Simulating Probability Amplitudes of Bound Particles with MCMC Methods

Rowan d'Auria, Supervised by Dr. Aidin Masouminia
University of Durham Physics Department
Submitted: March 13, 2020

Two MCMC methods are used to calculate the probability amplitude of a trapped particle. A simple Metropolis algorithm was used to optimise particle paths, and the Hamiltonian Algorithm was used to compute particle position proposals. In the one- & three-dimensional cases, both methods provide accurate approximations to the analytic solution for the quantum harmonic oscillator. The Metropolis Algorithm was used to calculate a more realistic simulation of a trapped particle in an oscillating electric field. Applications of the methods used herein are discussed in the context of antimatter particle trapping.

I. INTRODUCTION

Richard Feynman introduced the path integral formulation of Quantum Mechanics in 1942, in his postdoctoral thesis at Princeton[1]. This approach uses the concept of classical least-action to link the probability amplitude of a particle and its motion as a function of time. The classical path of the object is one in which the action along the path is minimized[2]. However, to find the probability amplitude of a particle at time $t > 0$, a sum is performed over all possible paths. This is impossible to perform numerically, as there are an infinite number of paths to sum over. To overcome this, the possible paths are discretized and only the paths with the highest contribution are summed. To optimise path selection, Markov Chain Monte Carlo (MCMC) algorithms are used to find the most probable paths.

The Metropolis Algorithm is the simplest implementation of MCMC, but more advanced methods have been developed that improve performance. One method for optimizing the Metropolis Algorithm is to increase the probability of proposal acceptance. This can be achieved through the introduction of Hamiltonian dynamics, allowing distant proposals with a high probability of acceptance. This method was introduced in 1987 by Duane et. al, in application to simulations of quantum chromodynamics[3]. Calculating proposals with a high probability of acceptance is valuable, because it enables samples to be proposed with a lower correlation than Metropolis. Furthermore, obtaining distant proposals can help mitigate the effects of potential barriers, such as a double harmonic potential, because distant proposals can jump between potential wells.

In this report, Metropolis and Hamiltonian algorithms are used to calculate the probability amplitude of a charged particle, in a one and three-dimensional potential well. The results of two algorithms are compared to analytic models obtained with the Schrödinger formulation. The effects of constant and oscillating electric fields are also simulated in three dimensions. The relative performance and breadth of applications for each algorithm are also discussed.

II. THEORY

If a particle is in an initial state $a = (x_a, t_a)$, the classical concept of action can be used to describe the probability that the particle will later be in a state $b = (x_b, t_b)$. Conceptually, the contributions from all possible paths between a and b are summed, not only the path of least action. In fact, all paths independent of S have the same weighting, but they have a different phase that depends on their action. The evolution of a particle state is given by

$$G(b,a) = A \sum_{paths} e^{iS[b,a]/\hbar} \tag{1}$$

where G is the propagator from a to b, S is the action along each path, $\hbar$ is the reduced Plank's constant, and A is a constant. The propagator describes the time evolution of a particle, such that $P(b,a) = |G(b,a)|^2$ where P is the probability of going from state a to b[2]. In units of $\hbar$, S on the classical scale is extremely large, so a small change S results in a large change to the phase contribution of the path. This means that when a path is far from the classical path of least action, tiny path alterations induce phase shifts, cancelling out neighbouring path contributions. Only along the path of least action, where path changes result in modest changes to S, does this not occur and we are left with the classical path.

However, summing over theoretically infinite paths is mathematically intractable. Instead, we use an discretisation approach like the Riemann integral method. The independent variable, which in our case is time, is split up into regular intervals of size δ. This results in a set of time values t_i, each with a corresponding position variable x_i. This subset is defined as

$$\begin{aligned} N\delta &= t_b - t_a & t_{i+1} &= t_i + \delta \\ t_0 &= t_a & t_N &= t_b \\ x_0 &= x_a & x_N &= x_b \end{aligned} \tag{2}$$

We can make a path by joining successive x_i points with a straight line. With a path defined in this way, it is possible to sum over this subset of possible paths between (t_a, x_a) and (t_b, x_b). This is performed with a multidimensional integral

$$G(b,a) = A \int ... \int \int e^{iS[b,a]/\hbar} dx_1 dx_2 ... dx_{N-1} \tag{3}$$

where A is again a constant of normalisation. We do not integrate the points x_0 or x_N because they are held fixed.

Another issue with this approach is caused by the phases of each path. Distant paths with a large S will oscillate rapidly, causing issues for convergence of the sum in Eq. 1. To avoid this problem, a Wick Rotation into imaginary time is performed, giving an action in Euclidean space. This converts the exponent to a real negative exponent, such that the phase associated with each path becomes a real weighting. The Wick Rotation also converts the Lagrangian in our action functional into a Hamiltonian.

A. MCMC Computation of Path Integrals

Although the approximations and reformulations of the Feynman Path Integral makes computation easier, calculations at a useful resolution require significant computing power. To more efficiently sum over paths, a method to select paths of the largest contribution to the propagator is needed.

The method applied in this paper uses the MCMC approach. MCMC is a combination of two prior methods used for statistical modelling: Markov Chains and Monte Carlo Sampling[4]. A Markov Chain is an example of a stochastic process, a sequence of random variables $\{X_k\}$ indexed by some parameter, k[5]. The Markov Chain is a stochastic process that forms a sequence of events where the value of X_k is only influenced by the value of X_{k-1}. Monte Carlo Sampling is where independent samples are produced, with each being independent and identically distributed[4]. Together, they form the MCMC method, where the outcome of X_{k-1} influences the probability distribution of possible X_k outcomes.

The simplest generalized implementation of an MCMC method takes the form of the Metropolis-Hastings algorithm. One update from u_1 to u_2 works as follows:

- Propose a new point, u_2.
- Calculate the ratio
$$r(u_1, u_2) = \frac{h(u_2)\rho(u_2, u_1)}{h(u_1)\rho(u_1, u_2)} \tag{4}$$
where $h(u)$ is the un-normalised target distribution, $\rho(u_2, u_1)$ is the conditional probability of u_2 given u_1 and $\rho(u_1, u_2)$ is the probability of u_1 given u_2.
- Accept u_2 with the probability $\min[1, r(u_1, u_2)]$. If not accepted, $u_2 = u_1$. [4]

After many iterations of this algorithm, the values in the sequence will be distributed according to the normalised target distribution.

In the context of path optimisation, the action along the path is minimised through incremental updates to the path points. Because the path, x, is weighted by $e^{-S[x]}$, we use this exponential as the probability of the original and proposed path, analogous to $h(u_1)$ and $h(u_2)$.

B. MCMC Using Hamiltonian Dynamics

Hamiltonian dynamics allow distant proposals to be made with the probability ratio $r = 1$. To simulate Hamiltonian dynamics for point proposals, the Hamiltonian, $H(q, p)$ is introduced. It is a scalar function of the position and momentum coordinates, (q, p). Here, q is the variable we are sampling and p is a synthetic variable introduced to make the dynamics function. The Hamiltonian is expressed

$$H(q, p) = K(p) + U(q) \tag{5}$$

where K is the kinetic energy and U is the potential energy. A key property of the Hamiltonian is that it is conserved under classical mechanics. We can use Hamilton's Equations

$$\frac{dq}{dt} = \frac{\partial H}{\partial p} \tag{6}$$

$$\frac{dp}{dt} = -\frac{\partial H}{\partial q} \tag{7}$$

to describe how q and p change over time, t. These equations allow us to produce proposals for new samples, (q', p'), where q' and p' are the position and momentum of the proposed state. Because we use Hamilton's Equations to propagate forward in time to proposed states (q', p'), the Hamiltonian of the conserved, $H(q, p) = H(q', p')$.

Using a concept from statistical mechanics where the probability of a state is related to its energy by the canonical distribution [4], we can see that the Hamiltonian is therefore related to the joint probability function $P(q, p)$ of the state (q, p). The relationship between them is

$$\begin{aligned} P(q, p) &= 1/Z \exp(-H(q, p)/T) \\ &= 1/Z \exp(-K(p)/T) \exp(-U(q)/T) \end{aligned}$$

where Z is a normalising constant and T is temperature, scaled to the Boltzmann constant. Because the Hamiltonian is preserved, we can see that a proposed state, (q', p'), will have the same probability, $P(q', p')$, as the original state. Because it has the same probability, the probability ratio $r((q, p), (q', p'))$ is unity, and the point is always accepted.

The standard Hamiltonian Monte Carlo (HMC) algorithm is similar to the Metropolis Algorithm, but HMC has a more complex process for calculating proposals. The HMC implementation in this paper uses the Leapfrog integration approach[4] to propagate position and momentum forward in time. The method advances each component of momentum and position with

$$p_j(t + \epsilon/2) = p_j(t) - (\epsilon/2)\frac{\partial U}{\partial q_j}(q(t)) \tag{8}$$

$$q_j(t + \epsilon) = q_j(t) + \epsilon\frac{p_j(t + \epsilon/2)}{m} \tag{9}$$

$$p_j(t + \epsilon) = p_j(t + \epsilon/2) - (\epsilon/2)\frac{\partial U}{\partial q_j}(q(t + \epsilon)) \tag{10}$$

where p_j and q_j are the components of p and q in the j^{th} dimension, ϵ is the time step and t is the initial time of the leapfrog step. To perform the Leapfrog method, Eq. 8 is used to calculate an initial momentum half-step, Eq. 9 describes a full position update using the momentum half-step, finally Eq. 10 updates the momentum another half-step. When multiple leapfrog steps are taken, the intermediate momentum half steps cancel out, and instead we can perform full momentum steps with spacing ϵ, except at the beginning and end of the trajectory[4].

III. NUMERICAL METHODS

To implement the Metropolis Algorithm for path updates, the path was again discretised into N points with time interval δ, and the action along the path is defined by

$$S = \delta \sum_{n=1}^{N} \left(\frac{m}{2} \left(\frac{q_n - q_{n-1}}{\delta} \right)^2 + V\left(\frac{q_n + q_{n-1}}{2} \right) \right) \tag{11}$$

where q_n is position at time $n\delta$. In higher dimensions, the inner product of $q_i - q_{i-1}$ with itself was calculated to find the scalar distance squared. The algorithm to update the path is described below[6]:

- Start at a point q_i in the path, propose a new point, q_i', such that $q_i' = q_i + \alpha$. α is a small random perturbation.
- Calculate $\Delta S = S[q'] - S[q]$ where ΔS is the action change between the proposed path and the original path, denoted $S[q']$ and $S[q]$ respectively.
- Accept the proposed point, q_i' with probability $\min[1, \exp(-\Delta S)]$. If rejected, set $q_i' = q_i$.
- Move on to point q_{i+1} in path.

In this procedure, the conditional probabilities $\rho(u_2, u_1)$ and $\rho(u_1, u_2)$ were ignored. This is because the perturbations, ϵ, are symmetric about zero and stochastically independent of q_i. This is a special case of Metropolis-Hastings when $\rho(u_2, u_1) = \rho(u_1, u_2)$, and the algorithm above is simply called the Metropolis Algorithm[4]. After a path had been updated, it was recorded and put back into the algorithm to optimise it further.

To calculate the point proposals using Hamiltonian dynamics, an algorithm described by the following pseudo code was used[7].

Algorithm 1: Hamiltonian Point Proposal

```
def HMC($q$, $L$, $\epsilon$):
    initialize;
    $p \leftarrow N(0, \sqrt{m})$, $r \leftarrow \text{random}(0,1)$;
    calculate $p(t + \epsilon/2)$;
    for $n$ from 1 to $L$:
        calculate $q(t + n\epsilon)$ with latest $p$;
        if $n \neq L$:
            calculate $p(t + n\epsilon + \epsilon/2)$ with
              $q(t + n\epsilon)$;
    calculate $p(t + L\epsilon)$ with $q(t + L\epsilon)$;
    $E = \exp[H(q,p) - H(q(t+L\epsilon), p(t+L\epsilon))]$;
    if $r < E$:
        $q' = q(t + L\epsilon)$;
    else:
        $q' = q$
    return: $q'$
```

When q' is returned, it is recorded and re-entered into Algorithm 1 to obtain a new position. When calculating proposals using the Hamiltonian algorithm, single point proposals were made. So that no optimised paths were calculated using the HMC approach, only a series of almost uncorrelated points.

To calculate the probability amplitude of a particle in a potential using optimised paths, all of the point components in the paths were projected on their respective axis. Then, histogram data of point frequency was calculated for each axis, this data was normalised to produce a probability density plot. 99 histogram bins were used for each calculation, with an overall histogram width of 6-10 units. To calculate probability amplitudes with HMC point data, all recorded points were used to calculate histogram data. The components of the recorded points were also projected onto their respective axis. The histogram width and resolution was the same as for Metropolis calculations.

Both algorithms require fine-tuning of exploration parameters to achieve optimum performance. To tune the Metropolis Algorithm, the number of time points, N, the time step between them, δ, and the maximum perturbation size, α, need to be adjusted. N and δ are set to give the desired level of accuracy of path simulation, and α was tuned to give a 40%-60%, point acceptance probability, given these N and δ values. When tuning the Hamiltonian Algorithm, the number of leapfrog steps L and size of leapfrog step, ϵ, are adjusted. Increasing L allows for more distant proposals to be made, but it increases computing time for proposals. A small L number will reduce computing time per proposal, but will result in a slow exploration of the sample. A large ϵ allows more distant proposals, but also decreases the accuracy of the leapfrog algorithm, causing a lower acceptance probability. To tune the HMC algorithm, values of ϵ and L were chosen to maximize proposal acceptance, while keeping rate of exploration and high and computation time as low as possible.

IV. RESULTS

Simple calculations were initially performed to confirm the code was working correctly. For the Metropolis update approach, the 1-dimensional probability amplitude for a quantum harmonic oscillator was plotted. For the Hamiltonian algorithm, a plot of leapfrog updates was produced that confirms the leapfrog process is working correctly. Also, plotting the 3-dimensional quantum harmonic oscillator probability amplitude for the ground state was used to confirm the functioning of the whole code.

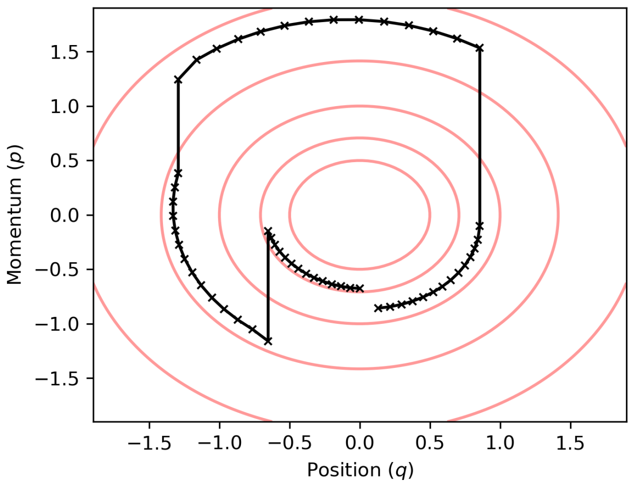

FIG. 1: A plot showing a sequence of 1 dimensional metropolis updates. A time step of 0.3 and 13 steps were used for the leapfrog proposal. Red lines illustrate curves of constant Energy.

Four leapfrog propagations are shown in Fig. 1. These were used to confirm the correct calculation of the leapfrog steps. The black plots following the curvature of the red lines indicate that the propagation is keeping the Hamiltonian constant. Also, the curve at the top of the plot covers 2 distance units, and the proposal is accepted. No changes were made to acceptance criteria to achieve this plot.

The probability amplitudes shown in Fig. 2 were calculated with 15 and 60 point paths. Increasing the resolution of the path allowed for a more accurate result with a closer

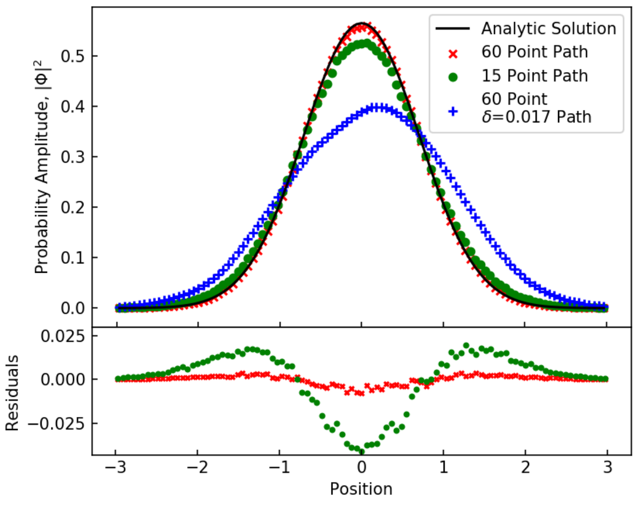

FIG. 2: Metropolis path calculation of the ground-state probability amplitude of a particle in a harmonic potential well. The color of each plot is the same in the residuals as the regular plot. Calculated against the analytic solution. Red and green plots used 100,000 thermalisation sweeps of the path with $\delta = 1.7$, blue plot used 3 million sweeps and $T_{path} = 0.017$. Residuals were calculated against the analytic solution.

fit to the analytic solution. The residuals of the plot show that the inaccuracy of the 15 point calculation is dominated a continuous deviation around the analytic solution, random errors are not as large. There is also a small continuous deviation visible in the 60 point path residuals.

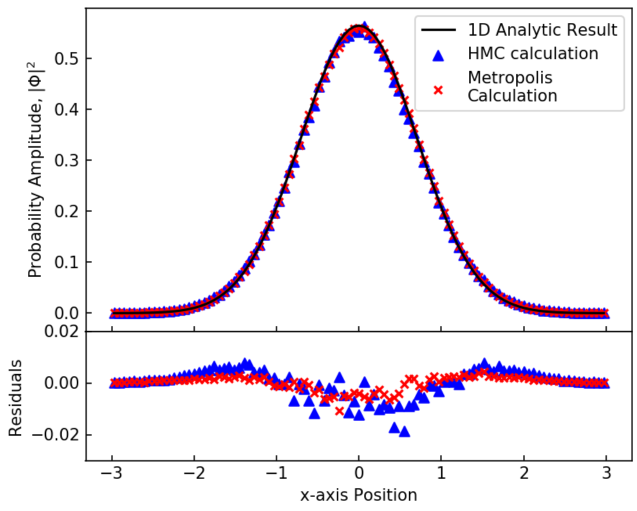

FIG. 3: A plot of the x-axis projection of the ground state probability amplitude for the 3d quantum harmonic oscillator. Results are shown for the Metropolis path optimization algorithm, and the Hamiltonian algorithm. 100, 000 thermalisation sweeps were performed with the Metropolis Algorithm, 1 million point proposals were made with the Hamiltonian Algorithm.

The numerical calculations of the wave function shown in Fig. 3 show agreement with the Schrödinger solution. Both show a very slight flattening of the probability amplitude, visible in the residuals plot. Only the x-axis projection is shown here, because the symmetric harmonic potential has very similar plots on each axis.

Calculations of the probability amplitude for the finite square well gave the results shown in Fig. 4. There is a more rapid decay outside the well boundary when a larger

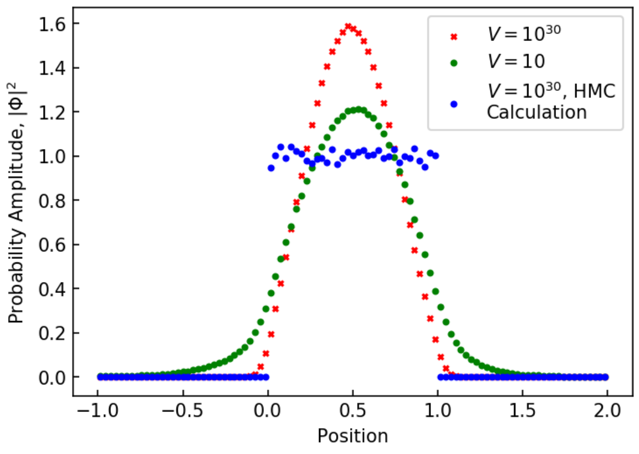

FIG. 4: Ground state probability amplitudes for a particle in a finite square well, where $V(x) = 0, 0 < x < 1$. 100,000 thermalisation sweeps were used for the green and red plots (Metropolis), with a 60 point path and $\delta = 0.017$. 1 million point updates were calculated for the HMC calculation, $\epsilon = .3$ and 14 leapfrog steps.

potential well was simulated. This qualitative result is in agreement with the solution obtained via the time independent Schrödinger equation. However, the calculation performed using the HMC approach gives an incorrect result.

Plots showing the effect of a particle in a potential well, under the influence of an electrostatic force from a constant electric field, are shown in Fig. 5. The probability amplitude modeled in both Figs. 5, 6 are for a positively charged particle with charge +1C. The plots in Fig. 5 show the expected shift of the no-field probability amplitude by a distance of -1 and 1 for the x-axis and y-axis respectively. The z-axis is left unchanged, which is the expected result, because the probabilities along each axis are independent. For the Metropolis calculation, 100,000 path updates were performed, with a 60 point path and $\delta = 1.7$. For the HMC calculation, $\epsilon = 0.355$, 18 leapfrog steps and 1 million point updates were used.

The Metropolis and HMC algorithms produced different results when modeling the time variant electric fields, shown in Fig. 6. Both methods showed a shift of centering of the probability distribution, related to the size of k relative to the path duration used in the algorithm, T_{path} and $T_{leapfrog}$. Metropolis calculations found no centering shift at values of $k = n\pi/2T_{path}$. However, shifting occurred in the same direction, from for 0¡$k < \pi/T_{path}$ and changed direction for $k > \pi/T_{path}$. The Metropolis algorithm preserved the probability distribution along the z-axis. HMC results exhibited no shift at $k = 17\pi/T_{leapfrog}$, and a large shift for $k = 17\pi/2T_{leapfrog}$. However, there was no observed periodicity to this behaviour. The HMC calculation did not preserve the probability distribution along the z-axis. The Metropolis algorithm was implemented with a 60 point path, $\delta = 1.7$ and 100,000 path updates. The HMC calculation used 18 leapfrog steps with a spacing of $\epsilon = 0.355$, and 1 million point proposals.

V. DISCUSSION

Fig. 1 is useful for refining time-step values used when simulating Hamiltonian dynamics for point proposals. This

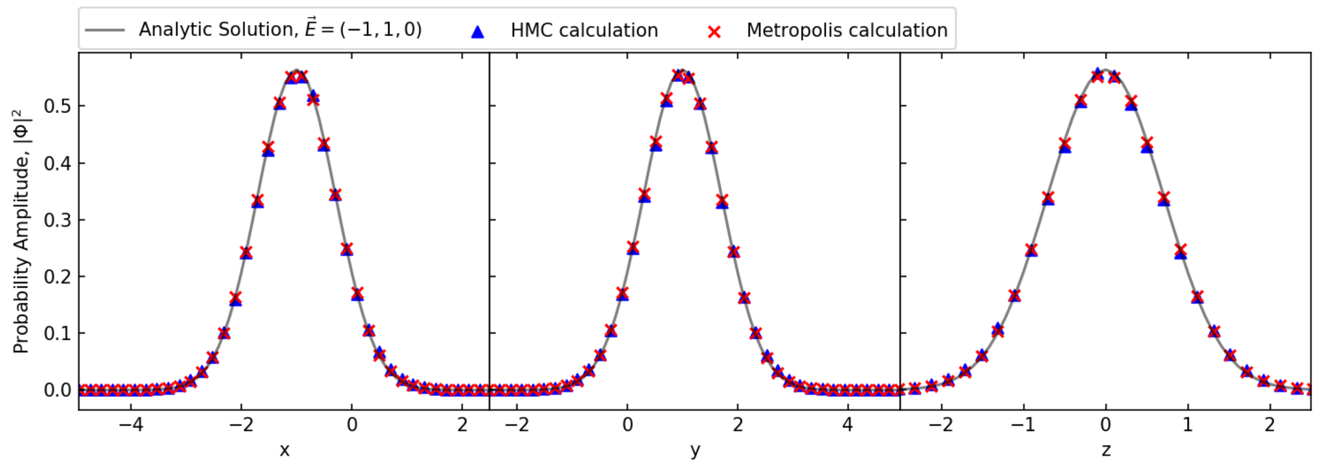

FIG. 5: Metropolis and HMC calculations of the ground state probability amplitude of a particle in a harmonic potential well, with a uniform electric field applied along the x and y axes. A field was applied in the negative x direction, and in the positive y direction. Every 2nd point is plotted to improve clarity.

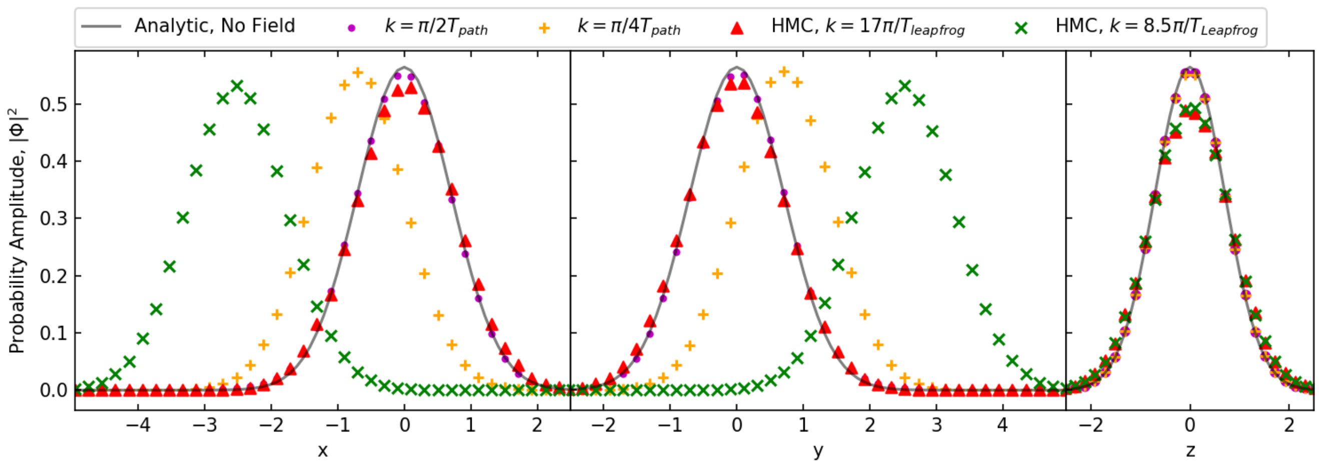

FIG. 6: Probability amplitudes of ground state particles in a harmonic potential well with a time variant, uniform electric field of the form $\vec{E}_{x,y}\sin(kt)$. Every 2nd point is plotted to improve clarity.

is because it is possible to see how larger time-steps , ϵ, explore the sample space more quickly. However, increasing the time-step increases the global error, which is the total error in the path simulation. A larger error on the path simulation will cause a variation in the Hamiltonian of the proposed point, potentially decreasing the probability of proposal acceptance. The global error along a simulated path calculated using the leapfrog method is of the order ϵ^2.[7] As the error increases with larger time-steps, plots such as those in Fig.1 are useful for finding a balance between sample exploration and accuracy of proposal simulation. Additionally, more leapfrog steps can produce distant proposals, but the proposal can circle back on itself and propose a nearby point. This is especially problematic, because it requires extra computing time to calculate more steps.

Histograms such as in Fig. 2 were used to fine-tune the path parameters to produce the most accurate simulation of the ground state probability amplitude. For the harmonic potential, this is

$$|\phi|^2 = \frac{e^{-x^2}}{\pi^{1/2}} \tag{12}$$

. Increasing the path resolution improved agreement of the Metropolis approach with the Shrödinger solution. This is because more path points can simulate paths of least action more precisely, which are the paths with the highest probability. Also, more points in the path allows there to be a higher concentration of points in the areas of high probability, because more points are selected in this region, the probability peak was sampled more accurately. This leads to a more accurate histogram with a sharp probability peak, and explains why the 15 point path calculation underestimates the probability peak. However, the same number of path points and the same time step, δ, did not accurately simulate a square finite potential well. With $\delta = 1.7$, the Metropolis approach produced an appropriately centred histogram, but there was the same probability in the finite potential regions when $V = 10$ and $V = 10^{30}$. This is not a physically realistic result, because in the Shrödinger solution, the wave function decays outside the well with decay constant $\kappa \propto \sqrt{V}$, where V is the depth of the potential well[8]. This means there should be a significantly increased decay rate outside the potential well when $V = 10^{30}$. However, $\delta = 1.7$, which was used to accurately model the square well probability, did not give an accurate approximation of the probability amplitude, seen in Fig. 2. Therefore decreasing the overall path time, so that path points are closer together, did not improve accuracy. Intermediate values of δ were tried, no value was found that models both accurately.

The plot of the ground state probability distribution in

Fig. 3 shows that the n-dimensional generalization of the Metropolis path optimisation and Hamiltonian algorithms both provide accurate results, in agreement with the Schrödinger solution. The residuals show that the Metropolis approach produces a result in closer agreement with the analytic solution, with larger random errors present in the HMC approach. It is likely that running the HMC algorithm with more repetitions would reduce this random error, producing a result similar to Metropolis.

Fig. 4 illustrates a clear advantage of the Metropolis algorithm in the case of a square well. The Hamiltonian Algorithm requires ∇U to be computed everywhere, except for a few points where $\nabla U = 0$[7]. The gradient of the potential is infinite when $x = 0, 1$, so HMC cannot be used for square-well potentials, which is a limitation if free particles need to be analysed.

The Metropolis and HMC algorithms successfully simulated the probability amplitude of a particle in an electric field and harmonic potential. This confirms that both algorithms can simulate a non-symmetric potential, as shifts in the correct magnitude and direction are present on each axis. The uniform field does not change the width or shape of the probability amplitude, because the constant field produces an effective shift in the potential. This has no effect on the shape of the probability amplitude, because the shape of the potential is the same. The correct simulation in Fig. 5 confirms that both algorithms are adequate for simulating simple asymmetric potentials.

The Metropolis and Hamiltonian algorithms calculate different probability amplitudes when an oscillating electric field is added. However, an exact solution is not available for oscillating fields, so definite comparisons cannot be made to analytic results. But there are clear errors in the HMC plot of the probability. Firstly, the electric field used to produce Fig. 5 was multiplied by $2\sin(kt)$, so the peak positions on the x and y axes should be a distance less than 2 units from the origin. The Hamiltonian calculation in Fig. 6 shows a peak at a 2.5 units from the origin, whereas the Metropolis result simulated a more realistic result between 0 and 1. A displacement less than 2 is realistic, because the peak magnitude of the field is 2, but it is less than 2 for most of the path duration. Whereas for a constant field, the magnitude is 2 for the entire path duration. Secondly, the probability amplitude along the z-axis incorrectly calculated by the HMC implementation. This is not a correct result, because the applied field does not have a z-component, so the potential is left unchanged.

The displacement of the probability amplitude exhibited a periodicity related to the oscillation frequency, k, when calculated with the Metropolis Algorithm. There was no amplitude displacement when the oscillation frequency was an integer multiple of $\pi/2T_{path}$, this is shown in Fig. 6 for $k = \pi/2T_{path}$. It was expected that no displacement would occur for $k = 2\pi/T_{path}$, because

$$\int_0^{T_{path}} \vec{E}_{x,y} \sin(2\pi t/T_{path}) dt = 0, \tag{13}$$

so there is no net field acting on a particle.

The effects of magnetic fields on charged particles is useful for the study of antimatter confinement. Here, a combination of electric and magnetic fields are used to trap antimatter particles. The simulations of constant electric fields shown above could easily be generalized to include magnetic fields. In this case, the probability amplitude of a particle in a Penning Trap [9] can be simulated. In fact, an arrangement of hyperboloidal electrodes leads to a harmonic potential well for particle trapping[10]. This method requires an alternating magnetic field to damp unwanted oscillations, the effect of which could be investigated by simulating the effect of an alternating field with the method used here.

VI. CONCLUSIONS

The Metropolis path optimization and Hamiltonian point proposal algorithms accurately simulate the Quantum Harmonic Oscillator ground state. Also, the effect of a constant electric field is calculated with both algorithms, and found to be in agreement with theoretical predictions. However, the Metropolis approach is more accurate for simulating the effect of an oscillating electric field, and the approach is applicable for a wider range of potentials, such as the finite square well.

Future investigations could analyse the performance of these algorithms in more complex potentials, with multiple potential wells. Such potentials may illustrate advantages for the Hamiltonian Algorithm. A more in-depth investigation of how time dependent fields affect Feynman paths would be of use. This would allow theoretical predictions to be made, to compare with simulated results.

References

[1] R. P. Feynman, *The Principle of Least Action in Quantum Mechanics*. PhD thesis, Princeton University, 1942.

[2] R. P. Feynman and A. R. Hibbs, *Quantum Mechanics and Path Integrals: Emended Edition (Dover Books on Physics)*. Dover Publications,31 East 2nd Street,Mineola, NY 11501-3852: Dover Publications, 2010.

[3] S. Duane, A. Kennedy, B. J. Pendleton, and D. Roweth, "Hybrid monte carlo," *Physics Letters B*, vol. 195, pp. 216–222, Sept. 1987.

[4] C. J. Geyer, "Introduction to markov chain monte carlo," in *Handbook of Markov Chain Monte Carlo (Chapman & Hall/CRC Handbooks of Modern Statistical Methods)* (S. Brooks, A. Gelman, G. Jones, and X.-L. Meng, eds.), ch. 1, 2&4 Park Square, Milton Park, Abingdon, OX14 4RN: Chapman and Hall/CRC, May 2011.

[5] R. Coleman, "What is a stochastic process?," in *Stochastic Processes*, pp. 1–5, Springer Netherlands, 1974.

[6] G. P. Lepage, "Lattice qcd for novices," 2005. arXiv:hep-lat/0506036.

[7] R. M. Neal, "Mcmc using hamiltonian dynamics," in *Handbook of Markov Chain Monte Carlo (Chapman & Hall/CRC Handbooks of Modern Statistical Methods)* (S. Brooks, A. Gelman, G. Jones, and X.-L. Meng, eds.), ch. 5, 2&4 Park Square, Milton Park, Abingdon, OX14 4RN: Chapman and Hall/CRC, May 2011.

[8] O. F. de Alcantara Bonfim and D. J. Griffiths, "Exact and approximate energy spectrum for the finite square well and related potentials," *American Journal of Physics*, vol. 74, pp. 43–48, Jan. 2006.

[9] J. Danielson, D. Dubin, R. Greaves, and C. Surko, "Plasma and trap-based techniques for science with positrons," *Reviews of Modern Physics*, vol. 87, pp. 247–306, Mar. 2015.

[10] W. Paul, "Electromagnetic traps for charged and neutral particles," *Reviews of Modern Physics*, vol. 62, pp. 531–540, July 1990.

APPENDIX A: PYTHON CODE

Listing 1: Code used to calculate point proposals with the Hamiltonian Algorithm. This code can be used for 1 and up dimensional systems.

```python
import numpy as np
m = 1.
k = np.pi/(2T)


def K(p):              # Defines synthetic kinetic energy used in leapfrog updates and
                       # metropolis accept/reject
    K = np.dot(p,p) / (2.*m)
    return K

def U(q, t):
    U = np.dot(q,q) / 2. + float(q[0]-q[1])*np.sin(k*t)*2
    return U

def dUdq(q, t):
    return q + np.array([1., -1.,0.])*np.sin(k*t)*2

def randp():
    return np.random.normal(0., np.sqrt(m), (3,))

def nDHMC(current_q, dt, nstep, t0):
    q=current_q
    p=randp()
    current_p=p
    t=t0

    p = p - (dt/2) * dUdq(q, t)  # leapfrog implementation
    t =+ dt/2
    for i in range(1, nstep):
        q = q + dt * p/m
        if i != nstep:
            p = p - dt * dUdq(q,t)
            t =+ dt

    p = p - (dt/2) * dUdq(q,t)
    t =+ dt/2

    current_U = U(current_q, t)
    current_K = K(current_p)
    proposed_U = U(q, t)
    proposed_K = K(p)

   #metropolis accept/reject step
    if np.random.uniform() < np.exp(current_U-proposed_U+current_K-proposed_K):
        return q , t
    else:
        return current_q , t

q0=np.array([0.,0.,0.]) #3D initial position
q=q0
t = #initial time
Ncf = #number of HMC updates

for i in range(Ncf):
    q, t=nDHMC(q,.355,18, 0)
    qs=np.append(qs,np.reshape(q, (3,1)),axis=1)
```

Listing 2: Code used to calculate optimised paths using the Metropolis Algorithm. This code works in n-dimensions

```
import numpy as np

m=1
timestep = 100/60
k=1

def V(qa, qb, t):
    q = (qa+qb)/2
    V = 1/2 * np.vdot(q,q) + np.sin(k * t)*(q[0]-q[1])
    return float(V)


def Action(path, dt, t0):
    Action = 0
    for i in range(1,path.shape[1]):
        q0, q1 = path[:,i-1], path[:,i]
        dq = q1-q0
        Ek =float(.5 * np.vdot(dq,dq) / (dt**2))
        t = dt*(i + i-1) + t0
        Action += dt*(Ek + V(q0, q1, t))
    return Action

def metropolis_sweep(path, t0):
    p = .85 #perturbation size
    for i in range(path.shape[1]): #iterates along length of path
        new_path = path.copy()
        new_path[:,i] = path[:,i] + 2.*p * np.random.rand(1,path.shape[0]) - p #array slice is path point
        if i == 0: # calculates change in action
            dS = Action(new_path[:,i:i+2],timestep, t0) - Action(path[:,i:i+2],timestep, t0)
        elif i == path.shape[1]-1:
            dS = Action(new_path[:,i-1:i+1],timestep, t0) - Action(path[:,i-1:i+1],timestep, t0)
        else:
            dS = Action(new_path[:,i-1:i+2],timestep, t0) - Action(path[:,i-1:i+2],timestep, t0)
        #dS = Action(new_path, timestep) - Action(path, timestep)
        if dS < 0 or np.exp(-dS) > np.random.uniform(0,1): # is th second cond. correct?
            path[:,i] = new_path[:,i]#update the path
        else:
             path[:,i]=path[:,i]#keep orig path
    return path

ndim = 3
path = np.zeros((ndim,npoints)) # initial path to be updated, ndim x npt array
t0 = 0.
print(metropolis_sweep(path, t0))

for i in range(500):
    path = metropolis_sweep(path, t0)

Ncf = #number of path updates
paths = path.copy()

for i in range(Ncf):
    path = metropolis_sweep(path, t0)
    paths = np.concatenate((paths, path), axis=1)
```