Quantum Mechanical Energy of Harmonic Oscillator.

Gbenga Agunbiade[1]

[1]Department of Physics and Astronomy, University of Kansas, Lawrence, KS 66045

gsagunbiade@ku.edu

Abstract. *This Python code is designed to estimate the energy of a quantum mechanical system in the context of a harmonic oscillator. The code uses a Monte Carlo method to integrate the probability distribution function defined by the trial wave function. The energy is then calculated using the local energy function for the ground or first excited state. The alpha parameter in the trial wave function is iterative updated using a stochastic gradient descent algorithm to minimize the energy. The code also allows the user to input command-line arguments to specify the number of Monte Carlo samples, the number of iterations for the minimization, the seed for the alpha parameter, and the energy level to estimate (ground state or first excited state). Finally, the code outputs a plot of the estimated energy as a function of iteration and the estimated alpha parameter as a function of iteration.*

1. Introduction

Python is usually used to develop software and websites, analysis of data, task automation, and data visualization. Since it's relatively easy to learn, Python has been adopted by many non-programmers such as accountants and scientists, for a variety of everyday tasks, like organizing finances. This code is a Python implementation of the Variation Monte Carlo method to estimate the ground and first excited state energies of a one-dimensional harmonic oscillator. The code reads some input parameters from the command line and runs a minimization loop for the parameter alpha that appears in the trial wave functions. The trial wave functions, along with the corresponding local energy functions, are defined at the beginning of the code.

2. Random Code

The random code begins with two import statements ‘import math‘ and ‘import numpy as np‘. While the former imports the ‘math‘ module, which provides various mathematical functions and constants, the latter imports the ‘numpy‘ module and assigns it the alias ‘np‘. ‘numpy‘ is a powerful library for numerical computing in Python. It provides support for multi-dimensional arrays and a wide range of mathematical functions. The code then defines a class named ‘Random‘ that serves as a random number generator. This class has several methods for generating different types of random numbers: The ‘init‘ is the initialization method for the ‘Random‘ class. It takes an optional argument ‘seed‘, which is set to 5555 by default. The ‘int64‘ is a method that generates a random 64-bit integer using the Xorshift algorithm. It updates the internal state variables and performs bitwise operations to generate the random number. In addition, the ‘rand‘ is a method that returns a random floating-point number between 0 and 1. ‘Categorical‘ is a method that generates a random integer (0 to 7) according to a categorical distribution.

The probabilities of each category are provided as arguments (‘p1‘ to ‘p7‘). The method uses the ‘rand‘ method to generate a random number and determines which category the random number falls into based on the provided probabilities. ‘Exponential‘ is a method that generates a random double according to an exponential distribution. The rate parameter ‘beta‘ is provided as an argument. The method uses the ‘rand‘ method to generate a random number and applies the inverse transform sampling method to obtain a random value from the exponential distribution. ‘Bernoulli‘ is a method that generates a random integer (0 or 1) according to a Bernoulli distribution. The probability of success ‘p‘ is provided as an argument. The method uses the ‘rand‘ method to generate a random number and returns 1 if the random number is less than the probability of success, indicating success, otherwise, it returns 0. ‘TruncExp‘ is a method that generates a random double according to a truncated exponential distribution. It takes the rate parameter ‘beta‘, and the lower and upper bounds of the truncation range (‘bottom‘ and ‘top‘) as arguments. The method repeatedly generates a random value from the exponential distribution using the ‘Exponential‘ method until it falls within the specified truncation range.

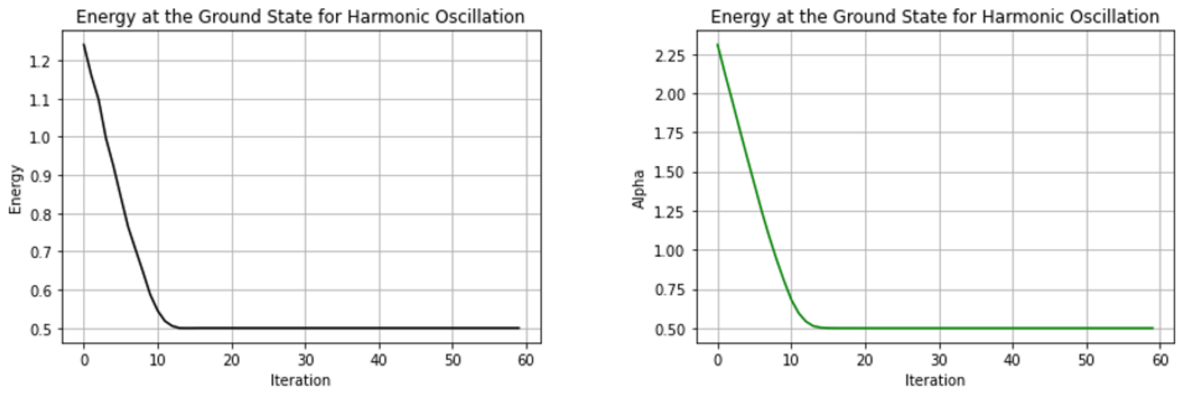

Figura 1. Ground state energy of a harmonic oscillator

3. Implementation, Results and Discussion

The Monte Carlo integration is performed by the function Monte Carlo(), which generates a random walk and calculates the energy and other quantities of interest for a given value of alpha. The random walk is controlled by the function PD(), which defines the probability distribution of the walker. The minimization loop is performed by iterating over Nwalk steps, each of which performs Nstep Monte Carlo steps. At each step, the energy and other quantities of interest are averaged over the samples, and the derivative of the energy with respect to alpha is used to update the value of alpha. The code imports necessary modules such as ‘sys‘, ‘numpy‘ (as ‘np‘), and ‘matplotlib.pyplot‘ (as ‘plt‘) for system-specific parameters, numerical operations, and plotting, respectively. The code appends the current directory to the system path to import a custom module named "Random"from the current directory. Two trial functions (‘ftrial 0‘ and ‘ftrial 1‘) are defined. These functions represent different trial wave functions used to approximate the true wave function of the harmonic oscillator system. They take the position ‘x‘ and a parameter ‘alpha‘ as inputs. Two local energy functions (‘E local 0‘ and ‘E local 1‘) are defined. These functions calculate the local energy of the harmonic oscillator system at a given position ‘x‘ and with a specific ‘alpha‘ value. The code then checks if it is being run as the main program using the ‘if name "main":‘ condition. This ensures that the code inside this block is executed only when the script is run directly. If the command-line arguments contain ‘h‘ or ‘help‘, a help message is printed explaining the usage of the script, and the program exits. Default values are assigned to several variables that control the behavior of the program. These variables include ‘seed‘, ‘ftrial‘, ‘E local‘, ‘E level‘, ‘NMC‘, ‘alpha‘, ‘Nwalk‘, ‘Nstep‘, and ‘gamma‘. They represent parameters such as the random seed, the trial, and local energy functions, the energy level, the number of Monte Carlo samples, the alpha parameter, and the number of iterations for minimization. The code reads user-provided command-line arguments and updates the corresponding variables accordingly. For example, if the ‘-seed‘ argument is present, the subsequent value is assigned to the ‘seed‘ variable. A probability distribution function (‘PD‘) is defined, which returns the square of the trial function evaluated at a position ‘x‘ with a given ‘alpha‘. This probability distribution function is used for Monte Carlo integration. The ‘Monte Carlo‘ function performs the Monte Carlo integration to estimate the energy of the harmonic oscillator system. It takes two arguments: ‘N‘ (the number of iterations) and ‘alpha‘ (the parameter value). Within the function, a random position ‘x‘ is generated within a certain range based on ‘alpha‘. The Metropolis algorithm is used to accept or reject a new position based on the probability distribution ratio. The local energy at each position is calculated, and running sums of the energies, energy squares, and other quantities are updated. The average values of these quantities are computed and returned. The code initializes empty arrays (‘E array‘, ‘alpha array‘, ‘variance array‘) to store the energies, alpha values, and variances obtained during the minimization process. Finally, the resulting energy estimates are accumulated, and the value of alpha is updated based on the gradient descent method. The current value of alpha, the estimated energy, and the squared variance of the energy are printed for each iteration. The energy, alpha, and variance values are appended to the respective arrays. The code produces two plots, one of the energies versus iteration, and one of the values of alpha versus iteration as shown in Figure 1. The title and y-axis label of the plots depends on whether the ground or first excited state is being calculated.

4. Conclusion

This work estimates the energy of a quantum mechanical system in the context of a harmonic oscillator. The code uses a Monte Carlo method to integrate the probability distribution function defined by the trial wave function. The random code begins with two import statements import math and import numpy as np. The code imports necessary modules such as sys, numpy (as np), and matplotlib.pyplot (as plt) for system-specific parameters, numerical operations, and plotting, respectively. The code appends the current directory to the system path to import a custom module named "Random"from the current directory. The code initializes empty arrays (E array, alpha array, ‘variance array‘) to store the energies, alpha values, and variances obtained during the minimization process. Lastly, the code produces two plots: one of the energies versus iteration, and one of the values of alpha versus iteration. The title and y-axis label of the plots depends on whether the ground or first excited state is being calculated.

5. Reference

G. Agunbiade (2023). Simulating and Visualizing the Behavior of a Two-Dimensional Random Walk using the Monte Carlo Method. PHSX 815 Project 2