;
    a0     = 1/(m*alpha);  %bohr radius

    % initializes a constant seed to get reproducable results
    RandStream.setDefaultStream(RandStream('mt19937ar','Seed',0));

    % compute the integral for Z in the interval [1,2]
    N = 50000;
    z = linspace(1,2,20);

    for i=1:length(z)
        Z = z(i);
        [I(i), stddev(i)] = impsampling();
    end

    % plot the result
    errorbar(z, I, stddev, '-o','Color', [0 0 0.8], ...
             'LineWidth',1,'MarkerEdgeColor',[0 0 0.8], ...
             'MarkerFaceColor', [1 1 1],'MarkerSize', 5);
    xlabel('Z', 'FontSize', 15);
    ylabel('E_T', 'FontSize', 15);
    set(gca, 'FontSize', 15);

    fh = figure(1);
    set(fh, 'color', 'white');
    set(gca, 'Box', 'off');
    axis([0.9,2.1,-81,-63])
    set(gca, 'TickDir', 'out');

    print -dpng -r200 vqmc

    % compute the integral at the minimum with more precision
    N          = 200000;
    [val,ind]  = min(I);
    Z          = z(ind);

    [I,stddev] = impsampling();
    function [I, stddev] = impsampling()
    % implements importance sampling by using the Metropolis algorithm
    % Output: I,      estimation of the integral
    %         stddev, approximation of the standard deviation of I
        x = metropolis(N, @P, a0/(2*Z), 5000, 10, 6);

        S = 0;
        stddev = 0;
        for k=1:N
            r = x(:,k);
            y = Hpsi(r)/psi(r);
            S = S + y;
            stddev = stddev + y^2;
        end
        I = S/N;
        stddev = sqrt(stddev/(N*(N-1)) - 1/(N-1)*(I)^2);
    end

    function y = psi(r)
    % \psi_T
        r1 = r(1:3);
        r2 = r(4:6);

        y =  Z^3/(pi*a0^3)*exp(-Z*(norm(r1)+norm(r2))/a0);
    end

    function y = dpsi(r,var)
    % Second derivative of \psi_T by r_var
        y   = Z*r(var)^2/(a0*norm(r)^3)  - Z/(a0*norm(r)) + ...
             Z^2*r(var)^2/(norm(r)^2*a0^2);
    end

    function y = Hpsi(r)
    % Computes H \psi_T
       r1 = r(1:3);
       r2 = r(4:6);

       Laplacenew = 0;
       for l=1:3
           Laplacenew = Laplacenew + dpsi(r1,l) + dpsi(r2,l);
       end
       Laplacenew = psi(r)*Laplacenew;

       Potentials  = -alpha*(2/norm(r1) + 2/norm(r2));
       Interaction = alpha*1/(norm(r2-r1));

       y =  -1/(2*m)*(Laplacenew) + Potentials*psi(r) + ...
            Interaction*psi(r);
    end

    function y = P(r,u)
    % Computes the fraction that is used in the Metropolis algorithm
        y = abs(psi(r))^2 / abs(psi(u))^2;
    end
end
```

Listing 8: Muon decay (CPU)

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <limits.h>
#include <time.h>

#define PI   3.14159265358979323846f

#define G    0.66f
#define M_W  80.4f
#define M_MU 0.105f

#define ITERATIONS_PER_THREAD 100
#define NUMBER_OF_THREADS 100
#define NUMBER_OF_BLOCKS  100
#define NUMBER_OF_ITERATIONS 1000

#define D 4

#define N  (ITERATIONS_PER_THREAD*NUMBER_OF_THREADS*NUMBER_OF_BLOCKS)

#define N_TOTAL (N*NUMBER_OF_ITERATIONS)


/* Xor-shift random number generator
*/
void randui(unsigned int *a) {
    unsigned int t = a[0] ^ (a[0] << 11);
    a[0] = a[1];
    a[1] = a[2];
    a[2] = a[3];
    a[3] = (a[3] ^ (a[3] >> 19)) ^ (t ^ (t >> 8));
}


void randf(float *y, unsigned int *seed) {
    randui(seed);
    *y = ((float)*seed)/((float)UINT_MAX);
}

void randuniform(float *p, float* V,unsigned int *seed) {
    int k=0;
    float x;
    for(;k<D;k++) {
        randf(&x, seed);
        p[k] = V[k] + (V[D+k] - V[k])*x;
    }
}


/* Function to be integrated
*/
```

```c
void f(float *p, float *ret) {
    float E2   = p[0];
    float E4   = p[3];
    float the2 = p[2];

    if(E4 < -E2 + M_MU/2.0f) {
            *ret = 0;
            return;
    }

    *ret = (1.912059445E-14)*E2*(M_MU-2*E2)*sin(the2);
}


/* Computes Volume
*/
void volume(float *V, float *vol) {
    int k = 0;
    *vol = 1;

    for(;k<D;k++)
            *vol *= V[D+k]-V[k];
}


/* Computes sum by means of a sum tree
*/
void sum(float *p, int n) {
    while(n > 1) {
            for(int j=0;j<n/10;j++)  {
            float I = 0;
            for(int k=0;k<10;k++) {
                            I += p[j*10 + k];
            }
            p[j] = I/10.0f;
        }
        n = n/10;
    }
}

/* Main
*/
int main() {
    clock_t start = clock();

    srand( 0 );
    unsigned int seed[4];
    seed[0] = rand();
    seed[1] = rand();
    seed[2] = rand();
    seed[3] = rand();

    float y;
    float I      = 0;
    float p[D];
    float vol;
    float V[2*D] = {0,0,0,0, M_MU/2.0f, 2*PI, PI, M_MU/2.0f};
    float *It    = (float*)malloc(sizeof(float)*N);
    float *Is    = (float*)malloc(sizeof(float)*NUMBER_OF_ITERATIONS);

    for(int j=0;j<NUMBER_OF_ITERATIONS;j++) {
        for(int k=0;k<NUMBER_OF_THREADS*NUMBER_OF_BLOCKS;k++) {
            It[k] = 0;
            for(int l=0;l<ITERATIONS_PER_THREAD;l++) {
                randuniform(p, V, seed);
                f(p, &y);
                It[k] += y;
            }
            It[k] /= (float)ITERATIONS_PER_THREAD;
        }

        sum(It, NUMBER_OF_THREADS*NUMBER_OF_BLOCKS);
        Is[j] = It[0];
    }

    volume(V, &vol);
    sum(Is, NUMBER_OF_ITERATIONS);
    I = vol*Is[0];

    printf("Computed␣%E␣samples␣in␣%f\n␣seconds\r\n", (float)N_TOTAL,
                   ((double)clock() - start) / CLOCKS_PER_SEC);
    printf("I:␣%E\r\n", I);
    printf("ERROR:␣%fE-19\r\n", (3.04226623514192E-19-I)*1E19 );
    printf("Rel.␣Error:␣%.14Lf\r\n",
                   abs(3.04226623514192E-19/((double)I)-1.0) );

    free(It);
    free(Is);
    return 0;
}
```

Listing 9: Muon decay (GPU)

```
#include <stdio.h>
#include <math.h>
#include <limits.h>
#include <cutil_inline.h>
#include <time.h>

#define PI   3.14159265358979323846f

#define G    0.66f
#define M_W  80.4f
#define M_MU 0.105f

#define ITERATIONS_PER_THREAD 100
#define NUMBER_OF_THREADS 100
#define NUMBER_OF_BLOCKS  100
#define NUMBER_OF_ITERATIONS 100000

#define D 4
```

```
#define N  (ITERATIONS_PER_THREAD*NUMBER_OF_THREADS*NUMBER_OF_BLOCKS)

#define N_TOTAL  (N*NUMBER_OF_ITERATIONS)
#define N_TOTAL_F ((float)N*(float)NUMBER_OF_ITERATIONS)


/* Xor-shift random number generator
*/
__device__ void randui(unsigned int *a) {
    unsigned int t = a[0] ^ (a[0] << 11);
    a[0] = a[1];
    a[1] = a[2];
    a[2] = a[3];
    a[3] = (a[3] ^ (a[3] >> 19)) ^ (t ^ (t >> 8));
}

__device__ void randf(float *y, unsigned int *seed) {
    randui(seed);
    *y = ((float)seed[3])/((float)UINT_MAX);
}

__device__ void randuniform(float *p, float* V, unsigned int *seed) {
    int k=0;
    float x;
    for(;k<D;k++) {
        randf(&x, seed);
        p[k] = V[k] + (V[D+k] - V[k])*x;
    }
}


/* Function to be integrated
*/
__device__ void f(float *p, float *ret) {
    float E2   = p[0];
    float E4   = p[3];
    float the2 = p[2];

    if(E4 < -E2 + M_MU/2.0f) {
            *ret = 0;
            return;
    }

    *ret = (1.912059445E-14)*E2*(M_MU-2*E2)*__sinf(the2);
}


/* (GPU) Kernel
*/
__global__ void plainmc(float *d_I, float *V, unsigned int *d_seed) {
    float y;

    unsigned int seed[4];
    seed[0] = d_seed[4*(NUMBER_OF_THREADS*blockIdx.x+threadIdx.x)];
    seed[1] = d_seed[4*(NUMBER_OF_THREADS*blockIdx.x+threadIdx.x)+1];
    seed[2] = d_seed[4*(NUMBER_OF_THREADS*blockIdx.x+threadIdx.x)+2];
    seed[3] = d_seed[4*(NUMBER_OF_THREADS*blockIdx.x+threadIdx.x)+3];

    float      I = 0;
    float p[D];

    for(int k=0;k<ITERATIONS_PER_THREAD;k++) {
        randuniform(p, V, seed);
        f(p, &y);

        I       += y;
    }

    d_I[NUMBER_OF_THREADS*blockIdx.x +  threadIdx.x]
        = I/((float)ITERATIONS_PER_THREAD);

    d_seed[4*(NUMBER_OF_THREADS*blockIdx.x+threadIdx.x)] = seed[0];
    d_seed[4*(NUMBER_OF_THREADS*blockIdx.x+threadIdx.x)+1] = seed[1];
    d_seed[4*(NUMBER_OF_THREADS*blockIdx.x+threadIdx.x)+2] = seed[2];
    d_seed[4*(NUMBER_OF_THREADS*blockIdx.x+threadIdx.x)+3] = seed[3];
}


/* Computes Volume
*/
void volume(float *V, float *vol) {
    int k = 0;
    *vol = 1;

    for(;k<D;k++)
        *vol *= V[D+k]-V[k];
}


/* Computes sum by means of a sum tree
*/
void sum(float *p, int n) {
        while(n > 1) {
                for(int j=0;j<n/10;j++)  {
                        float I = 0;
                        for(int k=0;k<10;k++) {
                                I += p[j*10 + k];
                        }
                        p[j] = I/10.0f;
                }
                n = n/10;
        }
}


/* Host code
*/
int main()
{
    clock_t start = clock();

    /*  Declare host and device variables */
```

```
size_t size   = NUMBER_OF_THREADS*NUMBER_OF_BLOCKS * sizeof(float);
size_t ulsize = NUMBER_OF_THREADS*NUMBER_OF_BLOCKS *
                sizeof(unsigned int) * 4;

float I = 0;
float vol;
float *d_I;
float *d_V;
unsigned int *d_seed;
float *h_I;
float *Is;
float h_V[2*D] = {0,0,0,0, M_MU/2.0f, 2*PI, PI, M_MU/2.0f};
unsigned int h_seed[NUMBER_OF_THREADS*NUMBER_OF_BLOCKS*4];

h_I      = (float*)malloc(size);
Is       = (float*)malloc(NUMBER_OF_ITERATIONS*sizeof(float));
cudaMalloc((void**)&d_I     , size);
cudaMalloc((void**)&d_V, 2*D*sizeof(float));
cudaMalloc((void**)&d_seed, ulsize);


/* Initialize variables */
srand( 0 );
for(int j=0;j<NUMBER_OF_THREADS*NUMBER_OF_BLOCKS;j++) {
    h_seed[4*j]    = (unsigned int)rand();
    h_seed[4*j+1]  = (unsigned int)rand();
    h_seed[4*j+2]  = (unsigned int)rand();
    h_seed[4*j+3]  = (unsigned int)rand();
}


/* Copy to device (graphic card) */
cudaMemcpy(d_V, h_V, 2*D*sizeof(float), cudaMemcpyHostToDevice);
cudaMemcpy(d_seed, h_seed, ulsize, cudaMemcpyHostToDevice);


/* Computation */
for(int l=0;l<NUMBER_OF_ITERATIONS;l++) {
    plainmc<<<NUMBER_OF_BLOCKS, NUMBER_OF_THREADS>>>
    (d_I, d_V, d_seed);
    cudaMemcpy(h_I, d_I, size, cudaMemcpyDeviceToHost);

    sum(h_I, NUMBER_OF_THREADS*NUMBER_OF_BLOCKS);
    Is[l] = h_I[0];
}

sum(Is, NUMBER_OF_ITERATIONS);
volume(h_V, &vol);
I = vol*Is[0];

/* Output results */
printf("Computed␣%E␣samples␣in␣%f\n␣seconds\r\n",
       N_TOTAL_F, ((float)clock() - start) / CLOCKS_PER_SEC);
printf("I:␣%E\r\n", I);
printf("ERROR:␣%fE-19\r\n", (3.04226623514192E-19-I)*1E19 );
printf("Rel.␣Error:␣%.14Lf\r\n",
            abs(3.04226623514192E-19/((double)I)-1.0) );


    /* Free variables */
    cudaFree(d_I);
    cudaFree(d_V);
    cudaFree(d_seed);
    free(h_I);
    free(Is);
    return 0;
}
```