Lattice Gauge Theory and a Simulation of a Pure SU(2) Gauge Theory

Edward Wardell
School of Physics, Edinburgh University
s2072334@ed.ac.uk

***Abstract*—The aim of this report is to give the reader an introductory insight into a simulation of a 2D, SU(2) pure gauge theory. An insight into some background detail of the theories behind lattice QCD. Utilising Yang-Mills theory to apply non-Abelian gauge theories on a lattice, specifically a SU(2) representation of this. In addition to the computational methods that was undertaken in this project used to simulate a gluon field of this specific group of SU(2) matrices. The role of the gauge coupling is investigated. Results reveal a remarkable interplay between beta and the acceptance rate of randomly generated matrices. Finally a discussion of the results.**

I. Introduction

The utilisation of *Yang-Mills theory* allow for the mathematical formulation describing the behaviour of elementary particles using non-Abelian Lie groups.
Starting with quantum chromodynamics, which is a theory of the strong interaction. The mathematical framework underlying this theory (which utilises Yang-Mills theory of non-Abelian symmetries, resulting in simple unitary matrices describing the properties of such fields) allows for interesting physical quantities. Like glueballs and colour confinement. However this theory is extremely difficult to resolve analytically.
The program will be a basic simulation of pure 2D, SU(2) gauge theory, making use of *lattice gauge theory* to described and simplify the fore-mentioned theory. The simulation is a task that is accessible to do on modern PCs.
The motivation to simulate the field, apart from "why wouldn't you want to?", is quite compelling.
The importance of self-interaction of gauge theories, provides insight into understanding fundamental fields, where gauge theories are crucial for the understanding of the underlying physics behind them.
The Euclidean path integral formulation is a fundamental tool for quantising fields on the lattice; a cornerstone of quantum field theory. It offers a powerful framework to describe the behavior of quantum fields, making it particularly valuable for understanding the dynamics of SU(2) gauge theories like the one under investigation in this report.
This approach accounts for quantum effects, interactions, and non-perturbative phenomena, allowing us to derive field equations of these quantum systems. The report will be focusing on the simplest case, quantisation of a scalar field theory.
Physical properties of the lattice that will be simulated can be measured. Ensuring the physical observables are gauge-invariant. Building on from this one must take care to ensure that the physics is still valid post discretisation of the continuous equations.
The observable under main focus in this report is the Wilson loop operator. It allows for the determination of potential between two static colour sources.

II. Background

Dating back to 1954, Chen-Ning Yang and Robert Mills developed a mathematical framework to characterise non-Abelian gauge fields in the hopes to describe the behavior of certain particles and their interactions. This incorporated experimental and theoretical understanding that something like the weak interaction has properties that can't be explained using traditional Abelian symmetries (like U(1) for electromagnetism) notably self interaction, where one must account for additional complexities. Importantly, the theories describing the field must ensure gauge invarience of the systems to be physically acceptable.
To see how Yang-Mills mathematical framework we start with the gauge fields of the theory, A_μ where μ is the space-time coordinate. These matrices capture the local symmetry of the theory.
The Yang-Mills equation for an SU(2) gauge theory is,

$$D^\mu F^a_{\mu\nu} = J_\nu, \;\; where \;\; F^a_{\mu\nu} = \partial_\mu A^a_\nu - \partial_\nu A^a_\mu + g f^{abc} A^b_\mu A^c_\nu \tag{II.1}$$

is the field strength tensor, defining how the gauge fields interact with each other. Noting that $gf^{abc}A^b_\mu A^c_\nu$ is required because of the non-Abelian behaviour of the SU(2) gauge theory.
g is the coupling constant and f^{abc} is the structure factor of the SU(2) gauge theory. D_μ is the contravariant derivative, which encapsulates the invarience of local transformations of the gauge theory. Meaning the left hand side of II.1 represents how the field strength tensor changes under these local gauge transformations. Finally J^ν describes how sources of the gauge field (charges) interact with the gauge field.

As mentioned, it is a requirement that there is invariance under local rotations. To represent such a gauge group, SU(2) (special unitary) matrices are used to describe the gauge transformations that govern the dynamics of the gauge theory in question in this report. For a continuous quantum field theory, these SU(2) matrices are operators at each point in space-time. Allowing for physical observations to remain the

same under gauge transformations. On a lattice, the gauge fields are defined on the links, moreover an element of the compact Lie group is assigned to a link.
Noting that in the 2D simulation of the *lattice gauge theory*, there will be a matrix for each link dimension, so two matrices for a lattice point; horizontal and vertical.

A. *Quantisation of the System*

Tackling the quantisation of a classical field is a tricky endeavor but the report will proceed. The quantised field equation is obtained from the classical scalar field *Klein-Gordon* equation. The corresponding action is given by,

$$S_G[\Phi] = \int dt\ d^3x\ L(\Phi(t,\mathbf{x}), \partial_\mu \Phi(t,\mathbf{x})) \qquad \text{(II.2)}$$

Where, $L(\Phi, \partial_\mu \Phi)$ is the lagrange density. Noting this is in 4D. Converting to 2D isn't an issue and the equations are similar. Noting that when plugging the Lagrange density into the *Euler-Lagrange* the *Klein-Gordon* will pop back out.
The next step of quantisation is is to utilise the canonical formalism. The Hamiltonian of the system is therefore required. This can be obtained from the Lagrangian density and naturally the canonical momentum,

$$\Pi(t,\mathbf{x}) = \frac{\partial}{\partial \dot{\Phi}(t,\mathbf{x})} L(\Phi(t,\mathbf{x}), \partial_\mu \Phi(t,\mathbf{x})) = \dot{\Phi}(t,\mathbf{x}) \qquad \text{(II.3)}$$

[2]
One can utilise the Legendre transformation, a transformation between the Lagrangian and Hamiltonian descriptions of a system. Giving,

$$H = \int d^3x\ \Pi(t,\mathbf{x})\ \dot{\Phi}(t,\mathbf{x}) - \int d^3x L(\Phi(t,\mathbf{x}), \partial_\mu \Phi(t,\mathbf{x})) \qquad \text{(II.4)}$$

[2] [3] Within the continuum limit the Hamiltonian is the first step in quantisation of the system. The Hamiltonian function, the canonical momentum and the scalar field $\Phi(\hat{t},\mathbf{x})$ are transformed into Schroedinger operators, $\hat{H}, \Phi(\mathbf{x})\ \ and\ \ \Pi(\hat{\mathbf{x}})$. It must be mentioned that there is no dependency of time in the Hamiltonian now, this is because of a time operator which is obtained from the Euclidean correlator II.10.

$$\hat{H} = \int d^3x (\frac{1}{2}\hat{\Pi}(\mathbf{x})^2 + \frac{1}{2}(\hat{\Phi}(\mathbf{x}))^2 + \frac{m^2}{2}\hat{\Phi}(\mathbf{x})^2 + V(\hat{\Phi}(\mathbf{x}))) \qquad \text{(II.5)}$$

[2] This means the *Klein-Gordon* equation has been quantised, as one can act the Hamiltonian onto quantum states of the system. From this Hamiltonian the Euclidean action of the field Φ can be calculated. The problem with this approach to quantising the scalar field is that it isn't well suited for simulating a quantum scalar field, as it doesn't deal with the discretisation of the field well. An alternative method which has advantages over this method is the *path integral method.*

B. *Path Integral Method*

1) *Discretising the Lattice:*

The discretization of a quantum field is a requirement for simulation and also the implementation of the Euclidean path integral method. This is due to quantum fields having an infinite number of degrees of freedom. Where one would be required to integrate over all field configurations to obtain physical observations.

It is approached by *lattice gauge theory*; discretising space-time. This ensures Lorentz invarience isn't butchered. For real world applications a 4D Euclidean lattice is put in place of the continuous space-time. However, similar physics applies for a 2D lattice. The lattice will have a lattice constant *a*. In this, the degrees of freedom are the classical field variables Φ. The discretisation is useful because it allows us to work with a finite number of degrees of freedom.

$$\Lambda_2 : \mathbf{x} \Rightarrow a\mathbf{n},\ \ n_i = 0, 1, ..., N-1\ \ for\ \ i = 1, 2 \qquad \text{(II.6)}$$

The continuous equation for the action for the SU(2) gauge field is the Yang-Mills action, which utilises II.1 and is necessary for a non-Abelian gauge field, like for pure SU(2) to be described. It is given as,

$$S_{YM} = \frac{1}{4g^2} \int d^4x F^a_{\mu\nu} F^a_{\mu\nu} \qquad \text{(II.7)}$$

noting the coupling constant $\beta = \frac{1}{g^2}$
Two more important equations that define pure gauge theory on the lattice are as follows,

$$\langle O \rangle = \frac{1}{Z} \int \mathcal{D}[\Phi] \exp^{-S_E[\Phi]} O(\Phi) \qquad \text{(II.8)}$$

[1] This is the vacuum expectation value of an observable in the quantized Euclidean gauge field theory. With respect to simulating a field, $\langle O \rangle$ cannot be analytically evaluated, unless really small lattices are considered. The partition function is given as

$$Z = \int \mathcal{D}[\Phi] \exp^{-S_E[\Phi]} \qquad \text{(II.9)}$$

[1] which represents the probability amplitude for all field configurations of the system. Intuitively this equation makes sense as the action of the field $S_E[\Phi]$ is a functional of the gauge field configurations. In addition to $\int \mathcal{D}[\Phi]$ being short hand for integrating over all field configurations.
After discretising the lattice the next step in the Euclidean path integral method of quantising the lattice is discretising the Euclidean action. Where in the limit of $a \to 0$ the continuum reuslts are recovered. Then this discrete action is then weighted by the boltzmann factor.
Next the Euclidean corrector is utilised

$$\langle O_2(t) O_1(t) \rangle_T = \frac{1}{Z_T} \int \mathcal{D}[\Phi]\ e^{-S_E[\Phi]} O_2[\Phi(., n_t)] O_1[\Phi(., 0)], \qquad \text{(II.10)}$$

The Euclidean correlator II.10 [3] under investigation involves the transformation of operators into functionals. This transformation is achieved by plugging in the field operators with the classical lattice field variables. It allows for the measurement of the properties of the field using classical lattice field variables Φ and ensures they are discrete variables. Finally, with translated functionals and lattice field configurations, the Euclidean correlator functions can be calculated. This involves evaluating these functionals on specific lattice field configurations, weighting them with the Boltzmann factor which would be derived from the lattice action. Then integrating over all possible field configurations. This ensemble averaging is a key to the path integral method.

It works because when the lattice is discretised, it essentially quantizes the classical field theory defined on the lattice. The discrete lattice field variables behave as quantum degrees of freedom, making it possible to perform quantum field theory calculations numerically. This method complements the Hamiltonian approach. However, allowing physical observables on the lattice to be calculated much easier.
For a pure SU(2) gauge theory, which is described by the necessary gauge fields. What it is saying is that the gauge fields that describe the field live on the links, between the lattice points. So in the case of this simulation, there are two SU(2) matrix for one lattice point. One for the horizontal and one for vertical. The SU(2) matrices describe the dynamics of the lattice for a SU(2) gauge field.
When transitioning to lattice gauge theory for the study of non-Abelian gauge fields, such as SU(2), when the Yang-Mills Euclidean action II.7 is discretised on the lattice. The Wilson action is obtained from this. Which uses plaquettes, which atone to simulating a gauge field which has non-Abelian symmetry. This is more inline with simulating actual physical forces, like the weak nuclear force.
The Wilson action, expressed as,

$$S_{Wilson}[U] = \frac{\beta}{N}\sum_{\Box}\Re\, Tr[\mathbb{I} - U_{\Box}(x)] \tag{II.11}$$

[5] This focuses on calculating the action by considering the product of SU(2) link variables $U_{\Box}$ associated with neighboring lattice links. These link variables form plaquettes, small, closed loops of lattice links, so in 2D it would be a square. Given as,

$$U_{\Box}(x) = U_{\mu}(x)U_{\nu}(x+\hat{\mu})U^{\dagger}_{\mu}(x+\hat{\nu})U^{\dagger}_{\nu}(x) \tag{II.12}$$

[5]
When calculating the Wilson action, the plaquette calculation has a trace term of the product of link variables around a loop, this gives a gauge of the local field interactions within the lattice. The Wilson action accurately represents a continuous action onto a discrete lattice, ensuring the preservation of essential non-Abelian properties and gauge invariance. This action is utilised in the simulation of the 2D, pure SU(2) gauge theory. Allowing for the determination of the field configurations that contribute to the path integral the most II.8. In addition to the fundamental use in the *metropolis algorithm* III.5.

The key observable that this project will be working with is the Wilson loop. Given by,

$$W_{\mathcal{L}}[\Phi] = tr[\prod_{(n,\mu)\in\mathcal{L}}\Phi_{\mu}(n)] \tag{II.13}$$

[3] where $\Phi_{\mu}(n)$ are the link variables. Where because we're working with an SU(2) representation of a field, $\mu = (0, 1, 2)$. As physical observables have to be gauge invarient. A gauge invarient object can be made from the trace of a product of link variables along a closed loop 'L'. One can customise the wilson loop size, e.g. a 10x1 of plaquettes. The simulation in question only considers the basic 1x1 plaquette, the values of this are discussed in the results section.

III. Methodology

Once the suitable quantised Euclidean gauge field theory on the lattice is formulated. Monte Carlos sampling is utilised to approximate the integral. This is required because the integrals from the Euclidean path integral method are infinitely dimensional. They are calculated by integrating over all configurations in space and weighted by the Boltzmann factor.
Generally speaking, the expectation of some function $f(x)$ in respect to a probability distribution is

$$\langle f\rangle_{\rho} = \frac{\int_a^b dx\rho(x)f(x)}{\int_a^b dx\rho(x)} \tag{III.1}$$

[3] The importance sampling Monte Carlos integration method allows for the expectation value to be approximated and averaged over N values,

$$\langle f\rangle_{\rho} = \lim_{N\to\infty}\frac{1}{N}\sum_{n=1}^{N} f(x_n) \tag{III.2}$$

[3]
Where for each $x_n(a, b)$ is drawn randomly from the distribution with a normalized probability density.

$$dP(x) = \frac{\rho(x)dx}{\int_a^b dx\rho(x)} \tag{III.3}$$

Applying the SU(2) operators and their expectation values II.8 II.9, it is clear to see that the link variables will follow the sampling of III.3, giving;

$$dP[\Phi] = \frac{e^{-S[\Phi]}\mathcal{D}[\Phi]}{\int \mathcal{D}[\Phi]e^{-S[\Phi]}} \tag{III.4}$$

A. Metropolis algorithm

The Metropolis Algorithm is a basic and effective technique to advance the Markov chain from a configuration to a new configuration.

1) Markov Chain: Markov chains are essentially a stochastic sequence that follows an equilibium distribution where the future state depends on the current state. The update to the new field configuration is called a *Monte Carlos step*. [4]

The Metropolis algorithm functions by iteratively proposing new states based on a proposal distribution III.4 and accepting or rejecting these proposals. In the case of our simulation, $P(\Phi) \propto \exp(-\beta S[\Phi])$ is the probability distribution used. Noting *S[Φ]* isn't the action of the whole lattice. That isn't required and will be mentioned later. Where β is the coupling constant of the field. In SU(2) lattice gauge theory, as one can see it is related to the Wilson action II.11. It determines the coupling strength to the temperature, but also has a hand in the lattice spacing. Physically, a larger value of β corresponds to stronger interactions between the gauge fields, resulting in a more confined phase in the theory. In the lattice formulation, β is often used to control the properties of the gauge fields, varying β allows one to explore different phases of the theory, such as the confinement-deconfinement transition. This is a little unattanable in a pure SU(2) gauge theory but one can hope.
The acceptance or rejection is determined by a specific criterion, which ensures that the Markov chain eventually converges to the desired distribution. The given acceptance probability is given by;

$$T_A(\Phi'|\Phi) = min(1, \frac{T_0(\Phi|\Phi')\exp(-\beta S[\Phi'])}{T_0(\Phi'|\Phi)\exp(-\beta S[\Phi])}) \qquad \text{(III.5)}$$

[3] , where the total transition probability is $T = T_0 T_A$ and T_0 is the proposal probability. $T(\Phi'|\Phi)$ would be the probability to get to the configuration Φ' if starting from Φ. One can prove that 'T' fulfills the detailed balance condition [6]. Due to the symmetry of the min operation, in many cases one is safe to assume a symmetric selection probability obeys,

$$T_0(\Phi|\Phi') = T_0(\Phi'|\Phi)$$

This result means III.5 can be simplified to,

$$T_A(\Phi'|\Phi) = min(1, \exp(-\beta\Delta S)) \qquad \text{(III.6)}$$

Where, $\Delta S = S[\Phi'] - S[\Phi]$.

Within the simulation the following determined whether or not the proposed link would be accepted. When a new link is proposed.
The action is calculated through a local action change, as the links that are being updated will affect the change in the action. As the links are updated iteratively it works out that because all the other links from opposing plaquettes will cancel out [*figure:1*]. So, only a local action change is required for the update of a single link.

The expectation value of the specific observable under investigation II.11 is dependant on the values of the plaquettes II.12. An autocorrelation function was implemented on the observable. The results of this and the Wilson loop are discussed in the next section.

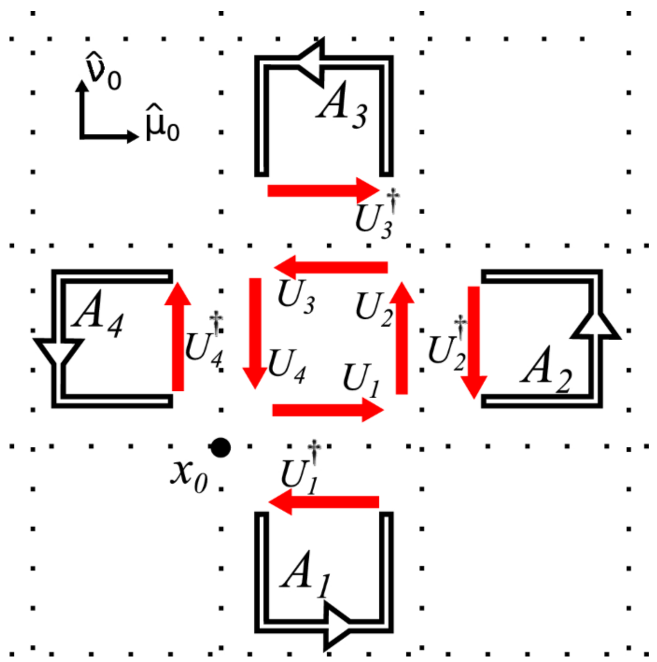

Fig. 1. Local Action Update of a Plaquette [5]. Where A_1, A_2, A_3, A_4 are the plaquettes and U_1 for example is a horizontal link going to a lattice point. The total value of the plaquette is given by II.12.

IV. Results

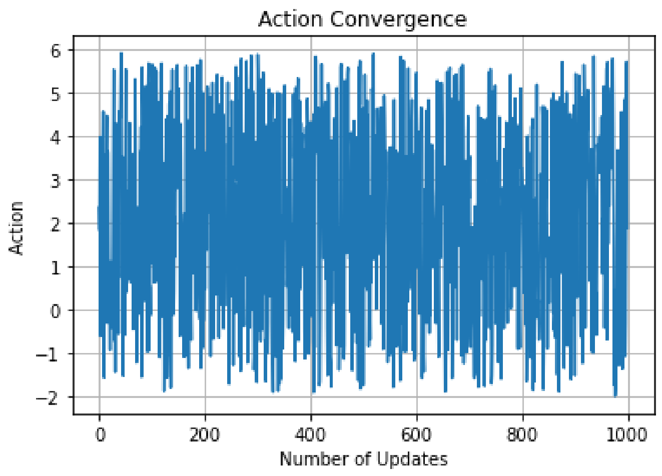

Fig. 2. The Convergence of The Action On The Lattice

In the results of our 2D SU(2) lattice simulation. The action, which characterizes the energy of the field configuration.

As expected, our simulation exhibited fluctuations in the action [*figure: 2*], this is in line with theoretical predictions. These fluctuations are intrinsic to lattice simulations, arising due to the Markov Chain Monte Carlo (MCMC) sampling process, because they work by exploring the lattice which results in the observed flutuations.
It is important to note that while the action exhibited fluctuations, the vacuum expectation values of physical observables

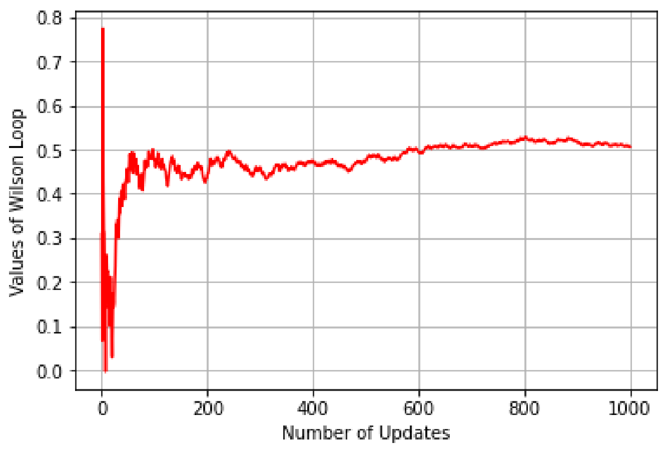

Fig. 3. Expectation Values of The Wilson Loop Observable: Plotted using a simulation of 2D pure SU(2) gauge theory of a scalar field. The coupling constant was 1.2 for this simulation.

that were under investigation, mainly the Wilson loop, demonstrated stability and convergence [*figure: 3*]. This behavior is consistent with the principles of lattice gauge theory, where gauge-invariant observables provide physically meaningful results despite the fluctuations in the action.

The expectation value of the Wilson loop observable acting on the lattice, [*figure*: 3]. Which is approximated using II.13, noting that each point was divided by the numstep of the current interation of the code, adhearing to the Monte Carlos approximation III.2.
One can see the expectation value converges around 0.5, at the specific coupling strength of $\beta = 1.2$. This is a specific coupling constant and maybe simulations were run, this value is where convergence of the wilson action were first observed. This is from positive and negative contributions seeming to balance out, resulting in the observed value. Although this doesn't directly hint towards a phase transition, where changes in the ground state would need to be observed. It does however the direct convergence of the observable does imply potential physical properties of the lattice. More experiments, simulations and further investigating could lead to something significant.

The autocorrelation of the observable. Which is a statistical tool to see how correlated a given observables values are at different time steps in the simulation; the correlation between link variables. This can be used to understand how quickly the system has reach equilibrium or has at all [3] [4]. The autocorrelation was simulated, noting that it is dependent on the *computer time* for a Markov Chain in equilibrium. The autocorrelation was calculated using the plaquettes. However when plotted, the results of the autocorrelation function of the plaquettes are too noisy, so it wasn't applicable to collate valid results.
This report doesn't go into any more detail into the autocorrelation function. But it could be useful for future reports. Because when you plot the relevant observables as a function of the Markov chain's trajectory, you can see how quickly your chain is converging and estimate the effective sample size for statistical analysis.
The simulation could be improved with the application of additional dimensions, 4D. This would mean actual, meaningful results would be produced. As there isn't much physical application for a 2D lattice. Additionally, changing to a SU(3) gauge field would be an added interest, as the fermion field can be simulated. Although, it would require a totally new algorithm if implemented.

V. Conclusion

The code could've been a little more refined, seeing as a vertical and horizontal 2x2 matrix was implemented for each link. A 2x2x2 matrix could've been used instead for each lattice point. Resulting in a more concise code.

VI. Personal Statement

During this project, I've gained invaluable skills and experiences that extend beyond the technical aspects. I learnt how to properly research on an individual basis, as there was very little contact time, this helped me learn from my mistakes well. Especially misunderstanding content and bouncing back to ensure the simulation and report is technically correct. Also how to read very advanced and unseen topics in physics. Additionally, my ability to analyze and solve problems has significantly improved. I've honed my critical thinking skills, helped by the sheer amount of troubleshooting that was required. In addition to applying the theory and simulating it. Furthermore, time management and organization have been essential for completing this project. I felt that I engaged with the content well.
Overall, this project has equipped me with skills I can in turn for future reports and professional endeavors.

VII. Lay Summary

This project was an attempt to simulate, on python, a quantum scalar field onto a 2D lattice. The report aims to discuss the theory behind the quantisation of the scalar field that is applied onto the field. Then the processes of how we prepare that to perform a simulation.
Leading onto the physical methods of how the theories can be computed, using correct distributions and utilising Monte Carlos Markov Chain, specifically the Metropolis algorithm, methods to determine high dimensional integrals.
Finally the application of observables are implemented onto the lattice to read off and determine physical results from the simulation.

Acknowledgments

I would like to thank Prof Tony Kennedy for supervising me through this project. Also to the School of Physics and Astronomy Career Development Summer Scholarship at the University of Edinburgh for allowing this project to be financially possible.

References

[1] Michael Creutz. “Lattice gauge theories and Monte Carlo algorithms”. In: *Nuclear Physics B - Proceedings Supplements* 10.1 (1989), pp. 1–22. ISSN: 0920-5632. DOI: https://doi.org/10.1016/0920-5632(89)90061-3. URL: https://www.sciencedirect.com/science/article/pii/0920563289900613.

[2] De-Chang Dai. “A note on the Hamiltonian of the real scalar field”. In: *High Energy Physics - Theory* (2011). DOI: 10.48550/arXiv.1106.1905. arXiv: arXiv:1106.1905 [hep-th].

[3] C. Gattringer and C. Lang. *Quantum Chromodynamics on the Lattice: An Introductory Presentation*. Lecture Notes in Physics. Springer Berlin Heidelberg, 2009. ISBN: 9783642018497. URL: https://books.google.co.uk/books?id=l2hZKnlYDxoC.

[4] Anosh Joseph. *Markov Chain Monte Carlo Methods in Quantum Field Theories*. 1st ed. Springer Cham, 2020, pp. 30–41. DOI: https://doi.org/10.1007/978-3-030-46044-0.

[5] R. Leme, O. Oliveira, and G. Krein. “Approximate dual representation for Yang–Mills SU(2) gauge theory”. In: *The European Physical Journal C* 78.8 (2018), p. 656. DOI: 10.1140/epjc/s10052-018-6101-9. URL: https://doi.org/10.1140/epjc/s10052-018-6101-9.

[6] Uwe-Jens Wiese. “An introduction to lattice field theory”. In: (Aug. 2009), pp. 66–67.

APPENDIX

```
\begin{verbatim}

\small{
# -*- coding: utf-8 -*-
"""
This Python code implements a Monte Carlo simulation for a 2D scalar field coupled to an SU(2)
gauge field using the Wilson action. The simulation investigates the behavior of the
lattice in relation to the coupling constant. It calculates the expectation value
of the Wilson loop. Which can be used to study the system's phase transitions and gauge
field dynamics. The code utilises the Metropolis algorithm for Markov chain updates and offers
insights into the complex interplay between lattice gauge theory and scalar field dynamics."

Created on Sat Jul  1 11:25:05 2023

@author: Edward Wardell s2072334
"""

import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm


class GluonField:
    def __init__(self, lattice_size):

        # Initialises an empty field with a given size

        self.lattice_size = lattice_size

        # Links will make a 2D array that will contain the SU(2) matrix which will describe
        # The field at each link of the lattice.
        # One matrix will be required for either direction of the lattice
        self.horizontal_links = np.zeros((lattice_size, lattice_size, 2, 2), dtype =
        np.complex128)
        self.vertical_links = np.zeros((lattice_size, lattice_size, 2, 2), dtype =
        np.complex128)

    def set_horizontal_link(self, i, j, su2):

        # Sets the value for the field at a specific site (i, j)
        self.horizontal_links[i, j] = np.conj(su2)

    def set_vertical_link(self, i, j, su2):

        self.vertical_links[i, j] = np.conj(su2)

    def recall_horizontal_link(self, i, j):

        # Gets back the values for an su2 at a specific link
        return self.horizontal_links[i, j]

    def recall_vertical_link(self, i, j):

        return self.vertical_links[i, j]

def random_su2_matrix():

        spread =  0.4

        s1 = np.matrix([[0,1],[1,0]])
        s2 = np.matrix([[0,-1j],[1j,0]])
        s3 = np.matrix([[1,0],[0,-1]])

        a = np.random.uniform(-0.5, 0.5)
        b = np.random.uniform(-0.5, 0.5)
        c = np.random.uniform(-0.5, 0.5)
        d = np.random.uniform(-0.5, 0.5)

        norm = np.sqrt(b**2 + c**2 + d**2)

        x_0 = np.sign(a) * np.sqrt(1-(spread)**2)
        x_1 = (b*spread)/norm
        x_2 = (c*spread)/norm
        x_3 = (d*spread)/norm

        x_0 = np.sign(a) * np.sqrt(1-(spread)**2)


        #su2 = np.array([[a + 1j * b, c + 1j * d], [-c + 1j * d, a - 1j * b]])
        random_su2 = np.identity(2)*x_0 + (s1*x_1 + s2*x_2 + s3*x_3) * 1j

        "return" random_su2

"class" Lattice:

    "def" __init__(self, lattice_size):
        self.lattice_size = lattice_size

        # Represents a lattice where the gluon field is defined,
        stores a specific gluon field config on lattice.
        self.field = GluonField(lattice_size)

    def calculate_plaquette(self, i, j):
        horizontal_link_1 = self.field.recall_horizontal_link(i, j)
        horizontal_link_2 = self.field.recall_horizontal_link(i, (j + 1) % self.lattice_size)
        vertical_link_1 = self.field.recall_vertical_link(i, j)
        vertical_link_2 = self.field.recall_vertical_link((i + 1) % self.lattice_size, j)

        plaquette = horizontal_link_1 @ horizontal_link_2 @  np.conj(vertical_link_2) @
        np.conj(vertical_link_1)

        return plaquette

    def calculate_action(self):
        action = 0.0
        for i in range(self.lattice_size):
            for j in range(self.lattice_size):
                plaquette_1 = self.calculate_plaquette(i, j)
                plaquette_2 = self.calculate_plaquette(i, (j + 1) % self.lattice_size)

        action += 1 - np.real(np.trace(plaquette_1))
        action += 1 - np.real(np.trace(plaquette_2))
        """Find the action of the whole lattice, but going through each column of plaquettes
        in the lattice and then moving one row over and repeating until the whole action of the
        lattice is calculated. This allows for it to be plotted and viewed."""

        return action

    def calculate_action_change_at_site(self, i, j):
        action_change = 0.0

        horizontal_link_1 = self.field.recall_horizontal_link(i, j)
        horizontal_link_2 = self.field.recall_horizontal_link(i, (j + 1) % self.lattice_size)
        vertical_link_1 = self.field.recall_vertical_link(i, j)
        vertical_link_2 = self.field.recall_vertical_link((i + 1) % self.lattice_size, j)

        plaquette_1 = horizontal_link_1 @ horizontal_link_2 @ np.conj(vertical_link_2) @
        np.conj(vertical_link_1)

        horizontal_link_3 = self.field.recall_horizontal_link(i, j)
        horizontal_link_4 = self.field.recall_horizontal_link((i + 1) % self.lattice_size, j)
        vertical_link_3 = self.field.recall_vertical_link(i, j)
```

```python
        vertical_link_4 = self.field.recall_vertical_link(i, (j + 1) % self.lattice_size)

        plaquette_2 = horizontal_link_3 @ np.conj(horizontal_link_4) @ vertical_link_4 @
        np.conj(vertical_link_3)

        """Utilises the local plaquette change to find the change in action, rather than calculating
        the total change of the whole lattice."""

        action_change += 1 - np.real(np.trace(plaquette_1))
        action_change += 1 - np.real(np.trace(plaquette_2))

        return action_change

    def perform_local_update(self, i, j, beta):
        proposed_horizontal_link = random_su2_matrix()
        proposed_vertical_link = random_su2_matrix()
        Using the local action change function to determine the
        delta action"""
        delta_action = (
            self.calculate_action_change_at_site(i, j) +
            self.calculate_action_change_at_site(i, (j - 1) % self.lattice_size) +
            self.calculate_action_change_at_site((i - 1) % self.lattice_size, j) -
            self.calculate_action_change_at_site(i, j) -
            self.calculate_action_change_at_site(i, (j + 1) % self.lattice_size) -
            self.calculate_action_change_at_site((i + 1) % self.lattice_size, j)
        )

        "if" delta_action <= 0.0 or np.random.rand() < np.exp(-beta * delta_action):
            self.field.set_horizontal_link(i, j, proposed_horizontal_link)
            self.field.set_vertical_link(i, j, proposed_vertical_link)
        """Implements the metropolis algorithm and accepts a given configuration accordingly"""


    def randomize(self):
        for i in range(self.lattice_size):
            for j in range (self.lattice_size):

                # Assignes SU(2) matrices to each link in the lattice. One per link.
                horizontal_link = random_su2_matrix()
                vertical_link = random_su2_matrix()

                # The self.field represents the Gluon field which will contain the links of the lattice.
                # The set_link will recall a su2 matrix at the link position (i, j).
                Thus creating a link.
                self.field.set_horizontal_link(i, j, horizontal_link)
                self.field.set_vertical_link(i, j, vertical_link)
                # The above might not work, as it doesn’t allow for complex number elements


    def calculate_wilson_loop(self, x, y, size):
        #start the loop with the identity
        """Calculates the Wilson loop at a given position (x, y) and size.
        The Wilson loop is a closed loop formed by the product of links around a plaquette.
        It provides information about the behavior of quarks and confinement in the gauge theory."""
        loopy_loop = np.eye(2, dtype = np.complex128)
        for i in range(size):
            #Top bit
            horizontal_link = self.field.recall_horizontal_link((x + i) % self.lattice_size, y)

            """
            The loop starts from the (x, y) position and moves to the right (x + i, y).
            The link matrix at (x + i, y) is directly multiplied with the loop value,
            maintaining the correct order."""
            loopy_loop = loopy_loop @ horizontal_link
            #Right one
            vertical_link = self.field.recall_vertical_link(x + size, (y + i) % self.lattice_size)
            loopy_loop = loopy_loop @ vertical_link.conj().T
            """
            The loop continues from the last position (x + size - 1, y) and
            moves downward along the right edge (x + size - 1, y + i). Here, we want to
            multiply the link matrix at (x + size - 1, y + i) with the loop value. However,
            since the link matrices are SU(2) matrices, their conjugate transpose operation
            is required to maintain gauge invariance."""
            horizontal_link = self.field.recall_horizontal_link((x + size - i) % self.lattice_size, y + s
            loopy_loop = loopy_loop @  horizontal_link.conj().T
            #Bottom edge
            vertical_link = self.field.recall_vertical_link(x, (y + size - i) % self.lattice_size)
            loopy_loop = loopy_loop @ vertical_link

        return loopy_loop



# Perfom the specified number of updates
lattice_size = 10
num_updates = 1000
beta = 1.2

with open("action.txt", "w") as file:
    pass

with open("wilson_loop.txt", "w") as file:
    pass

with open("su2_matrices.txt", "w") as file:
    pass

lattice = Lattice(lattice_size)
lattice.randomize()

action_history = []
wilson_loop_history = []
wilson_loop_expectation_values = []

wilson = 0

for _ in tqdm(range(num_updates), desc="Metropolis Updates", ncols=100):

    for i in range(lattice_size):
        for j in range(lattice_size):
            lattice.perform_local_update(i, j, beta)

    action = lattice.calculate_action()
    action_history.append(action)

    wilson_loop = lattice.calculate_wilson_loop(0, 0, 3)
    wilson_loop_history.append(wilson_loop)

    wilson_loop_expectation = np.trace(wilson_loop).real
    wilson += wilson_loop_expectation + 0.5
    wilson_loop_expectation_values.append(wilson / (_ + 1))

final_action = lattice.calculate_action()

print("Final Action:", final_action)

plt.plot(range(num_updates), action_history)
plt.xlabel(’ Number of Updates ’)
plt.ylabel(’ Action ’)
plt.title(’ Action Convergence ’)
plt.grid(True)
plt.show()
plt.plot(range(num_updates), wilson_loop_expectation_values, 'r')
plt.xlabel(' Number of Updates ')
plt.ylabel(' Values of Wilson Loop ')
plt.grid(True)
plt.show()
}
\end{verbatim}
```