Lattice QCD in Python

Paul Lashomb
Baylor University Department of Physics
(Dated: November 24, 2017)

Preliminary elements of a Lattice QCD simulation were developed in the scripting language Python and its details are outlined within this paper. The program features a Metropolis Monte Carlo algorithm that was tested using a simple case of a one-dimensional harmonic oscillator to calculate the two-point correlation functions and, from it, the excitation energy from the ground state to the first excited state. The excitation energies showed good agreement with theory as well as with other authors [2, 5] who have performed similar work. Cubic sources and sinks in the correlation function were also explored and their details outlined within the paper. Future work will include implementing the gluonic and fermionic action in order to do elementary Lattice QCD calculations using the Metropolis Monte Carlo algorithm developed within this paper.

I. INTRODUCTION

Quantum Chromodynamics (QCD) is a quantum field theory that describes the strong interaction. In calculating quantities of interest from the theory, however, it is generally not feasible to determine analytic solutions as the coupling constant grows large at low energies [1] and must be computed numerically. Lattice QCD is a computational approach to making calculations of such quantities in QCD; specifically, it is a lattice regularization of the theory. Quarks are placed on the lattice vertices while the gluons "live" on the links between vertices. Quantities on the lattice are commonly calculated using Monte Carlo algorithms to compute the path integral of quantum mechanics which, through renormalization, these bare quantites can be related to measurements made in experiment. As outlined in this paper, a Metropolis Monte Carlo algorithm was developed in order to calculate the path integrals of quantum mechanics in the scripting language Python. This particular algorithm can be extended in the future to determine quantities on the lattice using the gluonic and fermionic actions. As a test for the method, the excitation energy from the ground state to the first excited state was determined for the simple case of a one-dimensional harmonic oscillator for both linear and cubic sources and sinks.

II. THEORY

Various quantities can be determined on the lattice through discretizing the path integral. Recall that the path integral from quantum mechanics yields the transition amplitude from an initial state x_i at time t_i to some final state x_f at time t_f is defined as [2],

$$\langle x_f, t_f | e^{-\tilde{H}(t_f - t_i)} | x_i, t_i \rangle = \int \mathcal{D}x(t) e^{-S[x]}. \quad (1)$$

There is additional information that can be extracted from the system, however, by looking at times $t_i < t < t_f$ using what are known as correlation functions. Calculations of two-point correlation functions, $G(t)$, are of common interest in Lattice QCD and are defined as [2],

$$G(t) \equiv \langle 0 | x(t_2) x(t_1) | 0 \rangle = \frac{\int \mathcal{D}x(t) x(t_2) x(t_1) e^{-S[x]}}{\int \mathcal{D}x(t) e^{-S[x]}}. \quad (2)$$

The correlation function may be interpreted as the vacuum state expectation value [3]. While this is an example of a two-point correlation function, others such as three-point correlation functions have been used, for instance, to calculate quantities such as the gluon spin contribution to the spin of the proton [4]. Such calculations tend to be computationally expensive, however, and, as such, were not considered in this work.

A harmonic potential $V = \frac{x^2}{2}$ was used and the correlation function as well as the excitation energies were both were extracted and fit. In addition, cubic sources and sinks were used to evaluate the correlation functions, as well, to examine the asymptotic behavior for different sources and sinks. In other words, an additional correlation function,

$$G(t) \equiv \langle 0 | x^3(t_2) x^3(t_1) | 0 \rangle . \quad (3)$$

was also used to determine the excitation energy.

From theory, at large time t, the excitation energy becomes [2],

$$\Delta E_n \equiv \log(G_n / G_{n+1}) \xrightarrow{\text{n large}} (E_1 - E_0) a. \quad (4)$$

The theoretical prediction for the correlation function for large t can also be determined [2] to be,

$$G(t) \xrightarrow{\text{t large}} | \langle E_0 | \tilde{x} | E_1 \rangle |^2 e^{-(E_1 - E_0)t}. \quad (5)$$

However, if one recalls from quantum mechanics of a harmonic oscillator, the position operator in terms of the raising and lowering operators is given as [6],

$$\tilde{x} = \sqrt{\frac{\hbar}{2m\omega}}(a^\dagger + a). \tag{6}$$

If we let this act on the eigenstate $|1\rangle$, we find that,

$$\langle 0|\tilde{x}|1\rangle = \sqrt{\frac{\hbar}{2m\omega}}\,\langle 0|\,(a^\dagger\,|1\rangle + a\,|1\rangle), \tag{7}$$

and from the definitions of the raising and lowering operators, $a^\dagger\,|n\rangle = \sqrt{n+1}\,|n+1\rangle$, $a\,|n\rangle = \sqrt{n}\,|n-1\rangle$, we have that,

$$\langle 0|\tilde{x}|1\rangle = \sqrt{\frac{\hbar}{2m\omega}}\,\langle 0|\,(\sqrt{2}\,|2\rangle + |0\rangle), \tag{8}$$

so by orthogonality of the eigenstates of the harmonic oscillator,

$$\langle 0|\tilde{x}|1\rangle = \sqrt{\frac{\hbar}{2m\omega}}. \tag{9}$$

Therefore, the resulting correlation function becomes,

$$G(t) \xrightarrow{\text{t large}} \frac{1}{2}e^{-t}, \tag{10}$$

as, from before, $\Delta E(t) = 1$ for the harmonic oscillator. Note that, of course, in this instance, the choice of units, $\hbar, \omega = 1$, has been made.

The proceedings of this work focus on the calculation of this path integral using a Metropolis Monte Carlo algorithm and the extraction of the excitation energy from Equation (4) for both correlation functions, $G(t)$.

III. RESULTS

The simulation was coded in the scripting language Python. The simulation emulated previous work [2, 5] and was in good agreement with their simulations. The resulting excitation energies were examined for high statistics ($N_{cf} = 10^4$). The parameters used in this particular calculation are given in Table 1.

The correlation function is shown in Figure 1. The black lines in Figure (1) correspond to the theoretical prediction. The excitation energy $\Delta E(t)$ for the action $S[x]$ is shown in Figure 2. The horizontal line indicates the theoretical prediction for the excitation energy, as, if one recalls from the harmonic oscillator of quantum mechanics, the energy level of the state n is simply given by [6],

$$E_n = (n + \frac{1}{2}), \tag{11}$$

where units are in $\hbar^{-1}\omega^{-1}$, thus the energy difference between two excited states would be,

$$\Delta E \equiv E_{n+1} - E_n = 1, \tag{12}$$

which the results from the Metropolis algorithm are in good agreement with. Note that all data points were not plotted, in spite of $N = 20$. This is due to the definition of the time slice and the periodic boundaries. Since we have that $t = 0, a, 2a, ..., (N-1)a$, then if $a = \frac{1}{2}$, as is the case for this calculation, then $t = 0, \frac{1}{2}, 1, ..., \frac{19}{2}$, and so the boundary is reached after only ten data points and, thus, a symmetric pattern will appear beyond such points due to the boundary conditions. In addition, in the plot of $\Delta E(t)$, only six data points were given as the uncertainty grows much larger for large t and, as such, were not included.

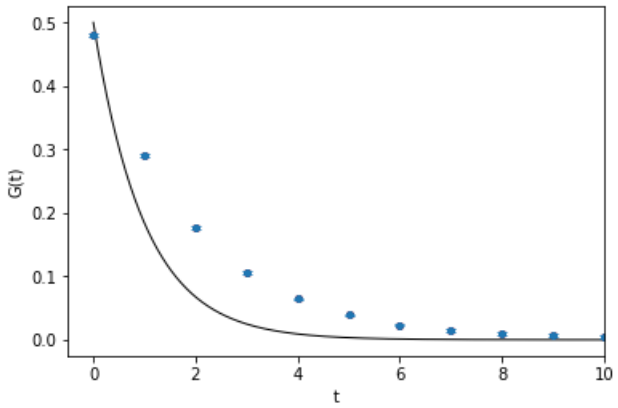

FIG. 1. Only the first ten data points are included, even though there are $N = 20$ different lattice sites. This is due to the fact that the lattice is periodic in its boundaries and there will be a symmetric mirroring of the correlation function after $t = 10$. It is not in agreement with the theoretical prediction given by the black curve, but for large t, the asymptotic behavior is the same, as expected from theory, given by Equation (10).

a	N	N_{corr}	N_{cf}	ϵ	BN	BTS
$\frac{1}{2}$	20	10	10^4	1.4	4	100

TABLE I. The simulation used a lattice spacing of a for N lattice sites. A total of N_{cf} different path configurations were used for this particular calculation with a Metropolis step size of ϵ. The correlation functions were also binned with a total number of bins BN. A total of BTS boot-strapped copies of the correlation function were also produced to determine uncertainties in the excitation energy.

The uncertainties of the excitation energies were determined by using a bootstrapping procedure. The correlation functions were first binned to reduce computational expense with a bin size of BN= 4 and then

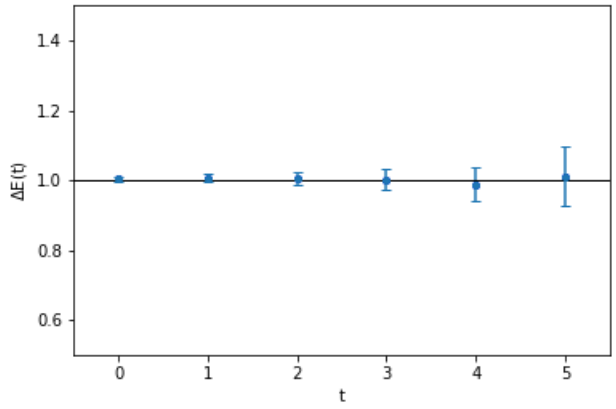

FIG. 2. While there are $N = 20$ time slices, only six data points are shown, here. This is due to the fact that the boundary conditions will mirror the data shown after $t = 10$ time slices. An additional four data points are also omitted as, for larger t, the uncertainty grows very large, albeit still in agreement with the predicted asymptotic behavior for the excitation energy $\Delta E(t)$.

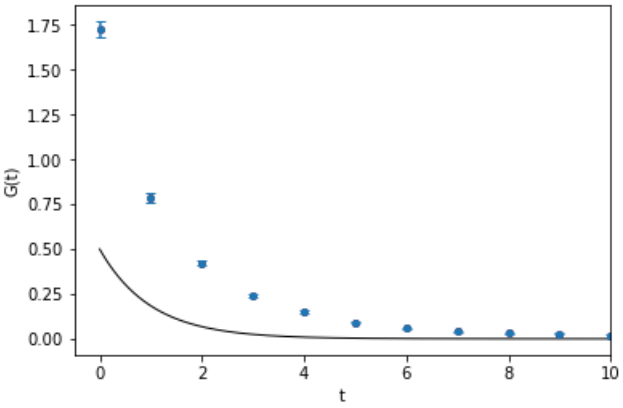

FIG. 3. The correlation function for the cubic source and sink. It has the same asymptotic behavior as predicted from theory for large t, as desired

bootstrapped copies of the binned correlation function were made. A total of BTS= 100 bootstrapped copies were made. From that, excitation energies were calculated, from which the standard deviation of the set of excitation energies could be determined. The largest uncertainty, from Figure 1 can be seen to be within 10% of the excitation energy. The uncertainties could be improved by modifying the action to include a better approximation of the kinetic energy term.

The correlation function and excitation energies were also examined for the cubic source and sink. The behavior of the correlation function can be seen in Figure 3. As seen in Figure 4, the excitation energy does not reach the asymptotic value as predicted for the same set of parameters for the simulation. The action used in the calculation was [2],

$$S_{lat}[x] = \sum_{j=1}^{N-1} [\frac{m}{2a}(x_{j+1} - x_j)^2 + aV(x_j)], \qquad (13)$$

where the approximation $\dot{x} = (x_{j+1} - x_j)/a$ was used. However, there are, of course, correction terms that could be added to this particular approximation. Thus, including a better approximation to the action could be used in order to reach better agreement with the theoretical prediction for the same set of parameters.

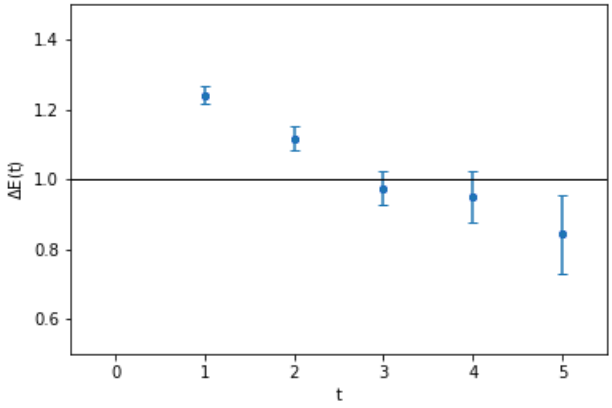

FIG. 4. The excitation energies for the correlation function from cubic sources and sinks. It does not agree with the theoretical prediction using the same set of parameters as in the linear source and sink case. This potentially could be resolved by including higher order terms in the approximation of the kinetic energy term in the action.

IV. CONCLUSION

While only preliminary work has been performed on the simulation, it does show good agreement with previous work [2, 5] and serves as a confirmation of the validity of the method. Extensions to the program include the implementation of the gluonic action in place of the harmonic oscillator potential used in the simulation. The current calculations could also be further explored, as improvements to the kinetic energy term of the action in order to increase the accuracy, as well as precision, of the predictions for the cubic source and sink could be made. This could be pursued in future work.

[1] J.R. Aitchison, J.G. Hey. *Gauge Theories in Particle Physics: A Practical Introduction Volume 1.* Boca Raton: CRC Press, 2013

[2] G.P. Lepage, "Lattice QCD for novices" arXiv hep-lat/0506036, 2005
[3] C. Morningstar, "The monte carlo method in quantum field theory", arXiv hep-lat/0702020, 2007
[4] Y-B. Yang, et al., "Glue Spin and Helicity in the Proton from Lattice QCD," Phys. Rev. Lett. 118, 102001 (2017)
[5] J. Scott (2016). *Monte Carlo Applications and Lattice QCD.* Undergraduate. University of Connecticut.
[6] J. Sakurai, J. Napolitano. *Modern quantum mechanics.* Boston: Addison-Wesley Publishing Company, 1994

Appendix A: Python Code

```
"""
Lattice_QCD

This is a Metropolis Monte Carlo algorithm for
Lattice QCD calculations
"""

import numpy as np
import matplotlib.pyplot as plt
from math import exp


'''
Declaring initial parameters and constants
'''

a = 1/2
# lattice spacing size
N = 20
# Number of lattice sites
N_COR = 10
# Number of correlations
N_CF = 10000
# Number of configurations/paths
EPS = 1.4
# Metropolis sweep step size
BIN_S = 4
# Bin size
LEN_G = int(N_CF/BIN_S)
# Length of binned G
BTS_N = 100
# Number of bootstrapping copies


'''
Randomly generates a new configuration x
and returns it if it meets the
correct criteria.
'''

def update(x):
    for j in range(0, N):
        old_x = x[j]
        old_Sj = S(j,x)
        x[j] += np.random.uniform(-EPS,EPS)
        dS = S(j,x) - old_Sj
        if dS > 0 and exp(-dS) <
        np.random.uniform(0,1):
            x[j] = old_x
    return x


'''
Defining the action S for harmonic
oscillator
'''
def S(j,x):
    jp = (j+1)%N
    jm = (j-1)%N
    return a*x[j]**2/2 + x[j]*(x[j]
    -x[jp]-x[jm])/a

'''
Computes the correlation function for a
time slice for one configuration
'''

#def compute_G(x,n):
#    g = 0;
#    for j in range(0, N):
#        g = g + x[j]*x[(j+n)%N]
#    return g/N

'''
Computes the correlation function using
cubic sources and sinks. Comment out
one compute_G in order to test the other
correlation function so that only
one is running.
'''

def compute_G(x,n):
    g = 0;
    for j in range(0, N):
        g = g + (x[j]**3)*(x[(j+n)%N]**3)
    return g/N


'''
Paths are randomly generated using the
update(x) and the correlation functions
are calculated for each path and time
slice and stored in a 2D array G
'''

def MCpaths(x,G):

    for j in range(0,N):
#Empties array
        x[j]=0;
    for j in range(0, 5*N_COR):
```

```
#Thermalizes array
        update(x)
    for alpha in range(0, N_CF):
#Calculates G[alpha][n]
        for j in range(0, N_COR):
            update(x)
        for n in range(0, N):
            G[alpha, n] =
            compute_G(x,n)


'''
Creates binned copies of G of a
specified binsize. Notice that if
binsize=N, then since the dimensions
of G is N_cf*N and for the binned
version it is (1/binsize)*N_cf*N,
then if binsize = N_cf, then one
simply obtains a matrix of size N
which just corresponds to the average
of each time slice over all alpha
configurations.
'''

def binning(G):
    G_binned = []

    for i in range(0, len(G), BIN_S):
        sumb_G = 0

        for j in range(BIN_S):
            sumb_G += G[i+j]
        G_binned.append(sumb_G/BIN_S)

    return G_binned


'''
Creates bootstrapped copies of the
binned G.
'''

def bootstrap(G):
    G_bootstrap = []
    for i in range(LEN_G):
        G_bootstrap.append(
        G[np.random.randint(0, LEN_G)])
    return G_bootstrap



'''
Calculates, for each time slice, G and
averages over all configurations for
each time slice and stores in an array
cf_aveG
'''
```

```
def ave_t_G(G):
    t_aveG = []

    for n in range(0, N):
        aveG = 0
        for alpha in range(LEN_G):
            aveG += G[alpha][n]
        aveG /= LEN_G
        t_aveG.append(aveG)
    return t_aveG



'''
Calculates the difference excitation
energy delta E(t) for each time slice t
and the uncertainty in G and delta E(t)
and stores each as an array
'''


def stdCalc_dE(ave_tot_G, btsp):
    unc_dE = []
    unc_G = []
    dE = []

    for n in range(N):
        err_1 = []
        err_2 = []
        dE.append((1/a)*np.log(np.abs(
        ave_tot_G[n]/ave_tot_G[(n+1)%N])))

        for beta in range(BTS_N):
            err_1.append((1/a)*(np.log(
            np.abs(btsp[beta][n] /
            btsp[beta][(n + 1)%N]))))
            err_2.append(btsp[beta][n])

        unc_dE.append(np.std(err_1))
        unc_G.append(np.std(err_2))

    return unc_dE, unc_G, dE



'''
Declaration of Main function
'''


def main():

    '''
    Defining path x as an array and the
    correlation function as a 2D array
    '''

    x = np.zeros(N);
```

```
G_cor = np.zeros((N_CF, N));
btsp = []
ave_tot_G = []


##Creates the correlation functions
for each time slice and configuration

MCpaths(x, G_cor)


binnedG = binning(G_cor)


for _ in range(BTS_N):
    btsp.append(ave_t_G(bootstrap(binnedG)))


ave_tot_G = ave_t_G(binnedG)

unc_dE, unc_G, dE = stdCalc_dE(ave_tot_G,
btsp)

t = np.arange(0.0, 20.0, 0.01)
exp_t = (1/2)*np.exp(-t)

plt.figure()
plt.plot(range(0, N), ave_tot_G, 'ro',
markersize = 4)
plt.plot(t, exp_t, lw=1, color = 'Black')
plt.errorbar(range(0, N), ave_tot_G,
yerr=unc_G, fmt ='o', markersize = 4,
capsize = 3)
plt.xlim(-.5, 10)
plt.xlabel('t')
plt.ylabel('G(t)')
plt.show()


#
#
#
plt.plot(range(0, N), dE, 'bo',
markersize = 4)
plt.errorbar(range(0, N), dE, yerr = unc_dE,
fmt = 'o', markersize = 4, capsize = 3)
plt.ylabel('$\Delta$E(t)')
plt.xlabel('t')
plt.ylim(0.5, 1.5)
plt.xlim(-.5, 5.5)
plt.plot((-.5, 6), (1, 1), lw = 1,
color = 'Black')
plt.show()


if __name__ == "__main__":
    main()
```