Solution of Selected Exercises from *Lattice QCD for Novices*

Luca Brunelli

Physics Department, University of Bologna

Bologna, September 11th 2023

Table of Contents

Outline

This project consists in the study of the paper *Lattice QCD for Novices* by G. Peter Lepage of 2005 [1], and the solution of selected exercises. Here is how it will be organized:

1) Introduction to numerical path integration with a simple Quantum Mechanics (QM) example.
2) Monte Carlo (MC) techniques for the evaluation of path integrals in QM.
3) Introduction to Lattice QCD (discretization, actions, improvements).
4) Implementation of an algorithm to compute simple Wilson Loops using MC.

Path Integrals in QM: Introduction

Consider the propagator in 1d Quantum Mechanics from initial state $|x_i\rangle$ to final state $|x_f\rangle$ of a system with Hamiltonian $\hat{H}$ between times t_i and t_f. This can be expressed through a path integral in Euclidean time:

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

where we integrate over the set of all paths:

$$\{x(t)\,|\,t \in [t_i, t_f],\, x(t_i) = x_i,\, x(t_f) = x_f\} \tag{2}$$

and we have an Euclidean action S:

$$S[x] = \int_{t_i}^{t_f} \left[m\frac{\dot{x}^2(t)}{2} + V(x(t)) \right] \mathrm{d}t \tag{3}$$

Moreover, setting $T = t_f - t_i$ and $x_i = x_f \equiv x$:

$$\langle x | e^{-\hat{H}T} | x \rangle = \sum_n \langle x | E_n \rangle\, e^{-E_n T}\, \langle E_n | x \rangle \xrightarrow{T \to \infty} e^{-E_0 T}\, |\langle x | E_0 \rangle|^2 \tag{4}$$

Path Integrals in QM: Discretization (1)

To evaluate path integrals numerically, we first of all discretize the continuous function $x(t)$, specifying its value only on a number N of equally-spaced nodes $x(t_j) \equiv x_j$, where:

$$t_j = t_i + aj \quad \text{where } a = \frac{T}{N} \text{ and } j = 0, ..., N \tag{5}$$

So, we have a $(N+1)$-dimensional vector $x = (x_0, x_1, ..., x_N)$. Therefore, the path integral becomes a multi-dimensional integral over the values of the x_j. First, we must impose the boundary conditions:

$$x_0 = x_N \equiv x \tag{6}$$

so that:

$$\int \mathcal{D}x(t) \mapsto A \int \mathrm{d}x_1 ... \mathrm{d}x_{N-1} \tag{7}$$

where A is a normalization constant that in our case is given by:

$$A = \left(\frac{m}{2\pi a}\right)^{N/2} \tag{8}$$

Path Integrals in QM: Discretization (2)

Now we have to discretize the action over the nodes x_j. To do that, we use the approximation of the Euclidean action on a discrete lattice:

$$S_{Lat}[x] = \sum_{j=1}^{N-1} \left[\frac{m}{2a} (x_{j+1} - x_j)^2 + aV(x_j) \right] \tag{9}$$

Depending on a, the discretization of the first derivative with a finite difference can be quite brutal as is. The most straightforward way to compute path integrals numerically is just:

$$\int \mathcal{D}x(t)\, e^{-S[x]} \mapsto A \int \mathrm{d}x_1 ... \mathrm{d}x_{N-1}\, e^{-S_{Lat}[x]} \tag{10}$$

where S_{Lat} is given in (9).

Path Integrals in QM: Implementation (1)

Let us start displaying the application of this method. We consider a very simple quantum system: the Harmonic Oscillator. This has potential ($m = 1$):

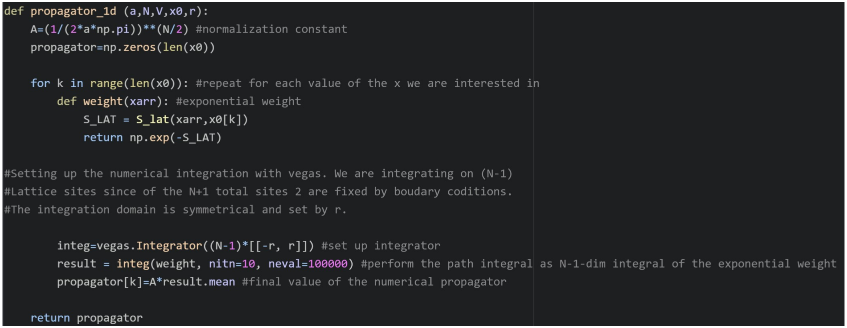
$$V(x) = \frac{x^2}{2} \tag{11}$$

To evaluate the path integral propagator of this system, we follow the procedure illustrated above. We use a numerical integrator called *vegas* [2]. The propagator of this simple system can be computed analytically to give:

$$\langle x|e^{-\hat{H}T}|x\rangle \simeq e^{-E_0 T}|\langle x|E_0\rangle|^2 \quad \text{with } \langle x|E_0\rangle = \frac{e^{-x^2/2}}{\pi^{1/4}} \tag{12}$$

and $E_0 = 1/2$.

Path Integrals in QM: Implementation (2)



```
def propagator_1d (a,N,V,x0,r):
    A=(1/(2*a*np.pi))**(N/2) #normalization constant
    propagator=np.zeros(len(x0))

    for k in range(len(x0)): #repeat for each value of the x we are interested in
        def weight(xarr): #exponential weight
            S_LAT = S_lat(xarr,x0[k])
            return np.exp(-S_LAT)

#Setting up the numerical integration with vegas. We are integrating on (N-1)
#Lattice sites since of the N+1 total sites 2 are fixed by boudary coditions.
#The integration domain is symmetrical and set by r.

        integ=vegas.Integrator((N-1)*[[-r, r]]) #set up integrator
        result = integ(weight, nitn=10, neval=100000) #perform the path integral as N-1-dim integral of the exponential weight
        propagator[k]=A*result.mean #final value of the numerical propagator

    return propagator
```

Figure: Code for the numerical integration of the harmonic oscillator propagator in 1d. The range of integration is between $-r$ and r. This function computes the numerical propagator for a certain range of values of x specified by the array x0

Path Integrals in QM: Implementation (3)

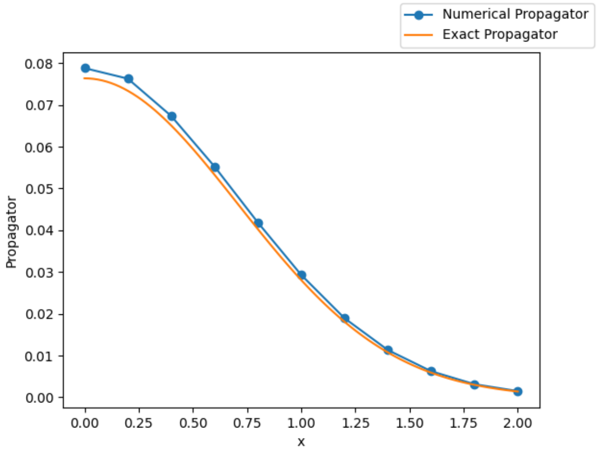


Figure: Graph of the propagator for the harmonic oscillator in 1d. Used parameters: a=0.5, N = 8, m=1.

Path Integrals in QM: Implementation (4)

We can also try with a different potential, like a quartic one: $V(x) = x^4/2$. We get:

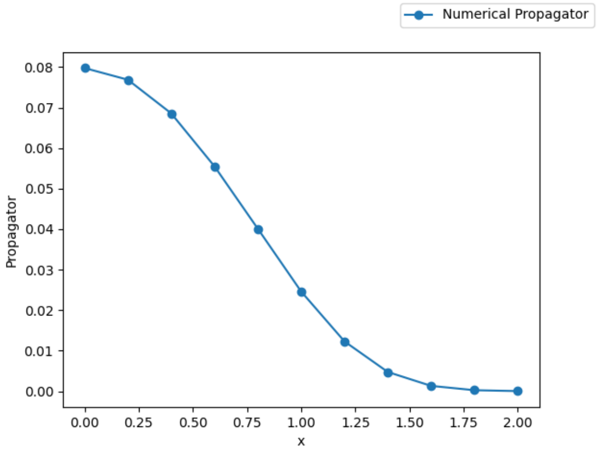


Figure: Graph for the propagator of the system with potential $V(x) = x^4/2$. Used parameters: a=0.5, N=8

Generic Path Integrals (1)

We can also compute the propagator for excited states with path integrals. From a purely theoretical point of view, we need to evaluate:

$$\langle x(t_2)|x(t_1)\rangle = \frac{\int \mathcal{D}x(t)x(t_2)x(t_1)e^{-S[x]}}{\int \mathcal{D}x(t)e^{-S[x]}} \tag{13}$$

Rewriting the expression using energy eigenstates gives:

$$\langle x(t_2)|x(t_1)\rangle = \frac{\sum e^{-E_nT}\,\langle E_n|\hat{x}e^{-(\hat{H}-E_n)t}\hat{x}|E_n\rangle}{\sum e^{-E_nT}} \tag{14}$$

where we set $t = t_2 - t_1$. If we suppose $t \ll T$, we can keep only the groundstate:

$$G(t) := \langle x(t_2)|x(t_1)\rangle \simeq \langle E_0|\hat{x}e^{-(\hat{H}-E_0)t}\hat{x}|E_0\rangle \tag{15}$$

Generic Path Integrals (2)

Taking t sufficiently large, in our harmonic oscillator example, yields:

$$G(t) \simeq |\langle E_0|\hat{x}|E_1\rangle|^2 e^{-\Delta E t} \tag{16}$$

where $\Delta E = E_1 - E_0$. Now, discretizing the time as before, one can extract ΔE as:

$$a\Delta E \simeq \ln\left(\frac{G(t)}{G(t+a)}\right) \tag{17}$$

for large t. More generally, given any functional $\Gamma[x]$, expectation values of physical observables can be computed as:

$$\langle\Gamma[x]\rangle = \frac{\int \mathcal{D}x(t)\Gamma[x]e^{-S[x]}}{\int \mathcal{D}x(t)e^{-S[x]}} \tag{18}$$

This quantity may still be computed using numerical integration. However, if we enlarge the number of dimensions of our lattice, the computational cost becomes very high.

Monte Carlo Average (1)

Notice that the expression of $\langle \Gamma[x] \rangle$ is a weighted average over paths $x(t)$ with weights $e^{-S[x]}$. Therefore, we can employ a probabilistic method, called *Monte Carlo* method to evaluate it. Let's illustrate the idea behind it. We generate a huge number N_{cf} of discretized path configurations:

$$x^\alpha = (x_0^\alpha, ..., x_N^\alpha) \quad \text{with } \alpha = 1, ..., N_{cf} \tag{19}$$

which are distributed according to the probability $P[x^\alpha]$ such that:

$$P[x^\alpha] \propto e^{-S[x^\alpha]} \tag{20}$$

and approximate the weighted average of Γ with an unweighted average over these random configurations:

$$\langle \Gamma[x] \rangle \simeq \bar{\Gamma} = \frac{1}{N_{cf}} \sum_{\alpha=1}^{N_{cf}} \Gamma[x^\alpha] \tag{21}$$

Monte Carlo Average (2)

$\bar{\Gamma}$ is called *Monte Carlo estimator* of $\langle \Gamma \rangle$, and being a statistical estimator, carries an uncertainty, which we can quantify as:

$$\sigma^2_{\bar{\Gamma}} = \frac{\overline{\Gamma^2} - \bar{\Gamma}^2}{N_{cf}} \tag{22}$$

Since the numerator is roughly independent from N_{cf}, the uncertainty scales as $1/\sqrt{N_{cf}}$.
But how do we numerically generate the configurations x^α distributed as we want?

Metropolis Algorithm (1)

The most straightforward way to do this is through *Metropolis Algorithm*. The algorithm proceeds as follows:

i Start with a generic configuration x^0 for the lattice (this can also be all zeros)

ii For each site x_j generate a random number $\zeta \in [-\epsilon, \epsilon]$ and modify $x_j \mapsto x_j + \zeta$

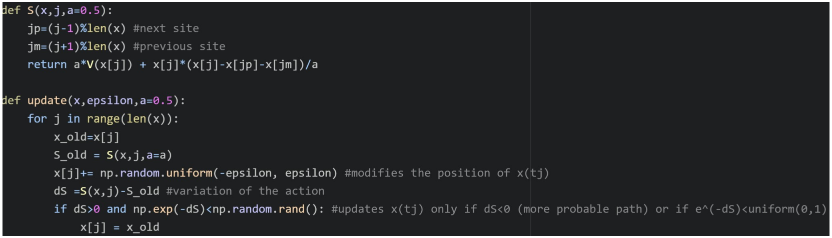
iii Compute the variation of the lattice action ΔS caused by this modification

iv If $\Delta S < 0$ keep the updated value for x_j and move to the next site; otherwise move to point v

v If $\Delta S > 0$, generate uniformly a random number $q \in [0, 1]$ and check whether $e^{-\Delta S} > q$. If it is, keep the updated value of x_j, otherwise restore the old value before point ii. Move to the next site

Metropolis Algorithm (2)

After completing a whole swipe in the lattice, we collect the new lattice configuration in the vector x^1, and repeat the process. My implementation for the update is shown below.



```
def S(x,j,a=0.5):
    jp=(j-1)%len(x) #next site
    jm=(j+1)%len(x) #previous site
    return a*V(x[j]) + x[j]*(x[j]-x[jp]-x[jm])/a

def update(x,epsilon,a=0.5):
    for j in range(len(x)):
        x_old=x[j]
        S_old = S(x,j,a=a)
        x[j]+= np.random.uniform(-epsilon, epsilon) #modifies the position of x(tj)
        dS =S(x,j)-S_old #variation of the action
        if dS>0 and np.exp(-dS)<np.random.rand(): #updates x(tj) only if dS<0 (more probable path) or if e^(-dS)<uniform(0,1)
            x[j] = x_old
```

Figure: Code for the implementation of the lattice action S for node x_j and the update function

Metropolis Algorithm (3)

Clearly, if we started from a very peculiar configuration like $x^0 = (0, ..., 0)$, this and the contiguous configurations would be not quite random. Therefore, one usually updates the lattice several times before starting storing the configurations in x^α. This process is called *thermalization*. Moreover, just one update in between two contiguous configurations would make them too similar to each other. This is why one usually updates the lattice N_{cor} times between each successive configuration is kept.

Monte Carlo Evaluation of Path Integrals (1)

Let us apply the Monte Carlo procedure we have seen above to some concrete examples. First, let us compute the MC average of the operator $G(t)$ for the harmonic oscillator we defined above. Discretizing the time, we can express it as:

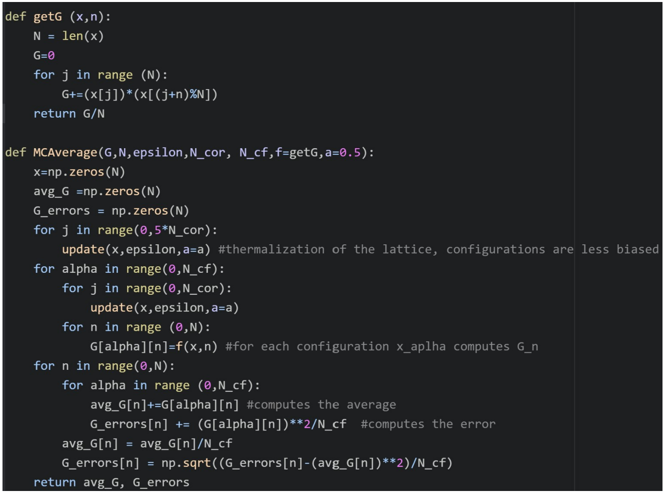
$$G_n = \frac{1}{N} \sum_j \left\langle x_{(j+n) mod N} x_j \right\rangle \tag{23}$$

for all $n = 0, ..., N-1$. From this calculation we can then extract the energy gap $\Delta E(t)$ which gets discretized as:

$$\Delta E_n = \frac{1}{a} \ln \left(\frac{G_n}{G_{n+1}} \right) \tag{24}$$

And we can compare the result with the analytical expectation of $\Delta E = 1$ for $t \to \infty$.

Monte Carlo Evaluation of Path Integrals (2)

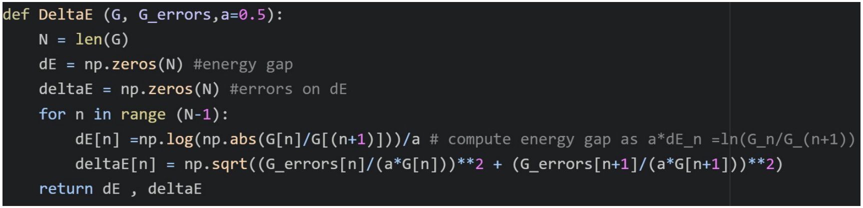


```python
def getG (x,n):
    N = len(x)
    G=0
    for j in range (N):
        G+=(x[j])*(x[(j+n)%N])
    return G/N

def MCAverage(G,N,epsilon,N_cor, N_cf,f=getG,a=0.5):
    x=np.zeros(N)
    avg_G =np.zeros(N)
    G_errors = np.zeros(N)
    for j in range(0,5*N_cor):
        update(x,epsilon,a=a) #thermalization of the lattice, configurations are less biased
    for alpha in range(0,N_cf):
        for j in range(0,N_cor):
            update(x,epsilon,a=a)
        for n in range (0,N):
            G[alpha][n]=f(x,n) #for each configuration x_aplha computes G_n
    for n in range(0,N):
        for alpha in range (0,N_cf):
            avg_G[n]+=G[alpha][n] #computes the average
            G_errors[n] += (G[alpha][n])**2/N_cf  #computes the error
        avg_G[n] = avg_G[n]/N_cf
        G_errors[n] = np.sqrt((G_errors[n]-(avg_G[n])**2)/N_cf)
    return avg_G, G_errors
```

Figure: Functions getG computing G_n and MCAverage which computes the average of a function f. The lattice is thermalized and then we compute the average over the configurations and the corresponding statistical error.

Monte Carlo Evaluation of Path Integrals (3)



```python
def DeltaE (G, G_errors,a=0.5):
    N = len(G)
    dE = np.zeros(N) #energy gap
    deltaE = np.zeros(N) #errors on dE
    for n in range (N-1):
        dE[n] =np.log(np.abs(G[n]/G[(n+1)]))/a # compute energy gap as a*dE_n =ln(G_n/G_(n+1))
        deltaE[n] = np.sqrt((G_errors[n]/(a*G[n]))**2 + (G_errors[n+1]/(a*G[n+1]))**2)
    return dE , deltaE
```

Figure: Function computing ΔE_n and the statistical uncertainty via error propagation. Note that in the function dE[n] is what we called ΔE_n, while ΔE is reserved for the statistical error.

Monte Carlo Evaluation of Path Integrals (4)

These are my results for the harmonic oscillator

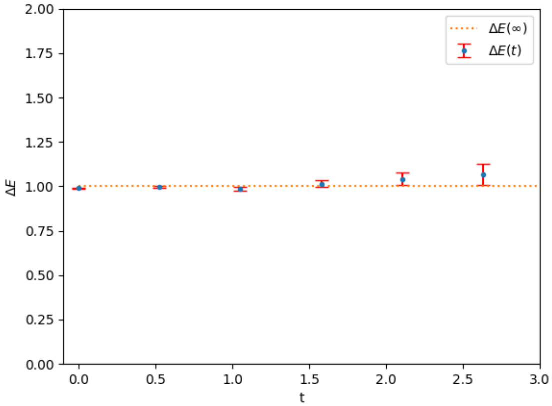


Figure: ΔE_n for the harmonic oscillator computed through MC average, using parameters: Ncf $= 10^4$, Ncor $= 20$, epsilon $= 1.4$, a $= 0.5$, N $= 20$

Monte Carlo Evaluation of Path Integrals (5)

We can modify the functional G. For example here I considered the case with *cubic* source and well:

$$G_3(t) = \frac{1}{N}\sum_j \langle x^3(t_j+t)x^3(t_j)\rangle \tag{25}$$

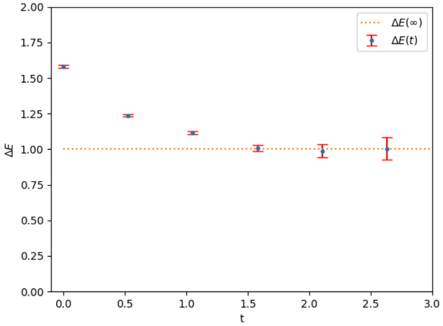


Figure: Plot of ΔE_n for the harmonic oscillator with cubic sources. Notice that it converges to the asymptotic value at higher values of n. Parameters used: Ncf = 10^4, Ncor = 20, epsilon = 1.4, a = 0.5, N = 20

Binning and Bootstrapping (1)

There are some ways to refine the MC estimate: *binning and bootstrapping*.
If we run simulations with low values of N_{cor}, contiguous configurations will be closely related. This would bring to a wrong estimate of $\langle \Gamma \rangle$ and the corresponding statistical error. One way to deal with it is through *binning*. At the end of the simulation, we'll have a large number of configurations x^α and the corresponding G^α, which will be averaged. However, to save space and reduce correlation between contiguous G^α, we can gather the estimates in different *bins* of fixed dimension, say b and only keep the averages $\overline{G}^\beta$ in each bin:

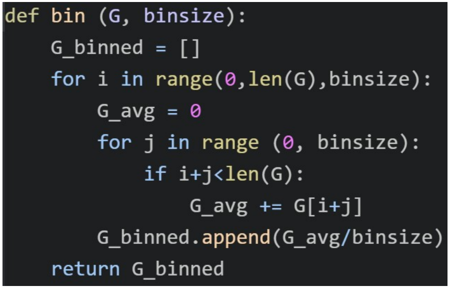
$$\overline{G}^1 = \frac{G^1 + ... + G^b}{b} \tag{26}$$

$$\overline{G}^2 = \frac{G^{b+1} + ... + G^{2b}}{b} \tag{27}$$

$$\vdots$$

Binning and Bootstrapping (2)

The binning procedure reduces the correlation between contiguous values of $\overline{G}^{\beta}$ and can lead to better estimation of errors. Moreover it saves memory space for simulations with a high number of configurations N_{cf}.



```python
def bin (G, binsize):
    G_binned = []
    for i in range(0,len(G),binsize):
        G_avg = 0
        for j in range (0, binsize):
            if i+j<len(G):
                G_avg += G[i+j]
        G_binned.append(G_avg/binsize)
    return G_binned
```

Figure: Code for the function that performs the binning of an array G in bins of size binsize

Binning and Bootstrapping (3)

Here is the comparison between highly-correlated runs, one done with classical MC average, and one using binning.

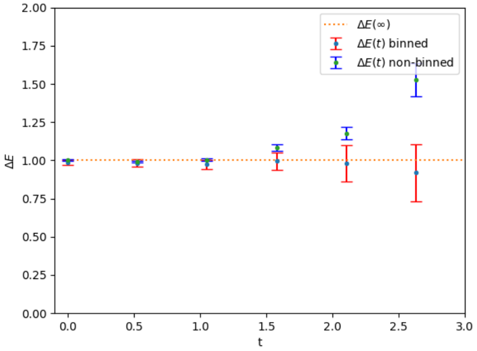


Figure: Comparison between ΔE_n obtained with standard MC average (green-blue dots) and including binning (blue-red dots). The parameter Ncor=1 here, while the other parameters used are: Ncf $= 10^4$, epsilon $=$ 1.4, a $= 0.5$, N $= 20$, binsize $= 20$

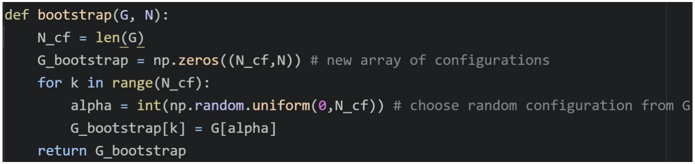
Binning and Bootstrapping (4)

To see whether the estimator is reliable, one may redo the same simulation hundreds of times and average the results. However, this is computationally very demanding. Therefore, we can use a different method to produce copies of G^α, called *bootstrapping*. Here is how it plays out:

1. Generate a random integer γ between 1 and N_{cf}
2. Keep G^γ as the first configuration of the bootstrap copies and repeat N_{cf} times.

At the end we'll have a new set of configurations $\tilde{G}^\alpha$ that is made exclusively by random copies of G^α. Repeating this N_{boot} times and averaging over all the iterations gives a better estimator of $\langle\Gamma\rangle$, and it is almost zero-cost.

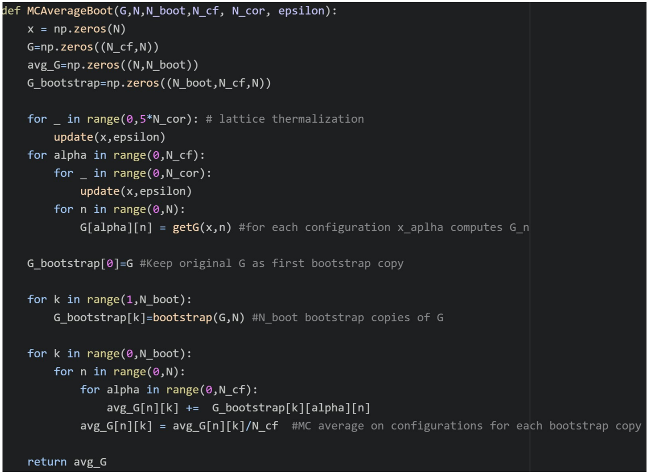
Binning and Bootstrapping (5)



```python
def bootstrap(G, N):
    N_cf = len(G)
    G_bootstrap = np.zeros((N_cf,N)) # new array of configurations
    for k in range(N_cf):
        alpha = int(np.random.uniform(0,N_cf)) # choose random configuration from G
        G_bootstrap[k] = G[alpha]
    return G_bootstrap
```

Figure: Code for the function performing a bootstrap copy of G

Binning and Bootstrapping (6)



```python
def MCAverageBoot(G,N,N_boot,N_cf, N_cor, epsilon):
    x = np.zeros(N)
    G=np.zeros((N_cf,N))
    avg_G=np.zeros((N,N_boot))
    G_bootstrap=np.zeros((N_boot,N_cf,N))

    for _ in range(0,5*N_cor): # lattice thermalization
        update(x,epsilon)
    for alpha in range(0,N_cf):
        for _ in range(0,N_cor):
            update(x,epsilon)
        for n in range(0,N):
            G[alpha][n] = getG(x,n) #for each configuration x_aplha computes G_n

    G_bootstrap[0]=G #Keep original G as first bootstrap copy

    for k in range(1,N_boot):
        G_bootstrap[k]=bootstrap(G,N) #N_boot bootstrap copies of G

    for k in range(0,N_boot):
        for n in range(0,N):
            for alpha in range(0,N_cf):
                avg_G[n][k] +=  G_bootstrap[k][alpha][n]
            avg_G[n][k] = avg_G[n][k]/N_cf  #MC average on configurations for each bootstrap copy

    return avg_G
```

Figure: Code for the function computing the MC average of G including bootstrap. The original copy of G^{α} is kept as first bootstrap copy.

Binning and Bootstrapping (7)

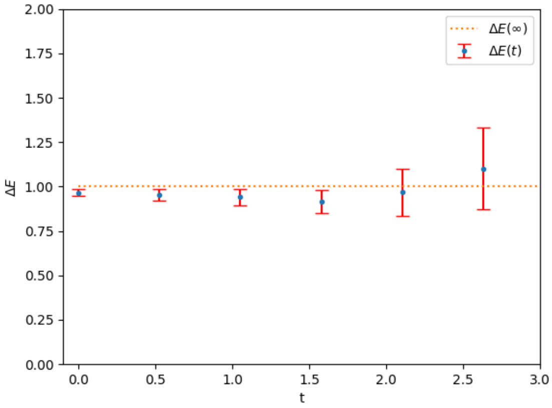


Figure: Plot of the ΔE_n obtained through a bootstrap average with Nboot = 100. This time the errorbars are computed as statistical errors on the different copies of ΔE for each bootstrap copy. Parameters used: Ncf = 10^4, epsilon = 1.4, a = 0.5, N = 20, Ncor = 20

Improved Action in QM (1)

As stated beforehand, the discretization of the action on the lattice may generate some problems linked to the approximation of the derivative with a finite difference. Now, integrating by parts the action $S[x]$ we can cast it in the form:

$$S[x] = \int_{t_i}^{t_f} \left[-\frac{m}{2}\ddot{x}(t)x(t) + V(x(t)) \right] \mathrm{d}t \tag{28}$$

Now, discretizing the time, we have to approximate the second time derivative with a finite difference. The first thing we can do is simply use the central difference approximation of order $\mathcal{O}(a^2)$:

$$\ddot{x}(t_j) \mapsto \Delta_2\, x_j = \frac{x_{j+1} - 2x_j + x_{j-1}}{a^2} \tag{29}$$

This discretization of the action carries an error of order $\mathcal{O}(a^2)$. This can be improved using an *improved discretization.*

Improved Action in QM (2)

we want to move to a more refined discretization of the second derivative, so we include also a next-to-leading order term:

$$\ddot{x}(t_j) \mapsto \left(\Delta_2 - a^2 \frac{(\Delta_2)^2}{12}\right) x_j \tag{30}$$

where:

$$\left(\Delta_2 - a^2 \frac{(\Delta_2)^2}{12}\right) x_j = \frac{1}{a^2}\left[\frac{-x_{j+2} + 16x_{j+1} - 30x_j + 16x_{j-1} - x_{j-2}}{12}\right] \tag{31}$$

which carries an error of order $\mathcal{O}(a^4)$. Thus the improved action becomes:

$$S_{imp} = \sum_{j=0}^{N-1} a\left[-\frac{m}{2}x_j\left(\Delta_2 - a^2\frac{(\Delta_2)^2}{12}\right)x_j + V(x_j)\right] \tag{32}$$

Improved Action in QM (3)

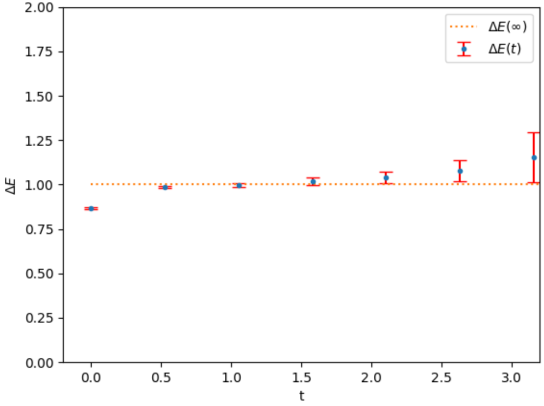


Figure: Graph of the ΔE_n computed using the improved action described above. Notice how the solutions approach the asymptotic value from below. Binning technique was used. Used parameters: Ncf = 10^4, epsilon = 1.4, a = 0.5, N = 20, Ncor = 20, Nbin = 100

Improved Action: Equations of Motion (1)

Before moving to the introduction to Lattice QCD, let us see a further application of the improved action. Given the euclidean-time action S of a system, we can find the classical equations of motion by the least action principle:

$$\frac{\delta S}{\delta x} = 0 \tag{33}$$

which yields to:

$$m\ddot{x} = \frac{dV}{dx} \tag{34}$$

Now, setting $m = 1$ and discretizing the second derivative at first order, we find:

$$\Delta_2 x_j = \frac{dV(x_j)}{dx_j} \tag{35}$$

We consider the potential for the harmonic oscillator of frequency ω_0:

$$V(x) = \frac{\omega_0^2}{2} x^2 \tag{36}$$

Improved Action: Equations of Motion (2)

Upon discretization, the differential operator of the second derivative becomes a *linear operator* acting on a N-dimensional vector $x = (x_0, ..., x_{N-1})$ with periodic boundary conditions. The equation can be read as:

$$(x_{j-1} + x_{j+1}) - (2 + (a\omega_0)^2)x_j = 0 \tag{37}$$

This can be cast in matrix form very simply:

$$\begin{pmatrix} A(\omega_0) & 1 & \dots & 0 & 0 \\ 1 & A(\omega_0) & 1 & \dots & 0 \\ 0 & 1 & A(\omega_0) & \dots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & A(\omega_0) \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-2} \\ x_{N-1} \end{pmatrix} = \begin{pmatrix} -x_{-1} \\ 0 \\ \vdots \\ 0 \\ x_N \end{pmatrix} \tag{38}$$

where $A(\omega_0) = -(2 + (a\omega_0)^2)$

Improved Action: Equations of Motion (3)

Looking for plane-wave solutions of the form $x_j = e^{-\omega a j}$, we see ω goes as:

$$\omega^2 = \omega_0^2 \left(1 - \frac{(a\omega_0)^2}{12}\right) + \mathcal{O}((a\omega_0)^4) \qquad (39)$$

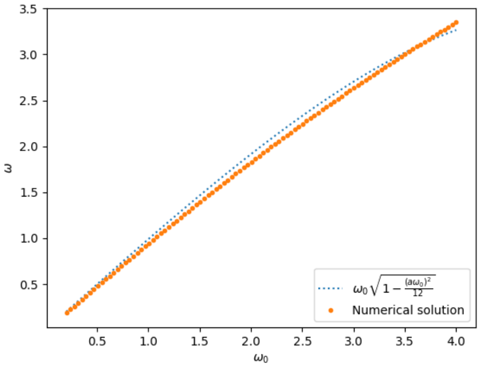


Figure: Solution of the equation of motion for unimproved action as a function of ω_0 (a = 0.5)

Improved Action: Equations of Motion (4)

To improve on this result, we consider the improved discretization:

$$\left(\Delta_2 - a^2\frac{(\Delta_2)^2}{12}\right)x_j = \frac{dV}{dx_j} \tag{40}$$

We translate this directly in matrix form as:

$$\begin{pmatrix} B(\omega_0) & 4/3 & -1/12 & \dots & 0 \\ 4/3 & B(\omega_0) & 4/3 & \dots & 0 \\ -1/12 & 4/3 & B(\omega_0) & \dots & \vdots \\ \vdots & \vdots & \vdots & \ddots & 4/3 \\ 0 & 0 & \dots & 4/3 & B(\omega_0) \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-2} \\ x_{N-1} \end{pmatrix} = \begin{pmatrix} -4/3x_{-1} + 1/12x_{-2} \\ 1/12x_{-1} \\ \vdots \\ 1/12x_N \\ -4/3x_N + 1/12x_{N+1} \end{pmatrix} \tag{41}$$

where $B(\omega_0) = -\left(\frac{5}{2} + (a\omega_0)^2\right)$

Improved Action: Equations of Motion (5)

Following what we did beforehand, we expect:

$$\omega^2 = \omega_0^2 \left(1 + \frac{(a\omega_0)^4}{90}\right) + \mathcal{O}((a\omega_0)^6) \tag{42}$$

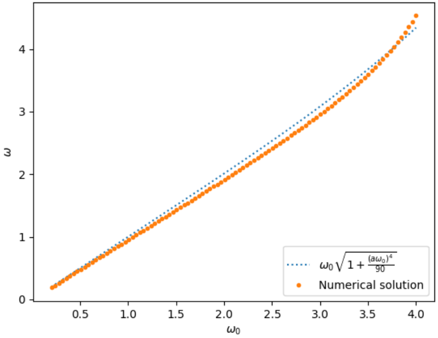


Figure: Solution of the equation of motion for improved action as a function of ω_0 (a = 0.5)

Improved Action: Equations of Motion (6)

The previous solution is way better than before, since it has an error scaling as $\sim a^4$. However, as an effect of the discretization, we have introduced some fictitious solutions, called *numerical ghosts*, which do not depend on ω_0 and simply go as:

$$\omega \simeq \frac{0.005}{a} \tag{43}$$

To eliminate these, we can perform a transformation of the potential:

$$V(x_j) \mapsto \tilde{V}(x_j) = \frac{1}{2}\omega_0^2 x_j^2 \left(1 + \frac{(a\omega_0)^2}{12}\right) \tag{44}$$

and use the transformed action $\tilde{S}$:

$$\tilde{S} = \sum_{j=0}^{N-1} a \left[-\frac{1}{2} x_j \Delta_2 \, x_j + \tilde{V}(x_j)\right] \tag{45}$$

Improved Action: Equations of Motion (7)

We expect the solution to go as:

$$\omega^2 = \omega_0^2 \left(1 - \frac{(a\omega_0)^4}{360}\right) + \mathcal{O}((a\omega_0)^6) \tag{46}$$

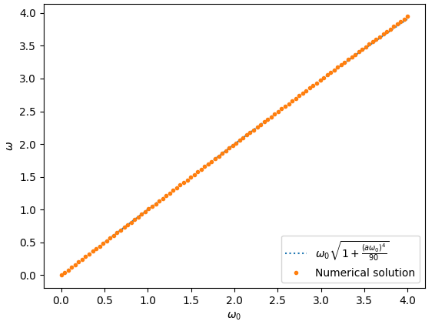


Figure: Solution of the equation of motion for transformed action $\tilde{S}$ as a function of ω_0 (a = 0.5)

Introduction to Lattice QCD (1)

In the field of Lattice QCD, we wish to compute path integrals of *gauge fields* by using similar discretization techniques as in the case of the 1d harmonic oscillator. Let us see the necessary ingredients for that.

- We begin by discretizing both 3d space and time in a *lattice*. The nodes are all separated by the same distance a, called *lattice spacing*.
- Adjacent nodes are connected by *links* which may be *oriented* in space and time.
- The length of each lattice side is usually indicated by L, and it is usually taken to be the same for all the 4 space-time directions.

A simple representation is the following:

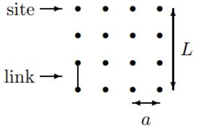


Figure: Pictorial representation of a 2d lattice

Introduction to Lattice QCD (2)

Depending on the lattice spacing a, lattice simulations can be quite computationally demanding. In fact, computational cost is proportional to $1/a^6$, making simulations on fine lattices very hard. Luckily, contrary to what was initially thought, lattice spacings of $a \simeq 0.3 - 0.4\text{fm}$ work well enough, thanks to a series of improvements in the action, comparable to what we have in QM.
First of all, we know QCD must be asymptotically free, which means that we have to suppose that the latttice approximation holds only for the non-perturbative case and we have to impose that for

$$p > \frac{\pi}{a} \tag{47}$$

the theory is perturbative. This suggests that we should be able to make LQCD work for lattice spacings up to 0.5fm.

Introduction to Lattice QCD (3)

Let us build QCD on a lattice. The continuous Yang-Mills action for QCD is given by:

$$S = \int \frac{1}{2} \operatorname{tr}[F_{\mu\nu} F^{\mu\nu}] \mathrm{d}^4 x \tag{48}$$

where $F_{\mu\nu}$ is the field strength tensor of the gluons in matrix form. QCD is a theory which is invariant under local SU(3) transformations. This is the reason behind the choice of the variables in LQCD. Instead of using the field configurations A_μ, we use *link variables*:

$$U_\mu(x) = \mathcal{P} e^{-i \int_x^{x+\hat{\mu}} g_3 A_\nu \mathrm{d}y^\nu} \tag{49}$$

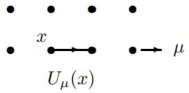
where $\mathcal{P}$ is the path-ordering operator and g_3 the SU(3) coupling. In fact, LQCD is *invariant* under SU(3) transformations of these variables on the lattice:

$$U_\mu(x) \mapsto U'_\mu(x) = \Omega(x) U_\mu(x) \Omega^\dagger(x + a\hat{\mu}) \tag{50}$$

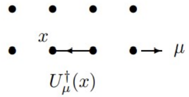
which is the ultimate reason for using link variables instead of field configurations.

Introduction to Lattice QCD (4)

Link variables are oriented and can be represented as follows on the lattice.



whereas the hermitian conjugates $U^\dagger_\mu(x)$ are oriented the opposite way



A *Wilson Loop* is a function of a closed path C on the lattice quantifying the variation of the field configurations on such path:

$$W(C) = \frac{1}{3}\,\mathrm{tr}\left[\mathcal{P}e^{-i\oint_C g_3 A_\nu \mathrm{d}y^\nu}\right] \tag{51}$$

Introduction to Lattice QCD (5)

The simplest non-trivial gauge-invariant operator one can build from link variables is the *plaquette operator* $P_{\mu\nu}$, involving the product of link variables on the smallest square starting at x in the directions μ and ν:

$$P_{\mu\nu}(x) := \frac{1}{3}\mathrm{Re}\,\mathrm{tr}\left[U_\mu(x)U_\nu(x+\mu)U_\nu^\dagger(x+\nu)U_\nu^\dagger(x)\right] \tag{52}$$

We can use it to write the traditional *Wilson action* for gluons on a lattice:

$$S_{Wil} = \beta\sum_x\sum_{\mu>\nu}(1-P_{\mu\nu}(x)) \tag{53}$$

where $\beta = 6/g_3$. In fact, simply expanding $P_{\mu\nu}$ into gauge fields, one finds it equals the Yang-Mills action up to order a^2

$$S_{Wil} = \frac{1}{2}\int \mathrm{tr}[F_{\mu\nu}F^{\mu\nu}]\mathrm{d}^4x + \mathcal{O}(a^2) \tag{54}$$

Computation of Wilson Loops

We can already apply what we know up to now to actually simulate some simple Wilson loops. To do that, we apply what we learned in simple QM to gluonic path integrals, using MC techniques to evaluate the loops. First, we need a specialized Metropolis algorithm to update the link variables. In this case the $U_\mu(x)$ are imaginary exponentials of gauge fields, i.e. SU(3) matrices. In the following slide we outline a specialized algorithm to update the U_μ.

Specialized Metropolis Algorithm (1)

The algorithm is based on the (una tantum) generation of random SU(3) matrices. To do this we proceed as follows

i Start with a random matrix H whose entries are imaginary numbers whose real and imaginary parts are uniformly distributed between -1 and 1.

ii Make this matrix hermitian by transforming :

$$H \mapsto \frac{H + H^\dagger}{2} \tag{55}$$

iii Generate a U(3) matrix from H computing the matrix exponential:

$$M = e^{i\epsilon H} \tag{56}$$

where ϵ is the (small) Metropolis step.

iv Normalize the matrix M to make its determinant equal to 1.

v Save both M and $M^\dagger$ and repeat until you have enough matrices to cover all of SU(3) with their products (100 should suffice)

Specialized Metropolis Algorithm (2)

Once we have a set of random SU(3) matrices M_k we can proceed with the true Metropolis part.

i Start with a generic set of link configurations.

ii Extract a random SU(3) matrix M from the set and transform $U_\mu(x) \mapsto MU_\mu(x)$.

iii Compute the variation of the action ΔS. This is not so computationally demanding, in fact we can write:

$$\Delta S(x, \mu) = \operatorname{Re} \operatorname{tr}[(MU_\mu(x) - U_\mu(x))\Gamma_\mu(x)] \tag{57}$$

where $\Gamma_\mu(x)$ is a sum of products of link variables independent of $U_\mu(x)$, so it can be computed once for each set of successive updates of the link variables $U_\mu(x)$.

iv Update the link variable only if $\Delta S < 0$ or $e^{-\Delta S} > q$ with $q \in Unif(0, 1)$. Otherwise restore the old value of $U_\mu(x)$

v Repeat this process about 10 times per link variable before moving on to the next one to speed up the thermailzation process

Unimproved Wilson Loops (1)

Let us apply this to compute the unimproved Wilson loops $a \times a$ and $a \times 2a$ with the simple Wilson action. One important thing to determine is the Γ_μ factor for each U_μ. Given the form of the plaquette operator, one can easily see that for the Wilson action:

$$\Gamma_\mu(x) = \sum_{\nu\neq\mu} \left[U_\nu(x+\mu)U_\mu^\dagger(x+\nu)U_\nu^\dagger(x) + U_\nu^\dagger(x+\mu-\nu)U_\mu^\dagger(x-\nu)U_\nu(x-\nu)\right]$$

since one has to count all the terms in the action involving $U_\mu(x)$. We thermalize the lattice updating the link variables a certain number of times before starting.

Unimproved Wilson Loops (2)

After that, we start computing the Wilson loops. We have that:

$$W_{a\times a}(x) = \frac{1}{6}\sum_{\mu>\nu}\frac{1}{3}\mathrm{Re}\,\mathrm{tr}\left[U_\mu(x)U_\nu(x+\mu)U_\mu^\dagger(x+\nu)U_\nu^\dagger(x)\right] \qquad (58)$$

and

$$W_{a\times 2a}(x) = \tfrac{1}{6}\textstyle\sum_{\mu<\nu}\tfrac{1}{3}\mathrm{Re}\,\mathrm{tr}\left[U_\mu(x)U_\nu(x+\mu)U_\nu(x+\mu+\nu)U_\mu^\dagger(x+2\nu)U_\nu^\dagger(x+\nu)U_\nu^\dagger(x)\right] \qquad (59)$$

where the factors of 1/6 in front come from the average over spacetime directions. We will also have to average the Wilson loops with respect to the starting point x on the lattice. The code for this whole calculation is quite long and does not fit these slides, it can be found in the repository in ref [3].

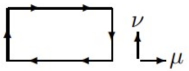
Unimproved Wilson Loops (3)

The results for unimproved Wilson Loops are given in the table. We ran Ncf=10 simulations on a cubic lattice with Nsite = 8 sites per side, a metropolis step $\epsilon = 0.24$, Ncor = 50 and setting $\beta = 5.5$. The results align with the expectations of 0.50 and 0.26 respectively.

Itn.	WL $a \times a$	WL $a \times 2a$
1	0.49508	0.25757
2	0.49840	0.26043
3	0.49513	0.25675
4	0.49331	0.25509
5	0.50335	0.26619
6	0.49375	0.25494
7	0.49506	0.25496
8	0.49833	0.26268
9	0.49156	0.25735
10	0.49289	0.25684
average:	0.49569	0.25828
std. dev:	0.00104	0.00112

Improved Action in LQCD(1)

The previous results, albeit correct, are not what we would expect in QCD. To correct this, one has to *improve the action*. The simplest thing one can do is add to the Wilson action an additional term, usually called the *rectangle operator* $R_{\mu\nu}$ which eliminates the a^2 error in the approximation of the Yang-Mills action. Schematically, we represent it as:

$R_{\mu\nu} = \frac{1}{3}\,\mathrm{Re}\,\mathrm{Tr}$


the new action, then, can be obtained combining this with $P_{\mu\nu}$ in such a way as to cancel $\mathcal{O}(a^2)$ errors.

$$S_c = \text{const} - \beta \sum_x \sum_{\mu>\nu} \left[\frac{5}{3} P_{\mu\nu} - \frac{R_{\mu\nu} + R_{\nu\mu}}{12} \right] = \frac{1}{2} \int \mathrm{tr}\left[F_{\mu\nu} F^{\mu\nu}\right] \mathrm{d}^4 x + \mathcal{O}(a^4) \tag{60}$$

Improved Action in LQCD(2)

However, this is not sufficient. Even if, classically, we improved the action considerably, our system is quantum, and it gets *renormalized*. Since we build our observables from *link variables* rather than directly fields, additional terms in the expansion of $U_\mu(x)$ produce subleading corrections containing powers of $A_\mu(x)$.

$$U_\mu \simeq e^{-iag_3 A_\mu} \simeq 1 - iag_3 A_\mu + \mathcal{O}((ag_3 A)^2) \tag{61}$$

These terms can get quite large, since the extra vertices generated cancel suppressing factors of a, leaving only powers g_3, which in non-perturbative regime is not small. This is known as *tadpole problem*, and the simplest way to solve it is to rescale each link variable by the so-called *mean link* u_0:

$$U_\mu(x) \mapsto \frac{U_\mu(x)}{u_0} \tag{62}$$

where

$$u_0 = \left\langle \frac{1}{3} \mathrm{Re}\,\mathrm{tr}[U_\mu(x)] \right\rangle \tag{63}$$

Improved Action in LQCD(3)

The problem with this is that u_0 is *gauge dependent*. To reduce this problem to a minimum, u_0 is usually defined in *Landau Gauge*, which is specifically the gauge choice that *maximizes* u_0, reducing the tadpole to a minimum. Otherwise, u_0 can simply be estimated as:

$$u_0 \simeq \langle 0|P_{\mu\nu}|0\rangle^{1/4} \tag{64}$$

Because of the very definition of $P_{\mu\nu}$, this is very close to the actual value, and can be used for numerical purposes. Thus, the fully improved action reads:

$$S_{imp} = -\frac{\beta}{u_0^4}\sum_x \sum_{\mu>\nu}\left[\frac{5}{3}P_{\mu\nu} - \frac{R_{\mu\nu}+R_{\nu\mu}}{12u_0^2}\right] \tag{65}$$

Improved Wilson Loops (1)

We can use this new action to rerun the simulations and get better results. The main differences will be the value of β that this time is set so that $\beta/u_0^4 \simeq 4.2$ and the expression of Γ_μ. In fact, in this case:

$$\Gamma_\mu^{(imp)} = \frac{\beta}{u_0^4}\left[\frac{5}{3}\Gamma_\mu - \frac{1}{12u_0^2}\Gamma_\mu^{(R)}\right] \tag{66}$$

where Γ_μ is the same as in the unimproved case while $\Gamma_\mu^{(R)}$ is the modification due to the (symmetrized) rectangle operator. The calculation of the Wilson Loops at this point proceeds as in the unimproved case.

Improved Wilson Loops (2)

The results for Improved Wilson Loops are given in the table. We ran Ncf=10 simulations on a cubic lattice with Nsite = 8 sites per side, a metropolis step $\epsilon = 0.24$, Ncor = 50 setting $\beta/u_0 = 4.2$. The results align with the expectations of 0.54 and 0.28 respectively.

Itn.	WL $a \times a$	WL $a \times 2a$
1	0.54077	0.28478
2	0.54155	0.28538
3	0.54232	0.28599
4	0.54063	0.28491
5	0.53883	0.28154
6	0.54204	0.28329
7	0.53609	0.27870
8	0.53899	0.28002
9	0.53891	0.28071
10	0.53972	0.27967
average:	0.53998	0.28250
std. dev:	0.00056	0.00080

References

- Lepage, G. Peter, Lattice QCD for Novices, 2005, arXiv, arXiv:hep-lat/0506036
- `https://vegas.readthedocs.io/en/latest/`
- `https://github.com/lucab99-it/Theoretical-and-Numerical-Aspects-of-Nuclear-Physics`

Thank you for your attention.