

# Quantum Mechanics on a Discrete Time Lattice

*DESY Summer Student Programme, 2021*

**Annamaria Benakova**

*Aberystwyth University, Wales, UK*

Supervisor: Karl Jansen



September 8, 2021

## **Abstract**

Feynman's path integral approach to quantum mechanics is introduced. The harmonic oscillator gets considered at a discrete time lattice. A rotation gets performed from Minkowski to Euclidean time to enable the harmonic oscillator to get explored. It's explored with Monte Carlo using python. The exact average squared position value of 0.44727272727272727 gets compared with the values from the Monte Carlo simulation.

---

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Feynman's Path Integral . . . . .	1
1.2	Harmonic Oscillator on a Discrete Time Lattice . . . . .	1
<b>2</b>	<b>Methodology</b>	<b>2</b>
<b>3</b>	<b>Results</b>	<b>3</b>
<b>4</b>	<b>Discussion of Results</b>	<b>3</b>
<b>5</b>	<b>Conclusions</b>	<b>4</b>
<b>6</b>	<b>Acknowledgements</b>	<b>5</b>
<b>A</b>	<b>Wick's Rotation from Minkowski to Euclidean Action</b>	<b>6</b>
<b>B</b>	<b>Exact Average Squared Position</b>	<b>6</b>
<b>C</b>	<b>Python Codes</b>	<b>7</b>

# 1 Introduction

## 1.1 Feynman's Path Integral

The idea behind it is that it sums all the paths possible that connect the initial point and final point. Feynman's path integral takes the original form of:

$$Z(x_f, x_i) = \sum_{paths} \exp\left(\frac{i \cdot S_m}{\hbar}\right) \quad (1)$$

where  $S_m$  is the Minkowski action. The classical action is defined using the Lagrangian,  $L(x, \dot{x}, t) = \frac{1}{2}m\dot{x}^2 - V(x)$  to give:

$$S_{Cl} = \int \left[ \frac{1}{2}m\dot{x}^2 - V(x) \right] dt \quad (2)$$

where in the brackets the kinetic energy is subtracted from the potential energy. The potential energy of a harmonic oscillator,  $V(x) = \frac{1}{2}m\omega^2 x^2$ . Equations (1) and (2) are used in order to get the Euclidean action (derivation in **Appendix A**) to substitute into the path integral:

$$Z(x_f, x_i) = \sum_{paths} \exp\left(-\frac{S_e}{\hbar}\right) \quad (3)$$

this eliminates the imaginary time in the exponential.

## 1.2 Harmonic Oscillator on a Discrete Time Lattice

A time lattice consists of a discrete time axis where the position is in terms of the time,  $t_j$ :

$$x(t_j) = x_j \quad (4)$$

The time lattice consists of slices of time which are separated by a spacing,  $\epsilon$ . So, for two neighbouring time slices:

$$t_{j+1} - t_j = \epsilon \quad (5)$$

this makes the time axis discrete due to the time slices. Due to a discrete time lattice, each separate path can be defined on it. Each path in turn has to be continuous for them to be defined on the discrete time lattice. At each time slice (i.e. lattice point)  $x_i$  gives the particle's location.

For a discrete time lattice the action can be defined:

$$S = \sum_{j=1}^N a \left[ \frac{1}{2}m_0 \frac{(x_{j+1} - x_j)^2}{a^2} + \frac{1}{2}m_0\omega^2 x_j^2 \right] \quad (6)$$

with: initial mass  $m_0$ , angular frequency  $\omega$ ,  $a = i\epsilon$  or  $a = \frac{T}{N}$ .

## 2 Methodology

An analytic continuation - i.e. Wick rotation - was performed on the Minkowski action to get the Euclidean action in order to write the path integral in terms of it - the way it was performed is shown in **Appendix A**.

The action was then computed for a discrete time lattice using equation (6) with the potential energy,  $V$  for a harmonic oscillator used. It hence required certain parameters to be defined;  $m_0 = 1$ , number of paths,  $N = 10$ ,  $\omega = \sqrt{2}$ ,  $T = 10$ , with  $a$  defined as  $a = \frac{T}{N}$ . The position,  $x_j$  was defined over periodic boundary conditions in order to fulfill the need to be continuous. This meant that  $x_{N-1} = x_0$ ; when lattice point  $N-1$  was reached, the next lattice point would be 0. The way this was implemented into the code through an if condition (shown in **Appendix C**) where the lattice point would return back to the beginning after  $N-1$  was reached.

Using the above calculation, the metropolis algorithm was formed. A random number,  $x'_j$  is generated with a uniform probability. The random number was accepted if it met the condition:

$$(x_j - \Delta) \leq x'_j \leq (x_j + \Delta) \quad (7)$$

where  $\Delta$  was taken as 2. Then, the action (equation (6)) was computed for each  $x_j$  and  $x'_j$  being  $S_{x_j}$  and  $S_{x'_j}$  respectively. The difference between them was taken:

$$dS = S_{x_j} - S_{x'_j} \quad (8)$$

considering the  $dS$ , if it was lowered when  $x_j$  was replaced by  $x'_j$  the  $x_j$  was replaced by  $x'_j$ . However, if  $dS$  was more than or equal to 0, another random number,  $r$  was generated with a uniform distribution between 0 and 1. If the random number  $r$  was less than  $\exp(-dS)$ , then the  $x_j$  was set to  $x'_j$ . The Metropolis algorithm is shown in **Appendix C**. This Monte Carlo (Metropolis algorithm) was ran  $k$  times, where  $k$  was equal to 500 000. This updated paths and the average squared position was taken for each generated path by the Monte Carlo and plotted against  $k$ . This was then compared to the exact value of the average squared position. The exact value was calculated using equations in **Appendix B** whilst the average squared position of the monte carlo simulation was found using a function defined in **Appendix C**.

### 3 Results

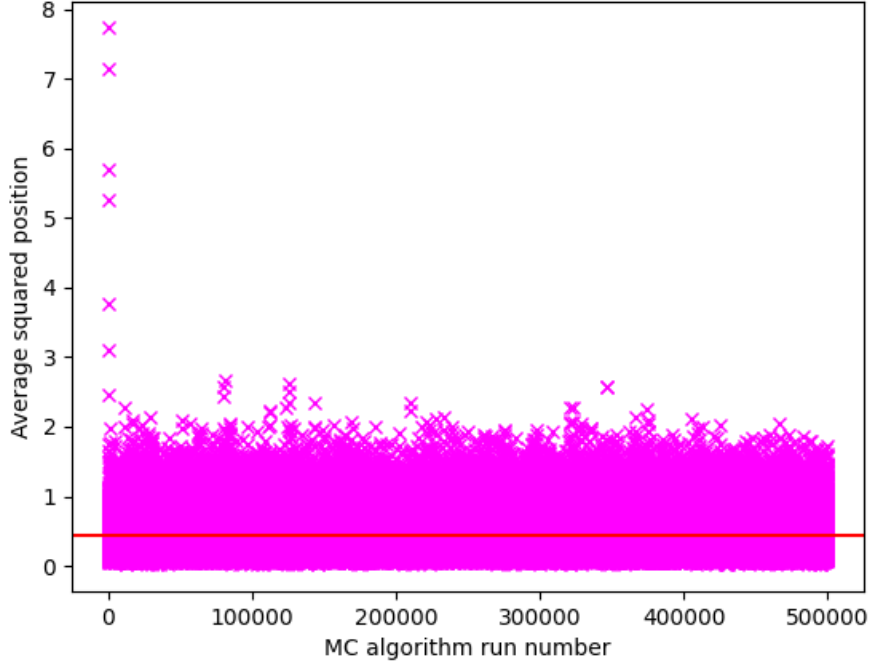


Figure 1: Average squared position plot (red line is the exact position)

1	Average squared position	0.44692049664576267
2	Standard error of mean (average squared position)	0.00033031109647451974
3	Exact Average squared Position	0.44727272727272727

Table 1: Averages

### 4 Discussion of Results

The average squared position was calculated after the first 10 000 points. At the beginning of the plot in **Figure 1** the first points were further from the exact value (red line) and so they were not considered in the average squared position value and gave a value that was in agreement with the exact value to three decimal places (0.447). By doing this the data further from the exact value is skipped, meaning that the system was considered when it was thermalised.

The values on the plot were concentrated between 0 and 2. The standard error of the mean was small, and so the distribution of the data spread was small - this means a more accurate result was reached.

## 5 Conclusions

The exact average squared position was found to three decimal places with the Monte Carlo simulation performed in python. The experimental value came out to be 0.447 (3 d.p.). It's hence been shown that the exact average squared position using this simulation can be found to reasonable accuracy if specific measures are taken to reduce error. In further projects it can be explored how to reach the exact value more accurately.

## 6 Acknowledgements

I'd like to thank both Karl Jansen and Arianna Crippa for all the patience, and support they gave whenever I needed it - they lead me to achieve things I wouldn't think I'd be able to. I would also like to thank Yasaman for being a pleasant colleague to work with whenever we worked together to solve a similar issue.

Working on this project allowed me to view some topics under a different perspective - and for that I'd like to thank the Karl and the DESY team for all the effort that was placed in order to make this project possible even during such strange times.

## References

- [1] M. Creutz and B. Freedman, *A Statistical Approach to Quantum Mechanics*, Annals of Physics, **132**, 427-462 (1981).

## A Wick's Rotation from Minkowski to Euclidean Action

A Wick rotation is an analytic continuation of the the time coordinate. The rotation is performed to get the Euclidean action. Consider the classical action,  $S_{cl}$ . Let  $t = -i\tau$ , thus:

$$dt = -id\tau \quad (9)$$

this is then substituted into the classical action equation,  $S_{cl}$ . Hence leading to the Euclidean action,  $S_e$ :

$$\begin{aligned} S_e &= \int -id\tau \left[ \frac{1}{2} m \dot{x}^2 - V \right] \\ &= i \int d\tau \left[ -\frac{1}{2} m \dot{x}^2 + V \right]. \end{aligned} \quad (10)$$

The Minkowski action can be written in terms of the Euclidean action as follows:

$$S_{cl} = -\frac{S_e}{i}. \quad (11)$$

Consider the original Feynman's path integral from equation (1), and substitute equation (6) into it:

$$\begin{aligned} Z(x_f, x_i) &= \sum_{paths} \exp\left(\frac{i}{\hbar} \left(-\frac{S_e}{i}\right)\right) \\ &= \sum_{paths} \exp\left(-\frac{S_e}{\hbar}\right) \\ &= \int Dx e^{-\frac{1}{\hbar} S_e} \end{aligned} \quad (12)$$

## B Exact Average Squared Position

The average squared position considers a quantity,  $R$  which is defined as:

$$R = 1 + \frac{a^2 \mu^2}{2} - a\mu \left(1 + \frac{a^2 \mu^2}{4}\right)^{\frac{1}{2}} \quad (13)$$

which can be then used in the equation for the average squared position. The way this is defined is:

$$\langle x^2 \rangle = \frac{1}{2\mu(1 + a^2 \mu^2/4)^{\frac{1}{2}} \left(\frac{1+R^N}{1-R^N}\right)} \quad (14)$$



## C Python Codes

```

import numpy as np
def ActionDiscreteT(m,i,xj,u,N,T):
    if(i < N-1):
        S = a*((0.5*m*(xj[i+1]-xj[i])**2/a**2)+(0.5*m*u**2*xj[i]**2)
              +(0.5*m*(xj[i]-xj[i-1])**2/a**2))
    else:
        S = a*((0.5*m*(xj[0]-xj[N-1])**2/a)+(0.5*m*u**2*xj[N-1]**2)
              +(0.5*m*(xj[N-1]-xj[N-2])**2/a))
    return S

def MetropolisAlgorithm(xj,delta):
    for i in range(len(xj)):
        xdj = np.copy(xj)
        xdj[i] = np.random.uniform(xj[i]-delta,xj[i]+delta)

        Sxdj = ActionDiscreteT(m,i,xdj,u,N,T)
        Sxj = ActionDiscreteT(m,i,xj,u,N,T)
        ds = Sxdj - Sxj
        if ds < 0:
            xj[i] = xdj[i]
        if ds >= 0:
            r = np.random.uniform(0,1)
            if np.exp(-ds) > r:
                xj[i] = xdj[i]
    return xj

def ExpValuex2(N,xj):
    xbar2 = 0
    for i in range(len(xj)):
        xbar2 += (1/N)*(xj[i])**2
    return xbar2

```