



**UNIVERSITÀ
DI TORINO**

UNIVERSITÀ DEGLI STUDI DI TORINO

CORSO DI LAUREA MAGISTRALE IN ASTROFISICA E FISICA TEORICA

**Quantum Field Theory on a Highly
Symmetric Lattice**

TESI DI LAUREA MAGISTRALE

Relatore:

Prof. Panero Marco

Candidato:

Aliberti Marco
Matricola 855766

ANNO ACCADEMICO 2022/2023

Abstract

The regularization on a Euclidean lattice, first proposed by Kenneth G. Wilson in 1974, remains the only approach to study strongly coupled, non-supersymmetric non-Abelian gauge theories (including, in particular, quantum chromodynamics: the fundamental theory of the strong nuclear interaction in the Standard Model of elementary-particle physics) from first principles.

While normally the theory is discretized on a four-dimensional hypercubic grid, this is not the only possible choice, and the fact that the explicit breaking of Lorentz-Poincaré symmetries due to the discretization has an impact on the approach to the continuum limit is a motivation to consider the regularization also on other, more symmetric, lattices.

The goal of this thesis consists in studying Yang-Mills theories based on local $SU(N)$ invariance on the lattice of the roots of the exceptional simple Lie group F_4 , which is a four-dimensional body-centered cubic lattice, and the most symmetric regular lattice that exists in four dimensions.

Contents

1	Yang-Mills Theories on the Lattice	1
1.1	The Yang-Mills Continuum Action	1
1.1.1	Scalar Fields	1
1.1.2	Dirac Spinor Fields	2
1.1.3	Quantum Electrodynamics	2
1.1.4	Non-Abelian Gauge Theories	3
1.1.5	Wick Rotation	6
1.2	Lattice Field Theory	7
1.2.1	Why Lattice Field Theory	7
1.2.2	Lattice	7
1.2.3	Scalar Fields on the Simple Hypercubic Lattice	8
1.2.4	Gauge Fields on the Simple Hypercubic Lattice	9
2	Computer Simulation of Pure Gauge Theories	13
2.1	Introduction	13
2.2	Markov Processes	14
2.3	Monte Carlo Algorithms	16
2.3.1	Metropolis Algorithm	16
2.3.2	Heat Bath Algorithm	18
2.3.3	Overrelaxation Algorithm	20
2.4	Measurements	21
2.4.1	Plaquette	21
2.4.2	Polyakov Loops	21
2.5	Summary of a Simulation	22
2.5.1	Initialization	22
2.5.2	Thermalization	22
2.5.3	Measurements	23
2.6	Statistical Analysis	23
2.6.1	Uncorrelated Data	23
2.6.2	Correlated Data	24
3	Symmetries and Non-Hypercubic Lattices	27
3.1	Spacetime Symmetry Restoration	27
3.1.1	Rotational Invariance of the Static Quark Potential	28
3.2	Other Types of Lattice	29
3.2.1	Body-Centered Tesseract	29
3.2.2	F_4 Coroots Lattice	30

3.3	Simulations on Higher Symmetric Lattices	31
3.3.1	Scalar Fields on F_4 Lattice	31
3.3.2	Gauge Theories on the BCT Lattice	32
4	Simulations Results	35
4.1	Rotational Invariance Restoration	35
4.1.1	Simulations Setup	35
4.1.2	Analysis of Data	36
4.1.3	Final Remarks	38
4.2	BCT Lattice	38
4.2.1	Formulation of the Algorithm	38
5	Conclusions	41
A	Code	I

Yang-Mills Theories on the Lattice

1.1 The Yang-Mills Continuum Action

The aim of this chapter is to review the discretization of the Yang-Mills action on a hypercubic lattice in 4 dimensions. In order to do so, the action is obtained firstly in the continuum, beginning from the simplest gauge theory, Quantum Electrodynamics.

This first section is based on material that can be found in standard textbooks on quantum field theory [1–5].

1.1.1 Scalar Fields

In quantum field theory, a real scalar massive field is described, in a 4-dimensional space-time with metric $\eta_{\mu\nu} = \text{diag}(-1, 1, 1, 1)$, by the following Lorentz-covariant action (in natural units, where $c = \hbar = 1$):

$$S[\phi] = \int d^4x \left(-\frac{1}{2} \partial^\mu \phi \partial_\mu \phi - \frac{1}{2} m^2 \phi^2 + V(\phi) \right) \quad (1.1.1)$$

where $V(\phi)$ is any potential, such as $\frac{g}{4!} \phi^4$. Real scalar fields do not describe any real-world elementary particle, though they are useful to learn basic principles of quantum field theory, as they are the simplest fields that can be written.

1.1.2 Dirac Spinor Fields

Let us now take into consideration a (free) quantum field theory describing a fermion, such as a quark or a lepton. Its action can be written as:

$$S_\psi[\psi(x), \bar{\psi}(x)] = \int d^4x (\bar{\psi} \not{\partial} \psi - m \bar{\psi} \psi) \quad (1.1.2)$$

from which, upon the application of the variational principle, the Dirac equation follows:

$$(i \not{\partial} - m) \psi(x) = 0 \quad (1.1.3)$$

It can now be easily checked by direct computation that this action is invariant under a rigid (namely, global) phase transformation, also called a global $U(1)$ transformation:

$$\begin{aligned} \psi(x) &\rightarrow \psi'(x) = e^{-i\alpha} \psi(x) \\ \bar{\psi}(x) &\rightarrow \bar{\psi}'(x) = \bar{\psi}(x) e^{i\alpha} \end{aligned} \quad (1.1.4)$$

where α is a real number that does not depend on the spacetime coordinate x ; note that, if α were a function of x , then the kinetic term of the action (1.1.2) would not be invariant under such transformation.

1.1.3 Quantum Electrodynamics

As the free field theory itself is non-interacting, it does not provide any real-world prediction, so it is useful to write an interacting action where the spinor field is coupled, to the vector field A_μ , i.e., the photon. One way to implement this interaction is to require local, instead of global, invariance of the action (1.1.2) under the phase transformation (1.1.4), where now $\alpha = \alpha(x)$. In order to do so, the covariant derivative has to be defined as follows:

$$D_\mu \equiv \partial_\mu + ig A_\mu \quad (1.1.5)$$

where g is the coupling constant.¹

The vector field's kinetic term is written in terms of its field-strength, namely:

$$\begin{aligned} F_{\mu\nu} &\equiv -\frac{i}{g} [D_\mu, D_\nu] = \\ &= -\frac{i}{g} (D_\mu (\partial_\nu + ig A_\nu) - D_\nu (\partial_\mu + ig A_\mu)) = \\ &= -\frac{i}{g} (\cancel{\partial_\mu \partial_\nu} + ig \partial_\mu A_\nu - g^2 A_\mu A_\nu - \cancel{\partial_\nu \partial_\mu} - ig \partial_\nu A_\mu + g^2 A_\nu A_\mu) = \\ &= \partial_\mu A_\nu - \partial_\nu A_\mu + ig [A_\mu, A_\nu] = \\ &= \partial_\mu A_\nu - \partial_\nu A_\mu \end{aligned} \quad (1.1.6)$$

where $[A_\mu, A_\nu] = A_\mu A_\nu - A_\nu A_\mu = 0$ in the Abelian theory.

Two different fields A_μ and A'_μ describe the same physics if one can be obtained from

¹Usually, in QED, g is called e , the electron charge, though g will be used in analogy to non-Abelian gauge theories.

another through a gauge transformation:

$$\begin{aligned} A'_\mu(x) &= A_\mu(x) + \frac{1}{g} \partial_\mu \alpha(x) \\ F'_{\mu\nu} &= F_{\mu\nu} + \frac{1}{g} (\partial_\mu \partial_\nu - \partial_\nu \partial_\mu) \alpha(x) = F_{\mu\nu} \end{aligned} \quad (1.1.7)$$

Thus, the free action for the vector field is:

$$S_{EM} = -\frac{1}{4} \int d^4x F_{\mu\nu} F^{\mu\nu} \quad (1.1.8)$$

That is also gauge invariant, i.e. invariant under (1.1.7), as $F_{\mu\nu}$ is gauge invariant in an Abelian theory.

The term that broke the local phase invariance of the action (1.1.2) can now be “absorbed” by A_μ through a gauge transformation (1.1.7), thus making the full action gauge invariant:

$$\begin{aligned} S_{QED} &= \int d^4x \left(i\bar{\psi} \not{D} \psi - m\bar{\psi} \psi - \frac{1}{4} F_{\mu\nu} F^{\mu\nu} \right) = \\ &= \int d^4x \left(i\bar{\psi} \not{\partial} \psi - m\bar{\psi} \psi - g\bar{\psi} \not{A} \psi - \frac{1}{4} F_{\mu\nu} F^{\mu\nu} \right) \\ S_{QED} \rightarrow S'_{QED} &= \int d^4x \left(i\bar{\psi} \not{\partial} \psi + \cancel{\bar{\psi} \not{\partial} \alpha \psi} - m\bar{\psi} \psi - g\bar{\psi} \not{A} \psi - \cancel{\bar{\psi} \not{\partial} \alpha \psi} - \frac{1}{4} F_{\mu\nu} F^{\mu\nu} \right) = \\ &= \int d^4x \left(i\bar{\psi} \not{\partial} \psi - m\bar{\psi} \psi - g\bar{\psi} \not{A} \psi - \frac{1}{4} F_{\mu\nu} F^{\mu\nu} \right) = S_{QED} \end{aligned} \quad (1.1.9)$$

1.1.4 Non-Abelian Gauge Theories

Let us now consider a theory in which the fermion has N internal degrees of freedom, labelled by an index $i = 1, \dots, N$. These degrees of represent the N possible charges of the same particle² and are not to be confused with, for instance, the different possible flavors of the quarks, that describe different particles with different masses.

The free action for this fermion field is:

$$S_\psi[\psi_i(x), \bar{\psi}_i(x)] = \sum_{i=1}^N \int d^4x (i\bar{\psi}_i \not{\partial} \psi_i - m\bar{\psi}_i \psi_i) \quad (1.1.10)$$

From now on, the sum over i (and all other repeated Roman indexes) will be omitted, unless differently specified. This action is invariant under the global transformation:

$$\begin{aligned} \psi_i(x) &\rightarrow \psi'_i(x) = U_{ij} \psi_j(x) \\ \bar{\psi}_i(x) &\rightarrow \bar{\psi}'_i(x) = \bar{\psi}_j(x) U_{ji}^\dagger \end{aligned} \quad (1.1.11)$$

if U is any (constant) $N \times N$ matrix such that $UU^\dagger = U^\dagger U = \mathbb{1} \Leftrightarrow U^\dagger = U^{-1}$, or in other words, if $U \in U(N)$. For this reason, this transformation is also called a global $U(N)$ transformation. The phase transformation (1.1.4) is the particular case where

²For example, if $N = 3$ the 3 possible charges are the color charges of QCD, as will be shown later.

$U = e^{-i\alpha} \in U(1)$, which is the only Abelian (commutative) unitary group.

In fact, the internal symmetry relevant for the strong nuclear interaction (in which the spinor fields describe quarks, with $N = 3$ color charges) is not based on the $U(N)$, but rather on the $SU(N)$ group: the reason is that, by imposing the constraint of the determinant being equal to one, one can construct invariant states by the totally antisymmetric combination of three quarks, which corresponds to a baryon.

In an analogous way to what has been done in Section 1.1.3, the $SU(N)$ invariance can be made local by implementing a proper covariant derivative, similar to (1.1.5). In order to do so, the infinitesimal $SU(N)$ transformation has to be considered:

$$U_{ij}(x) = \delta_{ij} + i\theta^a(x) (T^a)_{ij} + O(\theta^2) \quad (1.1.12)$$

where the indices i and j run from 1 to N (as before) and the index a runs from 1 to $N^2 - 1$ (the dimension of the group $SU(N)$). The matrices T^a are the $N^2 - 1$ generators of $\mathfrak{su}(N)$ (the Lie algebra of $SU(N)$), thus they are $N \times N$ hermitean and traceless, which obey the commutation relations:

$$[T^a, T^b] = if^{abc}T^c \quad (1.1.13)$$

where f^{abc} are called *structure constants* of $\mathfrak{su}(N)$. The normalization of these matrices can be chosen such that they obey the condition:

$$\text{Tr}(T^a T^b) = \frac{1}{2} \delta^{ab} \quad (1.1.14)$$

some examples are:

- $N = 2$, $T^a = \frac{\sigma^a}{2}$, with σ^a the Pauli matrices and $f^{abc} = \varepsilon^{abc}$;
- $N = 3$, $T^a = \frac{\lambda^a}{2}$, with λ^a the Gell-Mann matrices.

where ε^{abc} is the completely antisymmetric Levi-Civita symbol.

The covariant derivative, therefore, is written as:

$$D_\mu \equiv \partial_\mu + ig\mathbf{A}_\mu(x) \quad (1.1.15)$$

where an $N \times N$ identity matrix $\mathbf{1}$ multiplying ∂_μ has to be understood, and $\mathbf{A}_\mu(x)$ is a gauge field of $SU(N)$, i.e., a traceless, hermitean $N \times N$ matrix, or, in other words, $\mathbf{A}_\mu(x) \in \mathfrak{su}(N)$.

The covariant derivative can be written more explicitly acting on the set of spinors ψ_i :

$$(D_\mu)_{ij} \psi_j = \partial_\mu \mathbf{1}_{ij} \psi_j + ig (\mathbf{A}_\mu(x))_{ij} \psi_j$$

In order for the action to be gauge invariant, the field \mathbf{A}_μ must satisfy the gauge transformation property

$$\mathbf{A}_\mu(x) \rightarrow \mathbf{A}'_\mu(x) = U(x) \mathbf{A}_\mu(x) U^\dagger(x) - \frac{i}{g} U(x) \partial_\mu U^\dagger(x) \quad (1.1.16)$$

This expression is a little more complicated than (1.1.7), due to the fact that \mathbf{A}_μ is now a non-commuting matrix. However if the Abelian case $U(1)$ is taken into consideration,

where $U(x) = e^{-i\alpha(x)}$, (1.1.7) follows directly from (1.1.16).

Now, it can be easily checked that the kinetic term of the Lagrangian

$$\mathcal{L}_K = i\bar{\psi}_i \not{D}\psi_i = i\bar{\psi}_i \not{\partial}\psi_i - g\bar{\psi}_i \mathbf{A}\psi_i$$

is gauge invariant (i.e., invariant under (1.1.11) and (1.1.16)) through direct computation:

$$\begin{aligned} \mathcal{L}_K &\rightarrow \mathcal{L}'_K = i\bar{\psi}_i U^\dagger \not{\partial} (U\psi_i) - g\bar{\psi}_i \underbrace{U^\dagger U}_1 \mathbf{A} \underbrace{U^\dagger U}_1 \psi_i + i\bar{\psi}_i \underbrace{U^\dagger U}_1 (\not{\partial} U^\dagger) U\psi_i = \\ &= i\bar{\psi}_i U^\dagger (\not{\partial} U) \psi_i + \underbrace{i\bar{\psi}_i \not{\partial}\psi_i - g\bar{\psi}_i \mathbf{A}\psi_i}_{\mathcal{L}_K} + i\bar{\psi}_i (\not{\partial} U^\dagger) U\psi_i = \\ &= \mathcal{L}_K + i\bar{\psi}_i \gamma^\mu (U^\dagger \partial_\mu U + \partial_\mu U^\dagger U) \psi_i = \\ &= \mathcal{L}_K + i\bar{\psi}_i \gamma^\mu \partial_\mu (U^\dagger U) \psi_i = \\ &= \mathcal{L}_K + i\bar{\psi}_i \gamma^\mu \underbrace{\partial_\mu (1)}_{=0} \psi_i = \mathcal{L}_K \end{aligned}$$

Because of this fact, it is directly implied that the covariant derivative (1.1.15) must transform, under a gauge transformation, in the adjoint representation:

$$D_\mu \rightarrow D'_\mu = U D_\mu U^\dagger \quad (1.1.17)$$

The field-strength for the field \mathbf{A}_μ is obtained, as for the Abelian case, through the commutator of two covariant derivatives. The computation is the same as (1.1.6), but this time the commutator term is non-vanishing:

$$F_{\mu\nu} \equiv -\frac{i}{g}[D_\mu, D_\nu] = \partial_\mu \mathbf{A}_\nu - \partial_\nu \mathbf{A}_\mu + ig[\mathbf{A}_\mu, \mathbf{A}_\nu] \quad (1.1.18)$$

This expression can be simplified a little by considering that \mathbf{A}_μ and $F_{\mu\nu}$ are elements of $\mathfrak{su}(N)$, thus writing them in terms of their components w.r.t. the basis T^a :

$$\mathbf{A}_\mu(x) = A_\mu^a(x) T^a \quad (1.1.19)$$

$$F_{\mu\nu}(x) = F_{\mu\nu}^a(x) T^a \quad (1.1.20)$$

and by considering the relation (1.1.13):

$$\begin{aligned} F_{\mu\nu}^a T^a &= (\partial_\mu A_\nu^a - \partial_\nu A_\mu^a) T^a + ig[A_\mu^b T^b, A_\nu^c T^c] = \\ &= (\partial_\mu A_\nu^a - \partial_\nu A_\mu^a) T^a + ig A_\mu^b A_\nu^c \underbrace{[T^b, T^c]}_{if^{bca} T^a} = \\ &= (\partial_\mu A_\nu^a - \partial_\nu A_\mu^a - gf^{abc} A_\mu^b A_\nu^c) T^a \\ F_{\mu\nu}^a &= \partial_\mu A_\nu^a - \partial_\nu A_\mu^a - gf^{abc} A_\mu^b A_\nu^c \end{aligned} \quad (1.1.21)$$

In order to write a kinetic action for the field \mathbf{A}_μ , a term proportional to $F_{\mu\nu} F^{\mu\nu}$, like in (1.1.8), is not enough: because of (1.1.17) and the definition (1.1.18), it must transform as $F_{\mu\nu} F^{\mu\nu} \rightarrow U F_{\mu\nu} F^{\mu\nu} U^\dagger$, therefore it would not be gauge invariant. In fact a gauge invariant action, called Yang-Mills action, is:

$$S_{YM} = -\frac{1}{2} \int d^4x \operatorname{Tr} (F_{\mu\nu} F^{\mu\nu}) \quad (1.1.22)$$

because of the cyclic property of the trace.³ This action can be written in components, using (1.1.20) and the trace property (1.1.14):

$$S_{YM} = -\frac{1}{2} \int d^4x \operatorname{Tr} (F_{\mu\nu}^a F^{b\mu\nu} T^a T^b) = -\frac{1}{4} \int d^4x F_{\mu\nu}^a F^{a\mu\nu} \quad (1.1.23)$$

Here, there are two remarks that need to be done. The first one is that, if the gauge group is taken to be $U(1)$, the action (1.1.23) reduces to (1.1.8), as $a = 1$ because the group $U(1)$ has only 1 generator. The second one is that, if non-Abelian gauge groups are taken into consideration, this action naturally introduces self-interacting cubic and quartic terms, because the structure constants f^{abc} are non-vanishing. This is, for example, the case for the group $SU(3)$, that is used to describe gluon interaction, i.e., Quantum Chromodynamics (QCD). These self-interactions make the the Yang-Mills action interesting to be studied even alone, without any other fermionic or bosonic interacting field, as it will be shown later.

1.1.5 Wick Rotation

Up to now, actions were written in Minkowskian spacetime, where $\eta_{\mu\nu} = \operatorname{diag}(-1, 1, 1, 1)$. In order to have a positive-defined metric $\eta_{\mu\nu} = \delta_{\mu\nu}$ a Wick rotation can be performed by re-defining the time coordinate to be $\tau = it = ix^0$, as is rigorously proven in [6]. The principle behind Wick rotation is that the time coordinate is promoted to be a complex number and the integration (in the action) from $-\infty$ to ∞ is made to be from $-i\infty$ to $+i\infty$ through a rotation of $\frac{\pi}{2}$ in the complex plane, assuming that correlators and other physical quantities do not present singularities in the first and third quadrant of the complex plane of t . This rotation is not only a matter of convenience, in fact it is needed in perturbative QFT for the computation of (otherwise oscillating) functional integrals and in lattice field theory because a Minkowskian lattice cannot be rigorously defined.

The relation between the Minkowski actions written before and the Euclidean actions is $S^M = iS^E$, because $d^4x^E = i d^4x^M$.

Thus, the Euclidean actions are⁴:

$$S^E[\phi] = \int d^4x \left(\frac{1}{2} \partial^\mu \phi \partial_\mu \phi - \frac{1}{2} m^2 \phi^2 + V(\phi) \right) \quad (1.1.24)$$

$$S_F^E = \int d^4x \bar{\psi} (\gamma^\mu (\partial_\mu + ig \mathbf{A}_\mu) + m) \psi \quad (1.1.25)$$

$$S_{YM}^E = \frac{1}{4} \int d^4x F_{\mu\nu}^a F^{a\mu\nu} \quad (1.1.26)$$

where the superscript E , meaning that the action is written in the Euclidean spacetime, will be omitted from now on.

³Actually, a term proportional to $\det(F_{\mu\nu}^a F^{a\mu\nu})$ would be gauge invariant as well, but it would not be a suitable kinetic term as it would involve terms of higher order than 2 in the components $F_{\mu\nu}^a$.

⁴Note that they are not the same as their Minkowskian counterpart, as some redefinitions of the fields and the γ matrices have been implicitly made, in order to make the respective theories well-defined.

1.2 Lattice Field Theory

1.2.1 Why Lattice Field Theory

After the Wick rotation, quantum field theory can be studied perturbatively through the saddle-point evaluation of the path integral, where the partition function

$$Z = \int \mathcal{D}\varphi e^{iS^M[\varphi]} = \int \mathcal{D}\varphi e^{-S^E[\varphi]}$$

has become non-oscillating. This approach, however, works well only if the coupling constants are *small enough* and does not allow to obtain non-perturbative results, i.e., physical quantities that depend on essential singularities in the coupling constant.

For this reason, a non-perturbative approach has to be taken into consideration and one possibility is Lattice Field Theory: the spacetime is discretized to a lattice, with lattice spacing a , and the fields can assume different values only on particular parts of the lattice. More in detail, a scalar field is defined on lattice sites, a vector field is defined on links between sites and objects with k indices are defined on k -simplexes.

If the limit $a \rightarrow 0$ is taken, the theory must reproduce results obtained in the continuum, like the ones obtained in perturbation theory in the regime where it can be applied; note that the presence of a discrete spacetime provides a natural cutoff for the momenta, allowing one to take ultraviolet divergences under control, although they need to be taken into consideration when approaching the continuum limit.

In this section, based on standard quantum field theory textbooks cited before and lattice field theory textbooks [7–9], the regularization of the previous quantum field theories on a Simple Hypercubic (SH) lattice is presented. A more detailed description of the geometric properties of the figures mentioned below can be found in [10].

1.2.2 Lattice

A lattice Λ in \mathbb{R}^D is defined as the set of all possible linear combinations with integer coefficients of the vectors $\{v_i\}$, which form a basis of \mathbb{R}^D . Formally:

$$\Lambda = \left\{ \sum_{i=1}^D c_i v_i \mid c_i \in \mathbb{Z} \right\} \quad (1.2.1)$$

1.2.2.1 Simple Hypercubic Lattice

If the basis $\{v_i\}$ is taken to be orthonormal, the lattice is called *Simple Hypercubic*⁵.

In Figure 1.2.1 is represented a portion of a Simple Hypercubic lattice in $D = 3$.

For the SH lattice in $D = 4$, the fundamental region (the smallest D -dimensional polyhedron) is a 4-dimensional hypercube, also called a tesseract. This means that a SH lattice can be seen as a tassellation of the four-dimensional Euclidean space with tesseracts as elementary cells. Each point of the lattice has 8 nearest neighbours that are identified by

⁵The word *simple* is used to make a distinction with the Body-Centered Hypercubic lattice presented in Section 3.2.

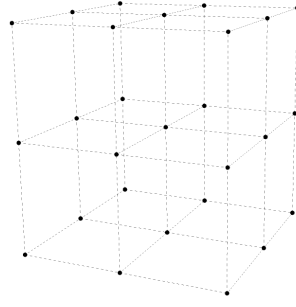


Figure 1.2.1: A cubic lattice.

the vectors obtained through all possible permutations of position and sign of $(\pm 1, 0, 0, 0)$. The plaquette (the simplest bidimensional figure) is a square. This will be important when implementing gauge theories on the lattice.

1.2.3 Scalar Fields on the Simple Hypercubic Lattice

Let us consider a simple hypercubic lattice Λ_{SH} extending over $L = L_1 = L_2 = L_3$ lattice spacings in the spatial directions and over $T = L_0$ lattice spacings in the temporal direction. Let us also consider a scalar field $\phi(x)$, in order to introduce some useful tools that will be needed later. As anticipated in the beginning of this section, a scalar field is defined on the sites of the lattice, therefore $x \in \Lambda_{SH}$, and let us assume, for simplicity, that the field satisfies periodic boundary conditions in all the four directions, therefore $\phi(x + a\hat{\mu}L_\mu) = \phi(x)$. The Fourier transform of the field can be written as

$$\tilde{\phi}(p) = \sum_x a^4 e^{-ip \cdot x} \phi(x) \quad (1.2.2)$$

and the allowed momenta are given by

$$p_\mu = \frac{2\pi}{aL_\mu} n_\mu, \quad n_\mu = 0, \dots, L_\mu \quad (1.2.3)$$

This ensures that the momenta can take only the assigned values of the Brillouin zone (1.2.3) and therefore cannot go to infinity, providing a natural cutoff given by the lattice spacing a .

The discretization also implies that derivatives of the fields cannot be computed, therefore the lattice forward derivative is used:⁶

$$\partial_\mu \phi(x) \rightarrow \nabla_\mu \phi(x) = \frac{\phi(x + a\hat{\mu}) - \phi(x)}{a} \quad (1.2.4)$$

Assuming a self-interaction potential $V(\phi) = \frac{\lambda}{4!} \phi^4$, the action (1.1.1) can be written in terms of the lattice in the following way:

$$S = a^4 \sum_x \left(\frac{1}{2a^2} [\phi(x + a\hat{\mu}) - \phi(x)]^2 + \frac{1}{2} m^2 \phi^2(x) + \frac{\lambda}{4!} \phi^4(x) \right) \quad (1.2.5)$$

A similar approach will be used in the following section to obtain the action for Yang-Mills theories.

⁶Note that, in the continuum limit, the two derivatives coincide.

1.2.4 Gauge Fields on the Simple Hypercubic Lattice

Let us now take into consideration a Yang-Mills theory, as in Section 1.1.4, with gauge group $SU(N)$. As anticipated before, gauge fields, being vector fields, are defined on the links between sites of the lattice.

One could naively think that putting gauge vectors $A_\mu(x)$ on the links is enough, but this would explicitly break gauge invariance. For this reason, Wilson's idea was to put the gauge group (and not algebra) elements, namely $U_\mu(x) = e^{iagA_\mu(x)}$, on the links.

The field $U_\mu(x)$ lives on the link connecting the site x with the site $x + a\hat{\mu}$, therefore the link is oriented and link variables in negative directions can also be defined:

$$U_{-\mu}(x) \equiv U_\mu^\dagger(x - a\hat{\mu}).$$

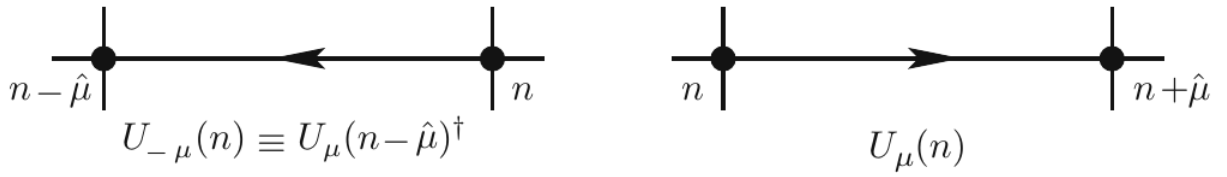


Figure 1.2.2: Schematic visualization of link variables.

Under a gauge transformation, the fields U_μ transform according to the following relation:

$$U_\mu(x) \rightarrow \Omega(x)U_\mu(x)\Omega^\dagger(x + a\hat{\mu}) \quad (1.2.6)$$

where $\Omega(x)$ is any $SU(N)$ matrix at the point x on the lattice. As a consequence of this relation, the trace of any product of links forming a closed path is a gauge-invariant quantity, thanks to the cyclic property of the trace.

With this in mind, Wilson's idea was to choose the simplest closed path possible: the plaquette, that in the simple hypercubic lattice is a square. The product of link variables along a square plaquette is defined in the following way (the lattice spacing a is set = 1 for brevity of notation):

$$\begin{aligned} U_{\mu\nu}(x) &\equiv U_\mu(x)U_\nu(x + \hat{\mu})U_{-\mu}(x + \hat{\mu} + \hat{\nu})U_{-\nu}(x + \hat{\nu}) = \\ &= U_\mu(x)U_\nu(x + \hat{\mu})U_\mu^\dagger(x + \hat{\mu} + \hat{\nu})U_\nu^\dagger(x) \end{aligned} \quad (1.2.7)$$

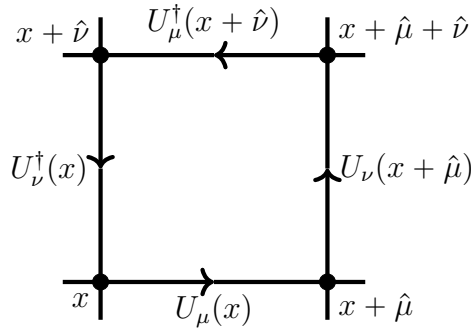
Thus a gauge-invariant action, called Wilson action, can be written as follows:

$$S_W[U] = \frac{\beta}{2N} \sum_{x \in \Lambda} \sum_{\mu < \nu} \text{Re Tr}[\mathbb{1} - U_{\mu\nu}(x)] \quad (1.2.8)$$

where β is a parameter that is going to be set in the following passages, N the number of color charges (the same N in $SU(N)$), the real part is needed to enforce invariance under charge conjugation, and $U_{\mu\nu}$ is the plaquette of (1.2.7).

In the following part, the naive continuum limit is taken, showing that the Wilson action reduces to the Yang-Mills action of (1.1.26) when $a \rightarrow 0$. Let us start by defining an auxiliary field $B_\mu(x)$:

$$B_\mu(x) = agA_\mu^a(x)T^a \quad (1.2.9)$$

Figure 1.2.3: Link variables building a plaquette $U_{\mu\nu}(x)$.

thus the gauge group element becomes $U_\mu(x) = e^{iB_\mu(x)}$.
The expansion for small a of the plaquette $U_{\mu\nu}$ is:

$$\begin{aligned} U_{\mu\nu} &= U_\mu(x) U_\nu(x + \hat{\mu}) U_\mu^\dagger(x + \hat{\nu}) U_\nu^\dagger(x) = \\ &= e^{iB_\mu(x)} e^{iB_\nu(x + \hat{\mu})} e^{-iB_\mu(x + \hat{\nu})} e^{-iB_\nu(x)} = \\ &\quad \text{applying (1.2.4), } B_\mu(x + \hat{\nu}) = a \nabla_\nu B_\mu(x) \\ &= e^{iB_\mu(x)} e^{iB_\nu(x + \hat{\mu}) + ia \nabla_\mu B_\nu(x)} e^{-iB_\mu(x) - ia \nabla_\nu B_\mu(x)} e^{-iB_\nu(x)} \simeq \end{aligned}$$

From now on, the dependence on x will be omitted.

$$\begin{aligned} &\text{Using the Baker-Campbell-Hausdorff formula, } e^X e^Y = e^{X+Y+\frac{1}{2}[X,Y]}, \text{ up to } O(a^2): \\ &\simeq \exp \left\{ i\cancel{B_\mu} + i\cancel{B_\nu} + ia \nabla_\mu B_\nu - i\cancel{B_\mu} - ia \nabla_\nu B_\mu - i\cancel{B_\nu} + \right. \\ &\quad \left. + \frac{1}{2} (-[B_\mu, B_\nu] + [B_\mu, B_\nu] + [B_\nu, B_\mu] - [B_\mu, B_\nu]) \right\} = \\ &= \exp \{ ia (\nabla_\mu B_\nu - \nabla_\nu B_\mu) - [B_\mu, B_\nu] \} \end{aligned}$$

Now, substituting back (1.2.9), the discretized electromagnetic tensor can be found:

$$\begin{aligned} U_{\mu\nu} &\simeq \exp \{ ia^2 g (\nabla_\mu \mathbf{A}_\nu - \nabla_\nu \mathbf{A}_\mu + ig [\mathbf{A}_\mu, \mathbf{A}_\nu]) \} = \exp \{ ia^2 g F_{\mu\nu} \} \simeq \\ &\simeq \mathbb{1} + ia^2 g F_{\mu\nu} - \frac{1}{2} a^4 g^2 F_{\mu\nu} F^{\mu\nu} + O(a^6) \end{aligned} \quad (1.2.10)$$

Finally, the small- a approximation for the Wilson action (1.2.8) can be obtained:

$$\begin{aligned} S_W &\simeq \frac{\beta}{2N} \sum_{x \in \Lambda} \sum_{\mu < \nu} \text{Re Tr} \left(\cancel{\mathbb{1}} - \cancel{\mathbb{1}} - ia^2 g F_{\mu\nu}^i T^i + \frac{1}{2} a^4 g^2 F_{\mu\nu}^i F^{j\mu\nu} T^i T^j \right) = \\ &= \underbrace{-i \frac{a^2 \beta}{2N} \sum_{x \in \Lambda} \sum_{\mu < \nu} \text{Re}(F_{\mu\nu}^i \text{Tr}(T^i))}_{=0 \text{ for (1.1.12)}} + \frac{g^2 a^4 \beta}{4N} \sum_{x \in \Lambda} \sum_{\mu < \nu} \underbrace{\text{Re}(F_{\mu\nu}^i F^{j\mu\nu} \text{Tr}(T^i T^j))}_{=\frac{1}{2} F_{\mu\nu}^i F^{i\mu\nu} \text{ for (1.1.14)}} = \\ &= \frac{g^2 \beta}{8N} a^4 \sum_{x, \mu, \nu} F_{\mu\nu}^i F^{i\mu\nu} \xrightarrow{a \rightarrow 0} \frac{g^2 \beta}{8N} \int d^4 x F_{\mu\nu}^i F^{i\mu\nu} \end{aligned} \quad (1.2.11)$$

where the Yang-Mills action (1.1.26) can be recognized if $\beta = \frac{2N}{g^2}$.

1.2.4.1 Wilson Loops

Now that the lattice action has been defined, one can define lattice observables, too. As mentioned below equation (1.2.6), the trace of the product of link variables along any closed path is a gauge-invariant quantity. This means that any quantity of the sort is a good candidate for an observable and, in fact, given any closed path γ on the lattice, the Wilson loop is defined as follows:

$$W[\gamma] \equiv \text{Tr} \left[\prod_{(x,\mu) \in \gamma} U_\mu(x) \right] \quad (1.2.12)$$

It is worthwhile to mention that this definition does not depend on the type of lattice and of boundary conditions chosen for such lattice. If the path γ does not include any temporal link, the loop is called *spacelike Wilson loop*, otherwise it is called *timelike Wilson loop*. If all the links in γ lie on the same plane, the Wilson loop is *planar*, otherwise it is *nonplanar*.

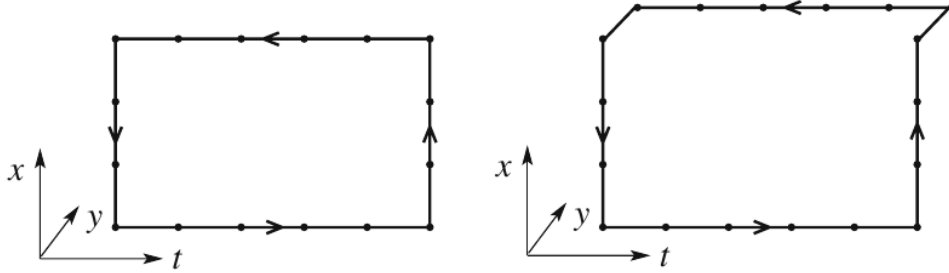


Figure 1.2.4: Example of a planar (left) and nonplanar (right) Wilson loop.

Wilson loops are important for a variety of reasons, that will not be elaborated further as this is not the focus of this project, however it is worth mentioning that they can be used as operators to represent purely gluonic bound states, the so-called *glueballs*: the mass spectrum of these states can be obtained through the exponential decay of Wilson loops' correlation functions.

1.2.4.2 Polyakov Loops

Another important observable can be defined in lattices where periodic boundary conditions in the time direction are assumed. Thanks to periodicity, a whole new class of closed paths arises: all paths that wind around the time direction. A Polyakov loop is thus defined as a line at a fixed spatial position $\vec{x} = (x^1, x^2, x^3)$ that runs along the whole time direction:

$$P(\vec{x}) \equiv \text{Tr} \left[\prod_{t=0}^{T-1} U_0(t, \vec{x}) \right] \quad (1.2.13)$$

The Polyakov loop has a lot of interesting properties: its physical interpretation is the world-line of a single static quark, therefore its expectation value is an order parameter of the deconfinement transition, as well as of a new global symmetry, called *center symmetry*, arising from the periodicity of the time direction.

This means that, if the system is confined (namely, isolated quarks and gluons do not exist as asymptotic states of the theory), its expectation value is 0, while if the system is deconfined, $\langle P(\vec{x}) \rangle \neq 0$.

Another property, that will be used in the simulations, is that the expectation value of the product of two Polyakov loops at distance $r = a|\vec{x} - \vec{y}|$ is a function of the static quark-antiquark potential $V(r)$, as indicated below:

$$\langle P(\vec{x})P^\dagger(\vec{y}) \rangle \propto e^{-TaV(r)} (1 + O(e^{-Ta\Delta E})) \quad (1.2.14)$$

where ΔE is the difference between $V(r)$ and the first excited energy level of the quark-antiquark pair.

The value of the potential $V(r)$ will be used, in this project, to study the recovery of rotational invariance when approaching the continuum limit.

In the following chapter we will discuss various methods to perform computer simulations of lattice field theory and to obtain numerical estimates for the expectation values of quantities such as the plaquette, Wilson and Polyakov loops.

Computer Simulation of Pure Gauge Theories

2.1 Introduction

Monte Carlo simulations are a powerful tool to obtain numerical estimates for the expectation values of observables in lattice field theory. Let us focus on purely gluonic lattice field theories. Given an observable O , its vacuum expectation value is given by the following integral

$$\langle O \rangle = \frac{1}{Z} \int \mathcal{D}[U] e^{-S[U]} O[U] \quad \text{with} \quad Z = \int \mathcal{D}[U] e^{-S[U]} O[U] \quad (2.1.1)$$

where $\mathcal{D}[U]$ is to be intended as the product of the Haar measure for every link variable defined on the lattice.

This expression, however, cannot be evaluated analytically except for very small lattices, therefore (2.1.1) is approximated by an average of the observable evaluated on a sample of

\mathcal{N} gauge field configurations $\{U_n\}^1$, distributed according a probability density $\propto e^{-S[U_n]}$. The expectation value is then obtained by computing the following sum for a sufficient number of configurations generated by the proper Monte Carlo algorithm(s):

$$\langle O \rangle \simeq \frac{1}{\mathcal{N}} \sum_{\{U_n\}} O[U_n] \quad (2.1.2)$$

Because of the probability density $\propto e^{-S[U_n]}$, only configurations $\{U_n\}$ close to the minimum of the Euclidean action give a non-negligible contribution to the integrals, while the contributions from configurations with much larger Euclidean action are exponentially suppressed. For this reason, generating totally random gauge fields on the lattice links is not an efficient way to evaluate (2.1.2), as most of the $\{U_n\}$ will have a very small Boltzmann factor (the $e^{-S[U_n]}$) and expression (2.1.2) will give incorrect results unless a huge number (orders of magnitude higher than what is reasonably possible) of different configurations are included in the sample.

In order to avoid the generation of a huge number of configurations that would contribute little-to-nothing to the observables' values, the integrals are estimated with a Monte Carlo method instead, namely, one chooses the sample of configurations “betting” on the ones that give the largest contribution to the integral. In practice, this can be done by generating a sequence of configurations $\{U_n\}$ through a Markov chain process, built such that its stationary distribution minimizes the action $S[U]$.

Every process of this type must begin from a starting configuration, usually chosen by the user, and one has to skip the configurations produce during a finite transient, before the system reaches *thermalization*. The two mostly used starting configurations are: a *cold start*, if the simulation begins with the fields in an ordered way (for example, all the gauge links set to the identity), or a *hot start*, if the simulation begins with random gauge fields on every link. Of course, the Markov chain must have the same stationary distribution irrespective of the choice of the starting configuration.

In this chapter some simple Monte Carlo algorithms used to thermalize the starting configuration are presented, then the methods used for evaluating some observables of interest, such as plaquettes and Polyakov loops, are explained.

2.2 Markov Processes

As anticipated before, a Markov chain is needed to evolve an arbitrary starting configuration U_0 up to a region of the configuration space with a relatively large Boltzmann factor e^{-S} , therefore with high probability. In Figure 2.2.1 a scheme of a Markov chain is shown.

A Markov process is characterized by a probability of transitioning from a configuration α to another configuration β depending only on α and β^2 and not on the history of the process, namely the previous transitions that already occurred. In formulae, this is expressed as

$$P(U_n = \beta | U_{n-1} = \alpha) = T(\beta | \alpha) \quad (2.2.1)$$

¹Here the subscript n distinguishes the different configurations.

²Note that this β has no relation with the previously defined parameter of the Wilson action.

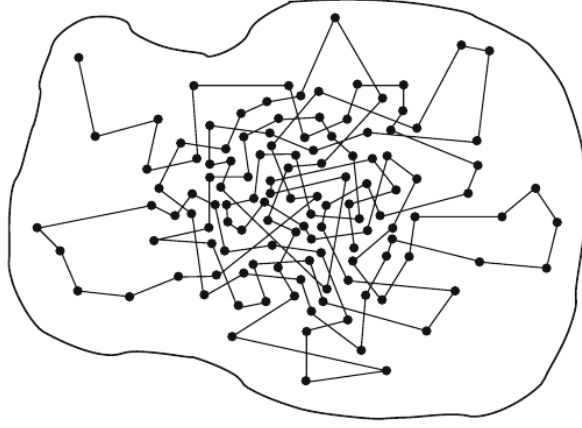


Figure 2.2.1: Schematic representation of a motion through the space of configurations through a Markov process.

That is to say that the transition matrix T does not depend on the index n , representing the computer time.

Being a transition probability, the matrix T must obey the following equations

$$0 \leq T(\beta|\alpha) \leq 1 \quad \forall \alpha, \beta \quad (2.2.2)$$

$$\sum_{\beta} T(\beta|\alpha) = 1 \quad \forall \alpha \quad (2.2.3)$$

where (2.2.2) is a consequence of $T(\beta|\alpha)$ representing a probability, and (2.2.3) means that the probability of transitioning to any configuration must be 1 (of course, the case when $\alpha = \beta$ is included as well).

In order for the stochastic process not to have any sink or source of probability, the following balance equation must be satisfied:

$$\sum_{\alpha} T(\beta|\alpha)P(\alpha) = \sum_{\alpha} T(\alpha|\beta)P(\beta) \quad (2.2.4)$$

This equation states that the probability of transitioning to the configuration β , written in the l.h.s. as the sum of the transition probability from the configuration α weighted by the probability $P(\alpha)$ that the system is actually in that configuration, must be equal to the probability of transitioning out of the configuration β , given by the probability of finding the system in the configuration β times the transition probability $T(\alpha|\beta)$ over all the final configurations, in the right-hand side. Thanks to (2.2.3), the r.h.s. is easily proven to be equal to $P(\beta)$.

A sufficient (albeit not necessary) condition to satisfy the balance equation (2.2.4) is obtained by requiring that it holds true term-by-term, thus obtaining the detailed balance condition:

$$T(\beta|\alpha)P(\alpha) = T(\alpha|\beta)P(\beta) \quad (2.2.5)$$

Although it is not a necessary condition, most algorithms, including the ones discussed in the following section, satisfy it.

2.3 Monte Carlo Algorithms

Monte Carlo algorithms are a class of algorithms that, singularly or combined together, allow to advance the Markov chain, while satisfying the balance condition (2.2.4).

Each algorithm, if applied once, allows the transition from a configuration U_{n-1} to a configuration U_n (which could possibly be equal to U_{n-1}). The repeated application of the algorithm allows to advance through the Markov chain.

2.3.1 Metropolis Algorithm

The first algorithm presented is the Metropolis algorithm. It is not very efficient and it is not often used in simulations unless for pedagogical purposes, however it contains the fundamental steps that are present in some of the more advanced algorithms and it is quite easy to understand. For this reasons it is usually viewed as the “*ancestor*” of all Monte Carlo algorithms and it is present in every textbook on the subject.

This algorithm consists in two steps that implement in one of the most simple ways the detailed balance condition (2.2.5):

- 1) A candidate configuration β is chosen, according to some *a priori* selection probability $T_0(\beta|\alpha)$, where $\alpha = U_{n-1}$.
- 2) The candidate configuration β is accepted as the new configuration U_n with the acceptance probability

$$T_A(\beta|\alpha) = \min \left(1, \frac{T_0(\alpha|\beta) \exp(-S[\beta])}{T_0(\beta|\alpha) \exp(-S[\alpha])} \right) \quad (2.3.1)$$

If it is not accepted, the unchanged configuration is considered again ($U_n = \alpha$) and a new candidate configuration is generated.

These two steps are repeated a sufficient number of times, until a sufficient number of configurations are generated.

Note that $P(\alpha) = \frac{e^{-S[\alpha]}}{Z} \propto e^{-S[\alpha]}$ has been used.

The total transition probability T is obtained through the product $T = T_0 T_A$, as the two steps are independent from each other, and it is straightforward to see that it satisfies the detailed balance condition (2.2.5):

$$\begin{aligned} T(\beta|\alpha)P(\alpha) &= \frac{1}{Z} T_0(\beta|\alpha) T_A(\beta|\alpha) \exp(-S[\alpha]) = \\ &= \frac{1}{Z} T_0(\beta|\alpha) \min \left(1, \frac{T_0(\alpha|\beta) \exp(-S[\beta])}{T_0(\beta|\alpha) \exp(-S[\alpha])} \right) \exp(-S[\alpha]) = \\ &= \frac{1}{Z} \min (T_0(\beta|\alpha) \exp(-S[\alpha]), T_0(\alpha|\beta) \exp(-S[\beta])) = \\ &= \frac{1}{Z} T(\alpha|\beta) \exp(-S[\beta]) = \\ &= T(\alpha|\beta) P(\beta) \end{aligned}$$

□

In many cases a symmetric selection probability is used $T_0(\alpha|\beta) = T_0(\beta|\alpha)$, thus (2.3.1) simplifies to:

$$T_A(\beta|\alpha) = \min(1, e^{-\Delta S}) \quad \text{with} \quad \Delta S = S[\beta] - S[\alpha] \quad (2.3.2)$$

That means that if the new configuration lowers the action, the change is accepted with probability 1 (as $e^{-\Delta S} > 1$), otherwise it is accepted with a certain probability that decays exponentially as a function of the difference in the action of the two configurations. This ensures that the algorithm *moves across* the configuration space towards the minima of the action, while allowing statistical fluctuations.

If the change in the action is local (it involves a single link variable), ΔS can be computed using only the field values in the local neighbourhood. This will be the case for $SU(N)$ gauge theories.

2.3.1.1 Application to $SU(N)$ Gauge Theories

For a $SU(N)$ gauge theory, the algorithm is implemented in the following way. During each iteration, a single link is changed, then the acceptance probability is computed, a (pseudo-)random number between 0 and 1 is extracted (from a uniform distribution) and, if it is less than the acceptance probability the change is accepted, otherwise it is rejected. The algorithm is then iterated a certain number of times to produce the configurations on which the various observables are evaluated.

The candidate link $U'_\mu(x)$ for the first step is generated in the vicinity of the old value $U_\mu(x)$, in order not to have a too great ΔS that would lead to too low acceptance rates. This can be done by exploiting the property that the product of any two elements of $SU(N)$ is still an element of $SU(N)$, therefore extracting a matrix $X \in SU(N)$ near the identity allows to write the candidate link as:

$$U'_\mu(x) = XU_\mu(x) \quad (2.3.3)$$

The matrix X is chosen such that it has the same probability as X^{-1} , this way the selection probability T_0 is symmetric and the computation of the acceptance probability T_A becomes easier.

For this reason, only the variation of the action ΔS must be computed, where of course the action is the Wilson action (1.2.8). In particular, only the plaquettes containing the candidate link must be evaluated: the change in the action is local, so all the other plaquettes will have the same value both before and after the change of the link. Hence $\Delta S = S[U'_\mu(x)]_{loc} - S[U_\mu(x)]_{loc}$.

In a SH lattice, each link is shared between 6 plaquettes, in 4 spacetime dimensions. For each plaquette the change of its value is given by the change of the link, while the product of the other three gauge links, that is called *staple* and will be indicated as P_i , remains unchanged.

Therefore, the local contribution to the action can be computed as:

$$S[U_\mu(x)]_{loc} = \frac{\beta}{2N} \sum_{i=1}^6 \text{Re Tr} [\mathbf{1} - U_\mu(x)P_i] = \frac{\beta}{2N} \text{Re Tr} \left[6\mathbf{1} - U_\mu(x) \sum_{i=1}^6 P_i \right]$$

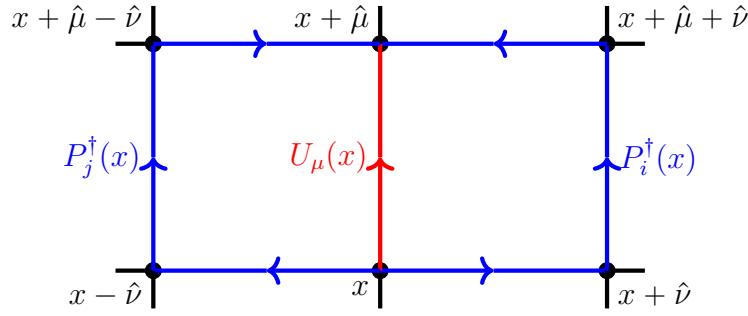


Figure 2.3.2: Example of a link (red) with two of its staples (blue).

where the sum over all the staples is:

$$A = \sum_{i=1}^6 P_i = \sum_{\nu \neq \mu} (U_\nu(x + \hat{\mu}) U_\mu^\dagger(x + \hat{\nu}) U_\nu^\dagger(x) + U_\nu^\dagger(x + \hat{\mu}) U_\mu^\dagger(x - \hat{\nu}) U_\nu(x - \hat{\nu})) \quad (2.3.4)$$

The change of the action can now be computed as

$$\Delta S = S[U'_\mu(x)]_{loc} - S[U_\mu(x)]_{loc} = \frac{\beta}{2N} \text{Re Tr} [(U_\mu(x) - U'_\mu(x)) A] \quad (2.3.5)$$

where A is not affected by the change of $U_\mu(x)$.

2.3.2 Heat Bath Algorithm

The heat bath algorithm can be considered as an improved version of the Metropolis algorithm, which combines the two steps into a single one and chooses the new candidate according to the probability distribution obtained by the computing the surrounding staples:

$$dP(U) = dU \exp\left(\frac{\beta}{2N} \text{Re Tr}[UA]\right) \quad (2.3.6)$$

where dU denotes the Haar integration measure of the gauge group and A is computed according to (2.3.4). In principle, this update method has the advantage that, unlike the Metropolis algorithm, the link variable always changes.

The implementation details depend on the gauge group, for this reason, the heat bath method for the gauge group $SU(2)$ will be now explained³.

Since the sum of any two $SU(2)$ elements is proportional to another $SU(2)$ element, the sum of all staples A (2.3.4) can be written in the form

$$A = V \sqrt{\det(A)} \quad \text{with} \quad V \in SU(2) \quad (2.3.7)$$

where it can be proven that $\det(A) \geq 0$. If $\det(A) = 0$, a random link variable is generated, otherwise the matrix V is well-defined. Plugging into (2.3.6) and using the invariance of

³As matrices of $SU(N)$ can be “built” combining $SU(2)$ matrices, the general case is just a little more complicated, but it follows the same principles.

the Haar measure under translations in the group ($dU = d(UV) = dX$), the distribution probability of the matrix $X = UV$ is

$$dP(X) = dX \exp\left(\frac{\beta}{2N} \sqrt{\det(A)} \operatorname{Re} \operatorname{Tr}[X]\right) \quad (2.3.8)$$

If a matrix X , distributed according to (2.3.8), is generated, then the new candidate link, distributed according to (2.3.6), is obtained as $U'_\mu(x) = XV^\dagger = \frac{1}{\sqrt{\det(A)}} XA^\dagger$.

Any $U \in SU(2)$ matrix can be written in the following representation, using 4 real numbers:

$$U = x_0 \mathbf{1} + i\mathbf{x} \cdot \boldsymbol{\sigma} \quad \text{with} \quad \det(U) = |x|^2 = \sum_{i=0}^3 x_i^2 = 1 \quad (2.3.9)$$

where $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \sigma_3)$ is a vector built using the Pauli matrices and $x = (x_0, \mathbf{x})$ can be seen as a normalized 4-component vector. Using this representation, the Haar measure in (2.3.8) can be written as:

$$\begin{aligned} dX &= \frac{1}{\pi^2} d^4x \delta(x_0^2 + \mathbf{x}^2 - 1) = \\ &= \frac{1}{\pi^2} d^4x \frac{1}{2\sqrt{1-x_0^2}} \left(\delta(|\mathbf{x}| - \sqrt{1-x_0^2}) + \delta(|\mathbf{x}| + \sqrt{1-x_0^2}) \right) \end{aligned} \quad (2.3.10)$$

where a well known property of the Dirac δ distribution has been used.

The volume element can be rewritten in terms of the components of the vector x :

$$d^4x = dx_0 d|\mathbf{x}| |\mathbf{x}|^2 \underbrace{d(\cos \theta) d\varphi}_{d^2\Omega} \quad (2.3.11)$$

Plugging back into (2.3.10) and integrating out the $|\mathbf{x}|$ using the δ distributions (actually, only the first one contributes, as $x_i^2 \leq 1 \forall i$), the Haar measure takes the form:

$$dX = \frac{1}{\pi^2} dx_0 d^2\Omega \frac{1-x_0^2}{2\sqrt{1-x_0^2}} = \frac{1}{2\pi^2} dx_0 d^2\Omega \sqrt{1-x_0^2} \quad (2.3.12)$$

Then, in terms of the variables, the probability distribution becomes:

$$dP(X) = \frac{1}{2\pi^2} dx_0 d(\cos \theta) d\varphi \sqrt{1-x_0^2} \exp\left(\frac{\beta}{2} \sqrt{\det(A)} x_0\right) \quad (2.3.13)$$

with $x_0 \in [-1, 1]$, $\cos \theta \in [-1, 1]$, $\varphi \in [0, 2\pi)$, where $\operatorname{Tr}(X) = 2x_0$ has been used.

Thus, the problem of generating a matrix X distributed according to (2.3.8) has been reduced to the determination of three random variables x_0 , θ , φ , whose distributions factorize, so they can be determined independently from each other.

The random variable x_0 , being distributed according to $\sqrt{1-x_0^2} \exp\left(\frac{\beta}{2} \sqrt{\det(A)} x_0\right)$, is determined through the auxiliary variable $\lambda \in [0, 1]$ such that $x_0 = 1 - 2\lambda^2$, therefore

$$dx_0 \sqrt{1-x_0^2} \exp\left(\frac{\beta}{2} \sqrt{\det(A)} x_0\right) \propto d\lambda \lambda^2 \sqrt{1-\lambda^2} \exp\left(-\beta \sqrt{\det(A)} \lambda^2\right) \quad (2.3.14)$$

The variable λ is generated with the polynomially modified Gaussian distribution density

$$p_1(\lambda) = \lambda^2 e^{-\beta \sqrt{\det(A)} \lambda^2} \quad (2.3.15)$$

and accepted with an accept/reject step using the square root function

$$p_2(\lambda) = \sqrt{1 - \lambda^2} \quad (2.3.16)$$

There are several algorithms that can perform this computation in an efficient way [11, 12].

After this, the length $|x| = \sqrt{1 - x_0^2}$ is computed, in order to determine the remaining variables.

The variables $\cos \theta$ and φ are uniformly distributed, therefore a possible way of proceeding is by generating three random uniformly distributed numbers r_1, r_2 and r_3 in the interval $[-1, 1]$ and accepting them only if $r_1^2 + r_2^2 + r_3^2 \leq 1$. Then, the 3-vector (r_1, r_2, r_3) is normalized to length $|x|$ computed before, obtaining $\mathbf{x} = (x_1, x_2, x_3)$.

After these steps, the vector $x = (x_0, \mathbf{x})$ can be used to generate the matrix X according to (2.3.8), using the representation (2.3.9), thus the new link variable can be obtained.

This algorithm rapidly leads the Markov process to a minimum of the action, however the risk of “getting stuck” in a local minimum is present. For this reason, it is usually combined with other algorithms, like the overrelaxation algorithm.

2.3.3 Overrelaxation Algorithm

The overrelaxation algorithm changes the link variables as much as possible, exploiting the property that new configurations are always accepted if they do not change the action, as can be seen in (2.3.2).

The case with gauge group $U(1)$ is the simplest one: the group element can be written as $U = e^{i\varphi}$, the sum of staples becomes $A = \rho e^{i\alpha}$ and the local action can be written as:

$$S[U]_{loc} = \frac{\beta}{2} \text{Re}(UA) = \frac{\beta}{2} \rho \text{Re}(e^{i\varphi} e^{i\alpha}) = \frac{\beta}{2} \rho \cos(\varphi + \alpha) \quad (2.3.17)$$

The reflection $\varphi + \alpha \rightarrow -\varphi - \alpha$ or the change $\varphi \rightarrow 2\pi - \varphi - 2\alpha$ leaves the local action invariant, thus a change of this type is always accepted. For a non-Abelian group, this change is performed through the ansatz:

$$U \rightarrow U' = V^\dagger U^\dagger V^\dagger \quad (2.3.18)$$

where V is a gauge group element chosen such that the local action is invariant. The selection probability for a transformation of this kind is symmetric, as can be easily proven by inverting (2.3.18), obtaining $U = V^\dagger U'^\dagger V^\dagger$.

For the gauge group $SU(2)$, the matrix V is chosen proportional to the sum of staples: $V = \frac{A}{\sqrt{\det(A)}}$, like in (2.3.7). Because of this,

$$\text{Tr}(U'A) = \text{Tr}\left(V^\dagger U'^\dagger V^\dagger \sqrt{\det(A)} V\right) = \sqrt{\det(A)} \text{Tr}(V^\dagger U'^\dagger) = \text{Tr}(A^\dagger U^\dagger) = \text{Tr}(UA) \quad (2.3.19)$$

therefore the choice for U' leaves the action invariant. In case $\det(A) = 0$, any random link variable is accepted.

As the overrelaxation algorithm leaves the action invariant, it moves the Markov chain in the subspace of constant Euclidean action (or, from a statistical-mechanics point of view, it corresponds to the *micro-canonical* ensemble), thus it is not ergodic: it does not visit every possible point in the configuration space if given enough time. For this reason, it has to be used in combination with other updating algorithms, such as some steps of the Metropolis or heat bath algorithms.

2.4 Measurements

In this section, the techniques used to compute the value of some observables are explained. After the proper Monte Carlo algorithms have been iterated a sufficient number of times, the configurations are *thermalized* and the new configurations generated from that point on can be used to estimate the expectation values of the different observables. The observables presented below are the plaquette and the Polyakov loop, because they are the ones that will be evaluated in the present study.

2.4.1 Plaquette

The mean value of the plaquette, indicated as $\langle \square \rangle$, is obtained by evaluating expression (1.2.7) for all possible plaquettes in the lattice, summing all the results and taking the trace of the result, divided by the total number of plaquettes present in the lattice. In formulae:

$$\langle \square \rangle = \frac{1}{12Nn_s} \sum_{x \in \Lambda} \sum_{\mu=0}^2 \sum_{\nu=\mu+1}^3 \text{Re Tr } U_{\mu\nu}(x) \quad (2.4.1)$$

where N is the number of color charges (the same N in $SU(N)$), $6 = 4 \cdot 3/2$ is the number of different planes in which the plaquette can lie, and $n_s = TL_1L_2L_3$ is the total number of sites of the lattice.

The expectation value of the plaquette is important because it is directly linked to the Wilson action, thus it gives an intuition of the thermalization of the lattice.

2.4.2 Polyakov Loops

Polyakov loops are another important observable that will be evaluated in the simulations. In particular, the correlator of two Polyakov loops will be considered, in order to obtain the static quark potential, as explained in (1.2.14).

In order to achieve better statistics, the discrete translational invariance of the lattice is exploited: because of this invariance, the quantity $\langle P(0)P^\dagger(x) \rangle$, with $x \in \Lambda$ is the same as $\langle P(y)P^\dagger(x+y) \rangle \forall y \in \Lambda$, therefore the expectation value is obtained as

$$\langle P(0)P^\dagger(x) \rangle = \frac{1}{n_s} \sum_{y \in \Lambda} \langle P(y)P^\dagger(x+y) \rangle \quad (2.4.2)$$

In this way, from the same configuration, more samples of the same quantity can be obtained (even though they may be not completely independent from each other), improving the statistical precision of the numerical estimates that one would like to compute.

2.5 Summary of a Simulation

Now that the Monte Carlo algorithms and the way observables are evaluated on the lattice have been described, we present a brief explanation of how a full simulation works in the implementation of our code.

2.5.1 Initialization

The first step is the initialization: the program reads from a file the size of the lattice (the number of sites in each direction: t, x, y, z), the parameter β that appears in the Wilson action (1.2.8), the thermalization time t_{TH} , the reunitarization period t_U (which is the number of Monte Carlo steps after which the gauge field variables are projected back to the $SU(N)$ group, to avoid the accumulation of rounding errors), the number of measurements to be made n_M , the number of updates to be done between each measurement n_{BM} and the start type, either fully ordered (cold start) or totally random (hot start). The number of color charges and other parameters, like the type of measurements that have to be made are set in the source code of the program. Then, according to these parameters, the variables are initialized to the proper value and the simulation can begin.

2.5.2 Thermalization

The thermalization step is needed, as anticipated before, to advance the Markov process up to equilibrium configurations. This is done by updating the configuration with the Monte Carlo algorithms until the number of updates equals the thermalization time t_{TH} indicated in the input file. Every update step consists of a certain number of heat bath steps followed by another number of overrelaxation iterations. Both these numbers are set in the source code and for these simulations 1 iteration of heat bath and 3 of overrelaxation have been used.

The thermalization time is determined by looking, in other simulations, at the behaviour of some observables in different types of start: for instance, when some observable from a cold start takes a mean value compatible to the one obtained from a hot start, the system is assumed as thermalized.

The thermalization time t_{TH} is then set as 2 or 3 times the number of steps needed to achieve thermalization, in order to keep a certain margin of confidence.

Additionally, as we mentioned above, every t_U iterations, all the lattice link variables are *reunitarized*, that means that each one matrix is checked, and eventually corrected, that it still is unitary (i.e., $U^\dagger = U^{-1}$), because unitarity can be lost due to computer roundig errors. After each update step, the mean value of the plaquette is written to a file.

2.5.3 Measurements

After the lattice has achieved thermalization, observables can finally be measured. Each observable is computed once, according to the methods illustrated in the previous section, and written to a file, then the system is updated a number of times equal to n_{BM} , with a reunitarization process every t_U updates, in the same way as the thermalization step. This process is iterated until the observables are measured n_M times. After that, the simulation is concluded.

2.6 Statistical Analysis

In this final section, an overview on the analysis of the data collected from the simulations is given. After a run of the program is finished, one usually has a few thousand values of each observable, obtained in different configurations of the fields. The average of these values is used as an estimate for the expectation value of the observable, with its own statistical uncertainty.

Let us assume that the values (o_1, \dots, o_N) of some observable O have been computed from a Markov sequence of Monte Carlo generated configurations in equilibrium. The expectation value and variance of these variables are the same:

$$\langle o_i \rangle = \langle O \rangle \quad \sigma_{o_i}^2 = \langle (o_i - \langle o_i \rangle)^2 \rangle = \sigma_O^2 \quad (2.6.1)$$

whose estimators can be:

$$\hat{O} = \frac{1}{N} \sum_{i=1}^N o_i \quad (2.6.2)$$

$$\hat{\sigma}_O^2 = \frac{1}{N-1} \sum_{i=1}^N (o_i - \hat{O})^2 \quad (2.6.3)$$

This holds for both correlated and uncorrelated datasets.

2.6.1 Uncorrelated Data

If the values are uncorrelated, the product $\langle o_i o_j \rangle$ factorizes

$$\langle o_i o_j \rangle = \langle o_i \rangle \langle o_j \rangle = \langle O \rangle^2 \quad \forall i \neq j \quad (2.6.4)$$

and the variance $\hat{\sigma}_O^2$ allows one to determine the statistical uncertainty of \hat{O} . In fact, the sample mean value (2.6.2) is itself a random variable, since its value may change from one

set of configurations to another. Its variance is:

$$\begin{aligned}
\sigma_{\hat{O}}^2 &= \left\langle \left(\hat{O} - \langle O \rangle \right)^2 \right\rangle = \left\langle \left(\frac{1}{\mathcal{N}} \sum_{i=1}^{\mathcal{N}} (o_i - \langle O \rangle) \right)^2 \right\rangle = \\
&= \frac{1}{\mathcal{N}^2} \left\langle \sum_{i,j=1}^{\mathcal{N}} (o_i - \langle O \rangle) (o_j - \langle O \rangle) \right\rangle = \\
&= \frac{1}{\mathcal{N}^2} \left\langle \sum_{i=1}^{\mathcal{N}} o_i^2 \right\rangle + \frac{1}{\mathcal{N}^2} \left\langle \sum_{i \neq j} o_i o_j \right\rangle - \frac{2}{\mathcal{N}} \langle O \rangle \sum_{i=1}^{\mathcal{N}} o_i + \frac{1}{\mathcal{N}^2} \left\langle \sum_{i,j=1}^{\mathcal{N}} \langle O \rangle^2 \right\rangle = \\
&= \frac{1}{\mathcal{N}} \langle O^2 \rangle + \frac{1}{\mathcal{N}^2} \sum_{i \neq j} \langle o_i o_j \rangle - 2 \langle O \rangle^2 + \langle O \rangle^2 = \\
\sigma_{\hat{O}}^2 &= \frac{1}{\mathcal{N}} \langle O^2 \rangle - \langle O \rangle^2 + \frac{1}{\mathcal{N}^2} \sum_{i \neq j} \langle o_i o_j \rangle
\end{aligned} \tag{2.6.5}$$

For uncorrelated o_i , the last two terms cancel each other because of (2.6.4) and

$$\sigma_{\hat{O}}^2 = \frac{1}{\mathcal{N}} \sigma_O^2 \tag{2.6.6}$$

Therefore, for \mathcal{N} uncorrelated measurements, the standard deviation σ , i.e., the statistical error, is $\sigma_{\hat{O}}$ and the final result is

$$\hat{O} \pm \sigma \quad \text{with} \quad \sigma = \frac{\hat{\sigma}_O}{\sqrt{\mathcal{N}}} \tag{2.6.7}$$

As can be easily seen, the statistical error decreases $\propto 1/\sqrt{\mathcal{N}}$ with the number of uncorrelated configurations \mathcal{N} .

2.6.2 Correlated Data

However, not every data sample is statistically uncorrelated and, since in this case it is taken from a computer-time series in the Monte Carlo simulation, the correlation between the data depends on the Monte Carlo algorithms and how many times they have been iterated between each measurement. If the data sample is correlated, the *autocorrelation function*

$$C_O(t) = C_O(o_i, o_{i+t}) = \langle (o_i - \langle o_i \rangle) (o_{i+t} - \langle o_{i+t} \rangle) \rangle = \langle o_i o_{i+t} \rangle - \langle o_i \rangle \langle o_{i+t} \rangle \tag{2.6.8}$$

is nonvanishing. Note that $C_O(0) = \sigma_O^2$.

Typically, the normalized correlation function $\Gamma_O(t)$ exhibits an exponential behaviour for large t :

$$\Gamma_O(t) \equiv \frac{C_O(t)}{C_O(0)} \underset{t \rightarrow +\infty}{\approx} e^{-t/\tau_O} \tag{2.6.9}$$

where τ_O is called the exponential autocorrelation time for the observable O . The exponential autocorrelation time τ is the supremum of the values τ_O for all possible observables:

$$\tau = \sup_O \tau_O \tag{2.6.10}$$

Hence, for the correlated case, the second line of (2.6.5) can be continued as follows, using (2.6.8):

$$\begin{aligned}
 \sigma_{\hat{O}}^2 &= \frac{1}{\mathcal{N}^2} \left\langle \sum_{i,j=1}^{\mathcal{N}} (o_i - \langle O \rangle) (o_j - \langle O \rangle) \right\rangle = \\
 &= \frac{1}{\mathcal{N}^2} \sum_{i,j=1}^{\mathcal{N}} C_O(|i-j|) = \frac{1}{\mathcal{N}^2} \sum_{t=-(\mathcal{N}-1)}^{\mathcal{N}-1} \sum_{k=1}^{\mathcal{N}-|t|} C_O(|t|) = \\
 &= \sum_{t=-\mathcal{N}}^{\mathcal{N}} \frac{\mathcal{N}-|t|}{\mathcal{N}^2} C_O(|t|) = \frac{C_O(0)}{\mathcal{N}} \sum_{t=-\mathcal{N}}^{\mathcal{N}} \Gamma_O(|t|) \underbrace{\left(1 - \frac{|t|}{\mathcal{N}}\right)}_{\rightarrow 1} \simeq \\
 &\simeq \frac{\sigma_{\hat{O}}^2}{N} 2 \left(\frac{1}{2} + \sum_{t=1}^{\mathcal{N}} \Gamma_O(t) \right) \equiv \frac{\sigma_{\hat{O}}^2}{\mathcal{N}} 2\tau_{O,int}
 \end{aligned} \tag{2.6.11}$$

where the *integrated autocorrelation time* has been introduced as:

$$\tau_{O,int} \equiv \frac{1}{2} + \sum_{t=1}^{\mathcal{N}} \Gamma_O(t) \underset{\mathcal{N} \rightarrow \infty}{\simeq} \int_0^{\infty} e^{-t/\tau} dt = \tau \tag{2.6.12}$$

Usually $\tau_{O,int}$ is estimated taking at least $1000\tau_{O,int}$ data values starting with small size lattices and proceeding to larger sizes while evaluating $\Gamma(t)$.

The variance computed in this way is larger than the one computed using (2.6.6), therefore for the correlated case the value of an observable O is given by

$$\hat{O} \pm \sigma_O \quad \text{with} \quad \sigma_O = \sqrt{\frac{2}{\mathcal{N}} \tau_{O,int} \hat{\sigma}_O^2} \tag{2.6.13}$$

Symmetries and Non-Hypercubic Lattices

3.1 Spacetime Symmetry Restoration

The restoration of spacetime symmetries in the continuum limit is an important aspect of lattice field theory. Minkowskian spacetime is invariant under the action of the Poincaré group, that includes translations, rotations in the 3-dimensional space, and boosts. When performing the Wick rotation, boosts become rotations in the planes formed by each spatial axis and the Euclidean time. The Euclidean spacetime is therefore invariant, in the continuum, under translations:

$$x^\mu \rightarrow x^\mu + \varepsilon^\mu \quad (3.1.1)$$

and under rotations in 4 dimensions:

$$x^\mu \rightarrow R^\mu_\nu x^\nu \quad \text{with} \quad R \in O(4) \quad (3.1.2)$$

These invariances do not hold anymore once the space is discretized: a generic lattice is, in fact, invariant only under translations by integer multiples of the elementary lattice

vectors:

$$x \rightarrow x + a\mu \quad \text{with} \quad x \in \Lambda \quad (3.1.3)$$

and under certain rotations:

$$x \rightarrow \Gamma x \quad \text{with} \quad x \in \Lambda, \Gamma \in G_\Lambda \quad (3.1.4)$$

where $G_\Lambda \subset O(4)$ is a discrete subset of the full rotational group $O(4)$ that depends on the lattice Λ . For example, $\Lambda = \Lambda_{SH}$ is a SH lattice, then $G_{\Lambda_{SH}}$ is the 384-elements group with all possible reflections on every coordinate plane and rotations of multiples of 90° around each one of the 4 axis.

The fact that the symmetries in the lattice theory and in the continuum theory are different is not a problem in itself, as long as when approaching the continuum limit the symmetries of the continuum theory are recovered. This is the case for the translational symmetry: if $a \rightarrow 0$, (3.1.3) becomes (3.1.1). However, the rotational symmetry is not restored (at least, not naively) when the continuum limit is taken: ((3.1.4) does not become (3.1.2) when $a \rightarrow 0$), as the Simple Hypercubic lattice is invariant under rotations of multiples of 90° independently from the value of the lattice spacing.

However, the restoration of continuum symmetries is more subtle than this intuition: in the lattice theory, it typically happens that the terms breaking the continuum rotational symmetry down to its discrete subgroup that is manifest on the lattice are actually suppressed by powers of the lattice spacing in the $a \rightarrow 0$ limit. This suppression, however, may fail, if the coefficients of the terms associated with higher powers of a are, instead, divergent in the continuum limit. For this reason, studies on the restoration of rotational symmetry have been made, through the computation of quantities that are expected to exhibit rotational invariance in the continuum theory.

3.1.1 Rotational Invariance of the Static Quark Potential

An example of these studies is provided by a 1982 article by Lang and Rebbi [13], where the static quark potential $V(r)$, obtained from the correlator of two Polyakov loops (see (1.2.14)), was studied at different lattice spacings in the $SU(2)$ lattice gauge theories. The potential

$$V(r) = -T \ln \langle P^\dagger(r) P(0) \rangle \quad (3.1.5)$$

was obtained from simulations for different values of $r = (r_x, r_y, r_z)$ on lattices extending for n_s lattice spacings in each of the space directions and n_t lattice sites in the time direction. In this work, the full gauge group $SU(2)$ was actually approximated by its discrete icosahedral subgroup \tilde{Y} (due to the computing power limitations of that time). The various (about 1000) potentials obtained for each site r were then fitted according to the expected form for a linearly confining potential

$$V(r) = c_0 + c_1/r + c_2 r \quad (3.1.6)$$

A representation of the equipotential surfaces of expression (3.1.6) is reproduced in Figure 3.1.1.

In Figure 3.1.1a a lattice with $n_s = 8$, $n_t = 4$ and $\beta = \frac{4}{g^2} = 2$ was used, while Figure 3.1.1b was obtained from data obtained on a lattice with $n_s = 16$, $n_t = 6$ and $\beta = 2.25$.

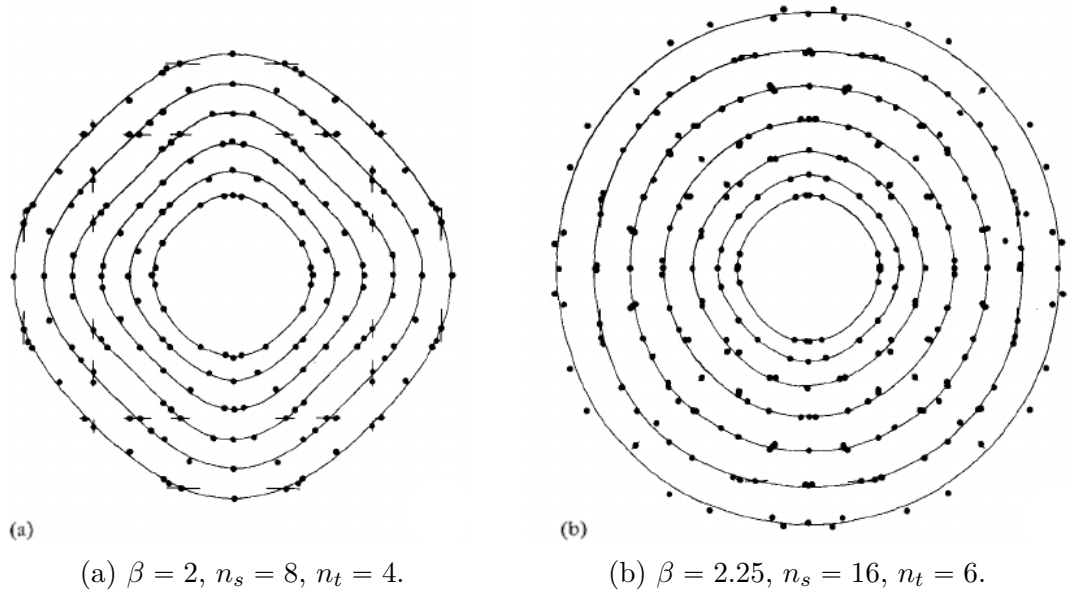


Figure 3.1.1: Representation of equipotential surfaces for larger (3.1.1a) and smaller (3.1.1b) lattice spacing.

Given that, at the leading order in perturbation theory, the lattice spacing depends on β in the following way:

$$a(\beta) \approx e^{-\frac{12\pi^2}{11N^2}\beta} = e^{-\frac{3\pi^2}{11}\beta} \quad (3.1.7)$$

the first plot corresponds to a higher value of a than the second one¹.

As can be easily seen, by lowering the lattice spacing equipotential surfaces tend to become circles, therefore the rotational invariance, that is broken in the lattice for any value of the lattice spacing, gets restored in the expectation value of the observables, making lattice field theory a “good” theory capable of making meaningful predictions.

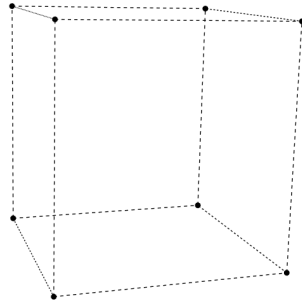
3.2 Other Types of Lattice

In Section 1.2.2.1 the Simple Hypercubic lattice has been defined. Of course, it is not the only possible choice, although it is the simplest. In fact, in order to further investigate the restoration of rotational symmetry and to make better predictions on rotational invariant quantities, other types of lattices have been used to simulate lattice field theories.

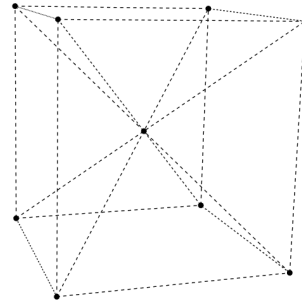
3.2.1 Body-Centered Tesseract

For example, the Body-Centered Tesseract (BCT) has been used for simulation of Yang-Mills theories. It consists of packing the spacetime with tesseracts, as the name suggests, but considering both the corners and the centers of every hypercube as lattice sites (see Figure 3.2.2 for a tridimensional representation of a cubic cell (3.2.2a) and a body-centered cubic cell (3.2.2b)).

¹The rigorous determination of the lattice spacing is a rather complicated matter that will not be explained further, as it is not the purpose of this project.



(a) Simple cube.



(b) Body-centered cube.

Figure 3.2.2: Tridimensional representation of a simple cubic cell (3.2.2a) and a body-centered one (3.2.2b).

Every site of the BCT lattice has, therefore, 24 nearest neighbours: 16 are identified by all possible sign permutations of $(\pm\frac{1}{2}, \pm\frac{1}{2}, \pm\frac{1}{2}, \pm\frac{1}{2})$, the 8 remaining are the ones of the SH lattice. The cell of this lattice is known as 24-cell, shown in Figure 3.2.3, and the plaquettes are triangular.

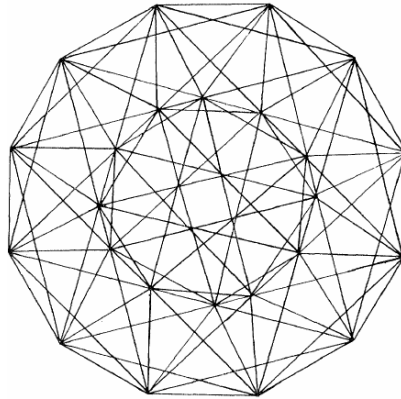


Figure 3.2.3: Bidimensional projection of the 24-cell.

This lattice has a 1152-elements symmetry group $G_{\Lambda_{BCT}}$, i.e., significantly more rotations and reflections than the 384 of the SH lattice.

3.2.2 F_4 Coroots Lattice

F_4 is one of the five exceptional simple Lie groups, with Dynkin diagram $\bullet \rightarrow \bullet \rightarrow \bullet \rightarrow \bullet$. A more detailed explanation of exceptional Lie groups and algebras can be found in [14]. Its root lattice is a 4-dimensional body-centered hypercubic lattice, a BCT, while its dual, that is called the F_4 coroots lattice, is the 4-dimensional lattice with the symmetry group of highest order. Each site of this lattice has 48 nearest neighbours:

- 24 corresponding to the roots of F_4 , individuated by all possible sign and position permutations of $(\pm 1, \pm 1, 0, 0)$;

- 24 corresponding to the coroots, the roots' dual vectors, individuated by
 - the 8 possible sign and coordinate permutations of $(\pm 1, 0, 0, 0)$
 - the 16 possible sign permutations of $(\pm \frac{1}{2}, \pm \frac{1}{2}, \pm \frac{1}{2}, \pm \frac{1}{2})$

This lattice is made up of two BCT lattices: the first one is the dual lattice, that is the same as Section 3.2.1, the other is the root lattice, that is a BCT with lattice spacing $\sqrt{2}$ as can be seen as represented in Figure 3.2.4.

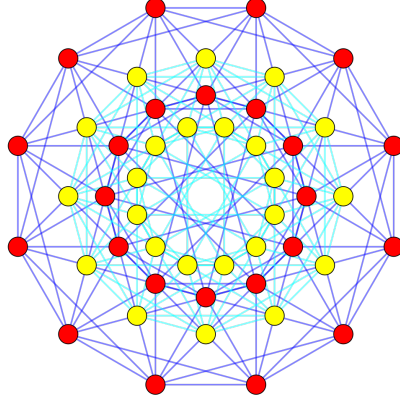


Figure 3.2.4: Bidimensional projection of the elementary cell of the F_4 coroots lattice. The roots are represented in red and the coroots in yellow.

This lattice has a symmetry group $G_{\Lambda_{F_4}}$ of order 2304, twice as the BCT symmetry group.

3.3 Simulations on Higher Symmetric Lattices

These lattices have been used, in literature, to perform Monte Carlo simulations of quantum field theories.

3.3.1 Scalar Fields on F_4 Lattice

In [15], Neuberger presented the formulation of scalar fields in terms of the F_4 lattice, where he discretized the action (1.1.24) assuming the simplest nearest neighbours interaction:

$$S = \frac{1}{6} \sum_{\langle xy \rangle \in \Lambda} (\phi(x) - \phi(y))^2 + \sum_{x \in \Lambda} \left(\frac{1}{2} m^2 \phi^2(x) + \frac{\lambda}{4} \phi^4(x) \right) \quad (3.3.1)$$

where $\langle xy \rangle$ indicates sum over nearest neighbours.

The Fourier transform of the field $\phi(x)$ in the limit of infinite lattice volume is given by:

$$\begin{aligned} \tilde{\phi}(k) &= \int d^4x e^{ik \cdot x} \phi(x) \\ \phi(x) &= \frac{1}{2(2\pi)^4} \int d^4k e^{-ik \cdot x} \tilde{\phi}(k) \end{aligned}$$

and the kinetic energy of (3.3.1) at infinite volume is:

$$\begin{aligned} \text{K.E.} &= \frac{1}{6(2\pi)^4} \int d^4k |\tilde{\phi}(k)|^2 \left(\sum_{\mu} (1 - \cos(k_{\mu})) + \sum_{\pm} \left(1 - \cos\left(\frac{k_1 \pm k_2 \pm k_3 \pm k_4}{2}\right) \right) \right) = \\ &= \frac{1}{12(2\pi)^4} \int d^4k |\tilde{\phi}(k)|^2 \left(k^2 - \frac{1}{72} k^4 + O(k^6) \right) \end{aligned} \quad (3.3.2)$$

The fact that the fourth-order term is proportional to the square of the second-order one is a signature of the symmetry-preserving nature of this lattice action.

This work lead to some analytical [16] and numerical [17] results and to an upper bound prediction on the mass of the Higgs boson [18] more than 20 years before its actual discovery.

3.3.2 Gauge Theories on the BCT Lattice

In [19], Celmaster presented the first computations for an $SU(2)$ theory on a BCT. Being a different lattice, where the plaquette is a triangle and not a square, a new action has to be defined. An action that is invariant under the BCT group, with the correct classical continuum limit is:

$$S_{BCT} = \frac{1}{2g^2} \sum_{\Delta} \text{Re Tr } U_{\Delta} = \frac{\beta}{8} \sum_{\Delta} \text{Re Tr } U_{\Delta} \quad (3.3.3)$$

where U_{Δ} is the triangular plaquette, defined as

$$U_{\Delta} = U_v(x)U_w(x + \hat{v})U_{-v-w}(x + \hat{v} + \hat{w}) = U_v(x)U_w(x + \hat{v})U_{v+w}^{\dagger}(x) \quad (3.3.4)$$

where v and w are labels indicating the possible nearest-neighbour directions, described in Section 3.2.1, such that $v + w$ is a valid direction.

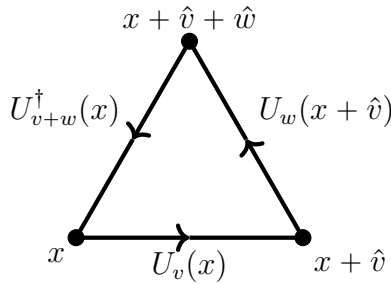


Figure 3.3.5: Schematization of an elementary triangular plaquette.

For example, if $v = (+1, 0, 0, 0)$ and $w = (0, +1, 0, 0)$, then they add up to $v + w = (+1, +1, 0, 0)$, that is not one of the nearest-neighbours vectors, whereas $v = (+1, 0, 0, 0)$ and $w = (-\frac{1}{2}, +\frac{1}{2}, -\frac{1}{2}, +\frac{1}{2})$ add up to $v + w = (+\frac{1}{2}, +\frac{1}{2}, -\frac{1}{2}, +\frac{1}{2})$, that is a valid direction. In Figure 3.3.5, an example of an elementary triangular plaquette is shown.

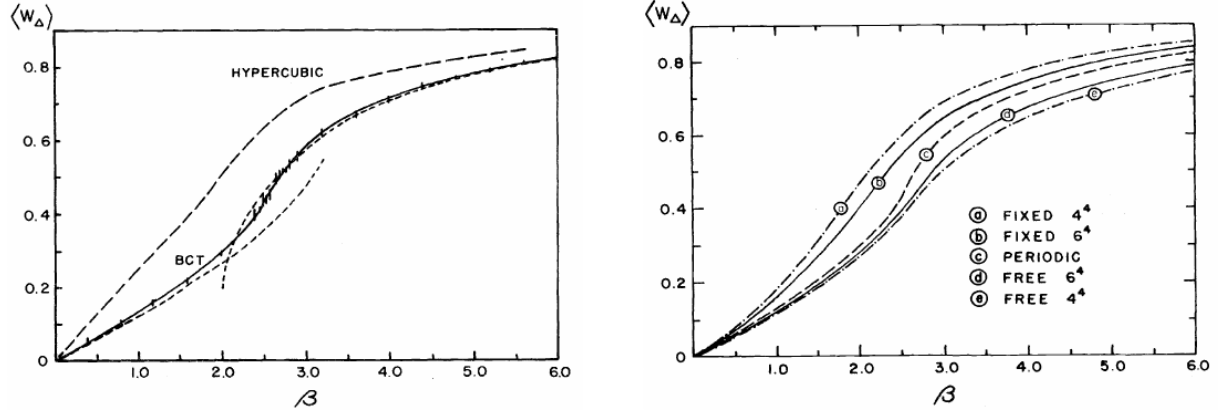
There are 96 such triangles touching each site, and each edge is shared by 8 triangles. By comparison, on the SH lattice there are 24 elementary square plaquettes touching each site and each edge is contiguous to 6 squares.

In [20] and [21] Celmaster presented the first simulation results for the $SU(2)$ gauge theory on the BCT lattice, explaining the algorithm used for the simulations in [22].

3.3.2.1 Average Plaquette Value on BCT

In [21] a study on the average value of the plaquette was presented: it is plotted w.r.t. β in Figure 3.3.6a.

As can be seen, the BCT average plaquette better follows the strong and weak coupling expansions, that are analytical approximations valid, respectively, for $\beta \rightarrow 0$ ($\Leftrightarrow g \rightarrow \infty$) and for $\beta \rightarrow \infty$ ($\Leftrightarrow g \rightarrow 0$).



(a) Comparison between $\langle W_\Delta \rangle$ on BCT and SH lattices. The short-dashed lines represent weak and strong coupling expansions.

(b) Dependence of $\langle W_\Delta \rangle$ on boundary conditions in different BCT lattices, with different boundary conditions.

Figure 3.3.6: Average value of the plaquette $\langle W_\Delta \rangle$ on a BCT lattice as a function of β .

In the same article the influence of the boundary conditions on the average value of the plaquette was also investigated. Three different types of boundary conditions were considered: periodic, that means that lattice has the topology of a 4-dimensional torus, free, that means that links on the boundary are free to take any $SU(2)$ value, and fixed, that means that links on the boundary are constrained to take a certain $SU(2)$ value. The results are plotted in Figure 3.3.6b.

From the plot, it can be seen that $\langle W_\Delta \rangle_{\text{free}} \leq \langle W_\Delta \rangle_{\text{periodic}} \leq \langle W_\Delta \rangle_{\text{fixed}}$, which is an inequality discussed by Mütter and Schilling in [23].

It was also pointed out that the computing time of BCT simulations was larger by a factor of approximately 2.5 as compared with SH simulations: this is due to having 3 times as many degrees of freedom, that become $\frac{11}{3}$ as many if a gauge fixing is done, and $\frac{16}{3}$ as many plaquettes for each site. However, having 12 symmetry axes instead of 4 provides a higher symmetry of the observables allowing for better checks on computations.

3.3.2.2 String Tension

In [20] the $SU(2)$ string tension was obtained from measurements of ratios of Wilson loops on a 6^4 BCT lattice.

On a BCT lattice, there are two types of Wilson loops:

- Rectangular Wilson loops $W_R(n, m) = \frac{1}{2} \langle \text{Tr } U_R(n, m) \rangle$
- Triangular Wilson loops $W_T(n) = \frac{1}{2} \langle \text{Tr } U_T(n) \rangle$

where $U_R(n, m)$ is the product of link variables over an $n \times m$ rectangle and $U_T(n)$ is the product of link variables over an equilateral triangle of side n .

Assuming that Wilson loops are fitted by:

$$W = e^{-\sigma A - b(\text{perimeter}) + \text{const.}} \quad (3.3.5)$$

where A is the area of the loop, then the string tension σ can be obtained through logarithmic ratios (also called *Creutz ratios*):

$$\chi(2) = -\ln\left(\frac{W_R(1, 1)W_R(2, 2)}{W_R^2(1, 2)}\right) \quad (3.3.6)$$

$$R_T(3) = -\frac{2}{\sqrt{3}}\ln\left(\frac{W_T(1)W_T(3)}{W_T^2(2)}\right) \quad (3.3.7)$$

Results of the simulations are plotted in Figure 3.3.7, assuming no correlations between loop fluctuations in the computation of the error bars.

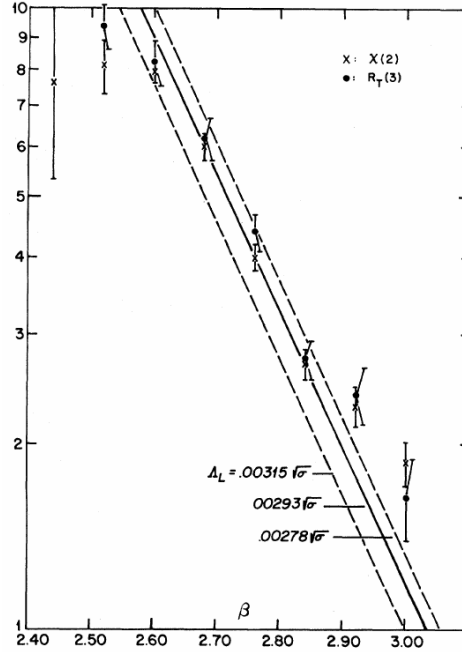


Figure 3.3.7: $SU(2)$ logarithmic loop ratios as a function of β on a 6^4 BCT lattice.

The logarithmic loop ratio $\chi(2)$ agrees with the asymptotic freedom curve in the region $2.68 \leq \beta \leq 2.84$ far better than any other result obtained with simulations on SH lattices. This is a consequence of the fact that the BCT action has smaller $O(a^2)$ corrections than usual at small β , therefore finite-spacing corrections are smaller for this action than for the action on a SH lattice. Therefore, BCT lattices with less sites than SH lattices could be used to obtain data with the same accuracy, at a given physical hypervolume.

Furthermore, $R_T(3)$ is compatible with $\chi(2)$ within the statistical uncertainties: this is a confirmation of the area-law dependence of equation (3.3.5).

In conclusion, simulations on BCT lattices provide better results, with more rotational invariance than those on other lattices, at a cost of a (slightly) higher computational time.

Simulations Results

In this chapter original results from simulations are presented. The code used to perform such simulations is based on the code originally developed for the calculations presented in refs. [24, 25].

4.1 Rotational Invariance Restoration

This first section has the aim to reproduce the rotational invariance restoration of [13] for the continuous group $SU(2)$ (and not for the discrete icosahedral group \tilde{Y} , like explained in Section 3.1.1).

This is done by evaluating the correlator of two Polyakov loops as explained in Section 2.4.2, computed on two different lattices with two different values of β , corresponding to two different lattice spacings.

4.1.1 Simulations Setup

The lattices are taken to be periodic in every direction, with n_s sites in each space direction ($n_s = n_x = n_y = n_z$) and n_t sites in the time direction. The lattice has, therefore, $n_s^3 n_t$ sites.

The simulations are run from a cold start, with 2500 thermalization steps, where each Monte Carlo step is composed of 1 heat bath step followed by 3 overrelaxation steps. Since the plaquette is observed to thermalize after $O(10)$ steps in preliminary simulations

with both hot and cold starts, 2500 thermalization steps are enough to ensure full thermalization.

After that, 20000 measurements are taken of every possible independent correlator between two Polyakov loops, with 100 updates between each measurement.

On a lattice with n_s spatial sites in each direction, only couples of Polyakov loops whose distance, in each spatial direction, is $\leq n_s/2$ are independent: for example a couple of Polyakov loops extending for $n_s/2 + 1$ sites in a certain direction is equivalent to the hermitian conjugate of a couple extending for $n_s/2 - 1$ sites, because of the periodic boundary conditions (see (2.4.2)). But, since the Polyakov loops correlator expectation value is real, they have the same numerical value.

In Figure 4.1.1 is represented a graphical visualization of this fact.

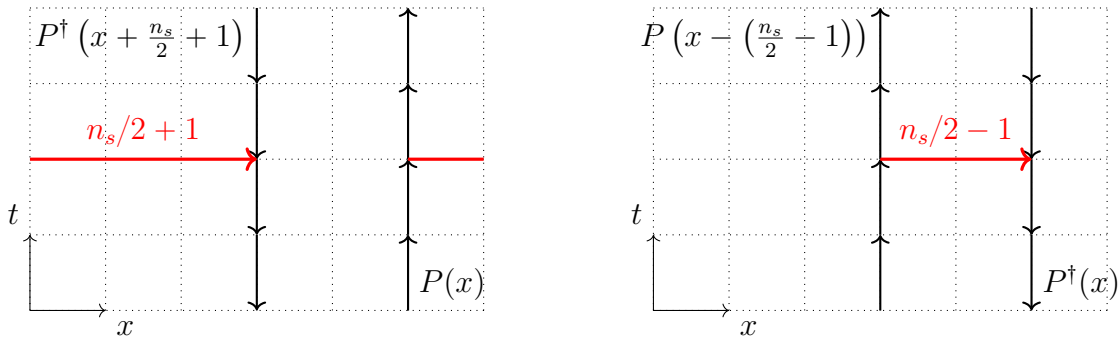


Figure 4.1.1: The correlator of two Polyakov loops distant $n_s/2 + 1$ sites (left) is equal to the hermitian conjugate of the correlator of two Polyakov loops distant $n_s/2 - 1$ sites (right), i.e., $\langle P(x)P^\dagger(x + \frac{n_s}{2} + 1) \rangle = \langle P(x - (\frac{n_s}{2} - 1))P^\dagger(x) \rangle^\dagger$.

Therefore, only Polyakov correlators extending from size $(0, 0, 0)$ to size $(n_s/2, n_s/2, n_s/2)$ are considered. Correlators of the same size are then averaged together and the uncorrelated empirical standard deviation is computed as an estimate of the error.

4.1.2 Analysis of Data

After checking that each correlator's mean value is real (its imaginary part is compatible with 0), these averages are used to compute the potential, in units of the lattice spacing a , as:

$$V(x, y, z) = -\frac{1}{n_t} \ln \langle P(0)P^\dagger(x, y, z) \rangle \quad (4.1.1)$$

where $(x, y, z) \in \{(0, 0, 0), \dots, (\frac{n_s}{2}, \frac{n_s}{2}, \frac{n_s}{2})\}$. The standard deviation is computed with the usual error propagation formula.

Graphics like Figure 3.1.1 are obtained considering a section of the lattices used, specifically the xy -plane, that is done by plotting only values of $V(x, y, z = 0)$.

In order to verify that there are no anisotropies, all the three coordinate planes are plotted in Figure 4.1.2.

Corresponding points in each plane are compatible with each other within the standard deviation and there is no visible difference between different coordinate planes. This justifies choosing arbitrarily one plane for each lattice.

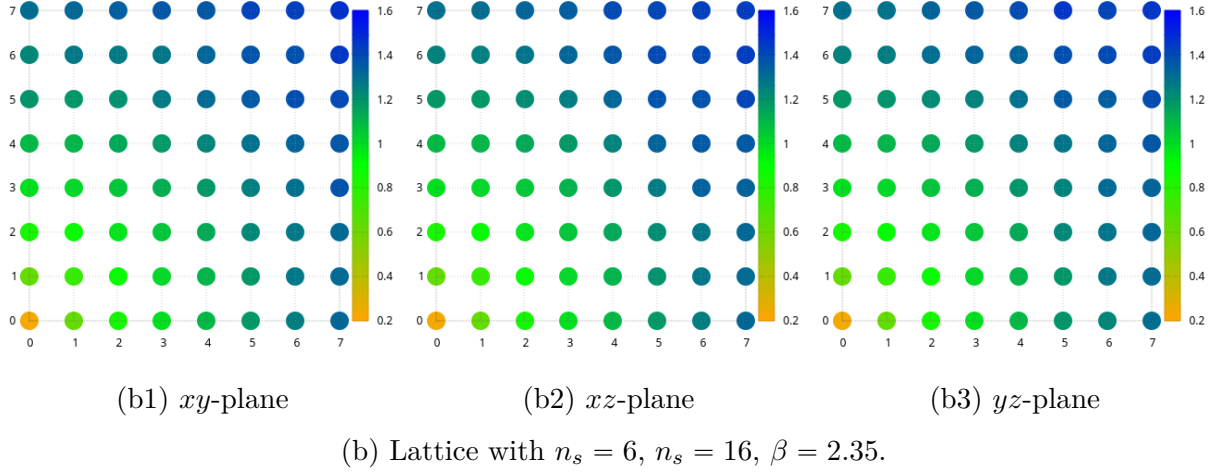
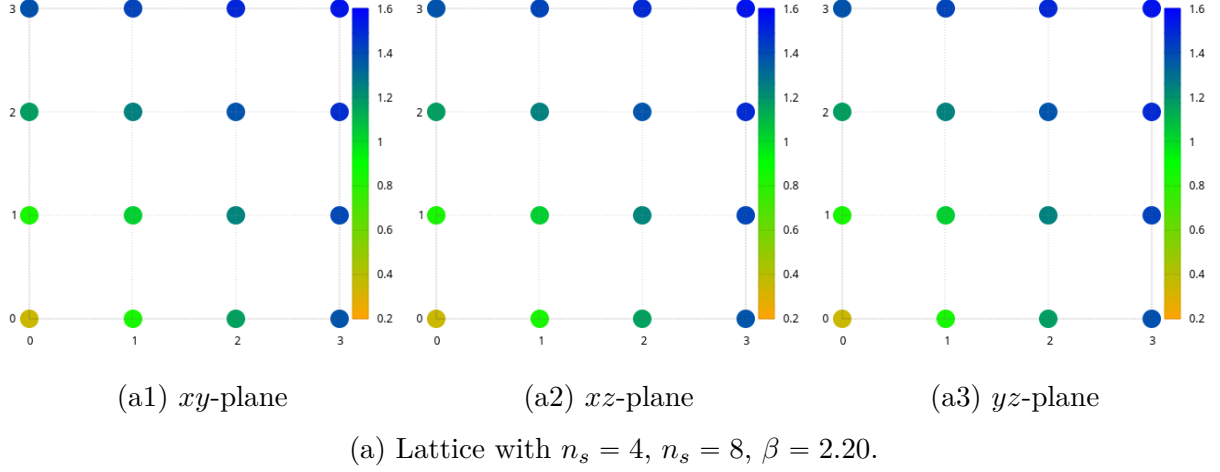
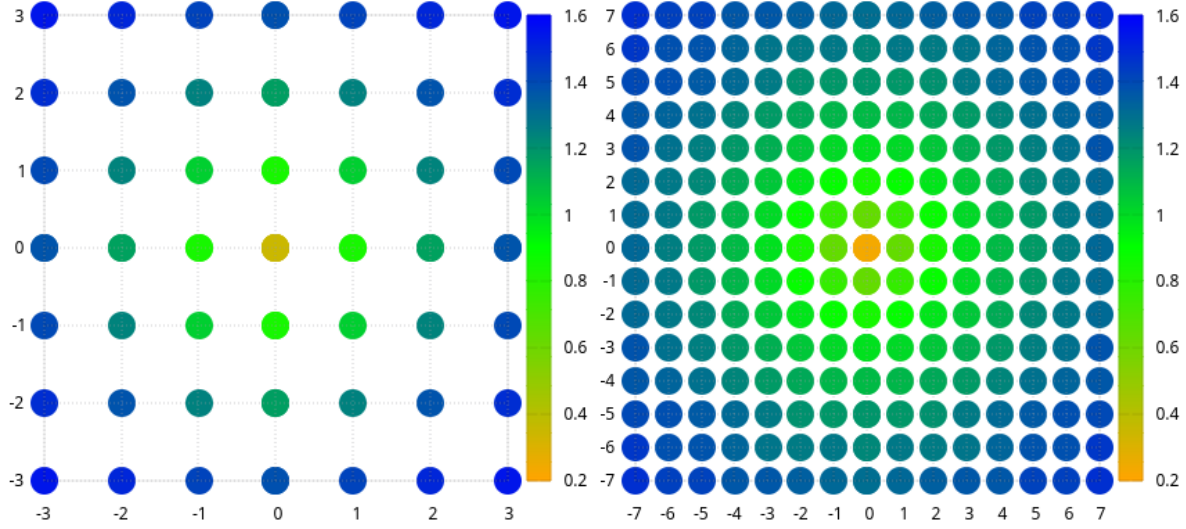


Figure 4.1.2: Plots of the static quark potential in the three coordinate planes for two different lattices. The colored scale represent the potential in lattice spacing units.



(a) Lattice with $n_s = 4$, $n_s = 8$, $\beta = 2.20$. (b) Lattice with $n_s = 6$, $n_s = 16$, $\beta = 2.35$.

Figure 4.1.3: Rotational invariance restoration of the static quark potential.

In order to give a more “complete” view, data from Figure 4.1.2a1 and Figure 4.1.2b1 is reflected relative to the x and y axis, obtaining the plots in Figure 4.1.3.

In both Figure 4.1.2 and Figure 4.1.3 correlators of Polyakov loops far from the origin (the ones represented by blue dots, with values of the potential $\gtrsim 1.4$) were compatible with each other. The interesting parts of these graphics are, therefore, the dots up to the dark green shade.

4.1.3 Final Remarks

It is interesting to note the resemblance of Figure 4.1.3 with Figure 3.1.1, although they are obtained in different ways and for different gauge groups. Note also that the values of β used in these simulations were slightly higher than the ones used in [13], in order to have a smaller lattice spacing that allowed a finer determination of the potential.

4.2 BCT Lattice

The code used for simulations on SH lattices has been adapted to simulate $SU(N)$ gauge theories on the BCT lattice.

4.2.1 Formulation of the Algorithm

The first obvious change was the number of sites: being body-centered, this lattice has twice the sites of a SH with the same dimensions.

Let us assume the lattice has L_μ sites in the $\mu = t, x, y, z$ direction. The total number of sites is, therefore, $N_{sites} = L_t L_x L_y L_z$ for a SH lattice and $N_{sites} = 2L_t L_x L_y L_z$ for a BCT.

4.2.1.1 Lattice Sites

Each site has coordinates (t, x, y, z) where $t = 0, 1, \dots, L_t - 1$, $x = 0, 1, \dots, L_x - 1$, $y = 0, 1, \dots, L_y - 1$ and $z = 0, 1, \dots, L_z - 1$ if the lattice is a simple hypercube. In the old code, each site was individuated by an integer $0 \leq s < N_{sites}$ obtained through the following computation:

$$s = z + L_z (y + L_y (x + L_x t)) \quad (4.2.1)$$

and the coordinates could be obtained as:

$$\begin{aligned} z &= s \mod L_z \\ y &= \frac{s - z}{L_z} \mod L_y \\ x &= \left(\left(\frac{s - z}{L_z} - y \right) / L_y \right) \mod L_x \\ t &= \left(\left(\frac{s - z}{L_z} - y \right) / L_y - x \right) / L_x \end{aligned} \quad (4.2.2)$$

On a BCT lattice, each site has either all integer or all semi-integer coordinates. If they are all integer, s is computed as:

$$s = 2(z + L_z (y + L_y (x + L_x t))) \quad (4.2.3)$$

Otherwise, it is computed as:

$$s = 1 + 2([z] + L_z ([y] + L_y ([x] + L_x [t]))) \quad (4.2.4)$$

where $[x]$ represents the *integer part* of x .

The inverted relation is the same as (4.2.2), but with s replaced with $s/2$ if s is even, otherwise, if s is odd, it becomes:

$$\begin{aligned} z &= \left(\frac{s - 1}{2} \mod L_z \right) + 0.5 \\ y &= \left(\frac{\frac{s-1}{2} - [z]}{L_z} \mod L_y \right) + 0.5 \\ x &= \left(\left(\left(\frac{\frac{s-1}{2} - [z]}{L_z} - [y] \right) / L_y \right) \mod L_x \right) + 0.5 \\ t &= \left(\left(\left(\frac{\frac{s-1}{2} - [z]}{L_z} - [y] \right) / L_y - [x] \right) / L_x \right) + 0.5 \end{aligned} \quad (4.2.5)$$

In the code, the integer s is stored in the variable `site` and the coordinates t, x, y, z are always stored in integer variables as their integer part. An additional variable `halfInt`, that can only assume 0 or 1 values, is used to keep track if the coordinates are actual integers or semi-integers.

4.2.1.2 Directions

Another important part is how the different possible directions are referred to in the code. Each site in the SH lattice has only 8 nearest neighbours, individuated by 4 different

directions, the vectors $(1, 0, 0, 0)$ (direction 0), $(0, 1, 0, 0)$ (direction 1), $(0, 0, 1, 0)$ (direction 2), $(0, 0, 0, 1)$ (direction 3), and their opposites. In the old version of the code, two arrays of dimension $4N_{sites}$, `neighbour_plus` and `neighbour_minus`, were used to keep track of each site's nearest neighbours.

As explained in Section 3.2.1, each site has 24 nearest neighbours, in 12 different directions. The two arrays are therefore changed to size $12N_{sites}$, where the first 4 directions (0 to 3) are the same as the SH lattice and the 8 new directions are:

Direction	<code>dir_plus</code>	<code>dir_minus</code>
4	$(+1/2, +1/2, +1/2, +1/2)$	$(-1/2, -1/2, -1/2, -1/2)$
5	$(+1/2, +1/2, +1/2, -1/2)$	$(-1/2, -1/2, -1/2, +1/2)$
6	$(+1/2, +1/2, -1/2, +1/2)$	$(-1/2, -1/2, +1/2, -1/2)$
7	$(+1/2, -1/2, +1/2, +1/2)$	$(-1/2, +1/2, -1/2, -1/2)$
8	$(-1/2, +1/2, +1/2, +1/2)$	$(+1/2, -1/2, -1/2, -1/2)$
9	$(+1/2, +1/2, -1/2, -1/2)$	$(-1/2, -1/2, +1/2, +1/2)$
10	$(+1/2, -1/2, -1/2, +1/2)$	$(-1/2, +1/2, +1/2, -1/2)$
11	$(+1/2, -1/2, +1/2, -1/2)$	$(-1/2, +1/2, -1/2, +1/2)$

This is done in

Conclusions

Code

File initialize.cc:

```
1 void initialize() {
2
3     int t, x, y, site, dir, next_t, next_x, next_y, next_site, i;
4     #if dim==4
5         int z, next_z;
6     #endif
7     #ifdef __F4RootLattice__
8         int halfInt, next_halfInt;
9     #endif
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107     for (t=0;t<nt;t++)
108         for (x=0;x<nx;x++)
109             for (y=0;y<ny;y++)
110                 #if dim==4
111                     for (z=0;z<nz;z++)
112                         #endif
113                         #ifdef __F4RootLattice__
114                             for (halfInt=0;halfInt<2;halfInt++)
115                                 #endif
116                                 {
117 #ifndef __F4RootLattice__
118                 site=y+ny*(x+nx*t);
119                 #if dim==4
120                     site*=nz;
121                     site+=z;
122                 #endif
123             }
124 }
```

```

133 #else
134     site=(z+nz*(y+ny*(x+nx*t)))*2+halfInt;
135 #endif
136     for (dir=0;dir<maxDir;dir++) {

160         next_t=t;
161         next_x=x;
162         next_y=y;
163 #if dim==4
164         next_z=z;
165 #endif
166 #ifdef __F4RootLattice__
167         next_halfInt=halfInt;
168 #endif
169         switch (dir) {
170             case 0: {
171                 next_t=(t+1)%nt;
172 #ifndef __F4RootLattice__
173                 next_site=next_y+ny*(next_x+nx*next_t);
174 #if dim==4
175                 next_site*=nz;
176                 next_site+=next_z;
177 #endif
178 #else
179                 next_site=next_halfInt+2*(next_z+nz*(next_y+ny*(next_x+nx*
180                     next_t)));
181 #endif
182                 neighbor_plus[dir*nsites+site]=next_site;
183                 next_t=(t+nt-1)%nt;
184 #ifndef __F4RootLattice__
185                 next_site=next_y+ny*(next_x+nx*next_t);
186 #if dim==4
187                 next_site*=nz;
188                 next_site+=next_z;
189 #endif
190 #else
191                 next_site=next_halfInt+2*(next_z+nz*(next_y+ny*(next_x+nx*
192                     next_t)));
193 #endif
194                 neighbor_minus[dir*nsites+site]=next_site;
195                 break;
196             }
197             case 1: {
198                 next_x=(x+1)%nx;
199 #ifndef __F4RootLattice__
200                 next_site=next_y+ny*(next_x+nx*next_t);
201 #if dim==4
202                 next_site*=nz;
203                 next_site+=next_z;
204 #endif
205 #else
206                 next_site=next_halfInt+2*(next_z+nz*(next_y+ny*(next_x+nx*
207                     next_t)));
208 #endif
209                 neighbor_plus[dir*nsites+site]=next_site;
210                 next_x=(x+nx-1)%nx;

```

```

208 #ifndef __F4RootLattice__
209     next_site=next_y+ny*(next_x+nx*next_t);
210 #if dim==4
211     next_site*=nz;
212     next_site+=next_z;
213 #endif
214 #else
215     next_site=next_halfInt+2*(next_z+nz*(next_y+ny*(next_x+nx*
216     next_t)));
217 #endif
218     neighbor_minus[dir*nsites+site]=next_site;
219     break;
220 }
221 case 2: {
222     next_y=(y+1)%ny;
223 #ifndef __F4RootLattice__
224     next_site=next_y+ny*(next_x+nx*next_t);
225 #if dim==4
226     next_site*=nz;
227     next_site+=next_z;
228 #endif
229 #else
230     next_site=next_halfInt+2*(next_z+nz*(next_y+ny*(next_x+nx*
231     next_t)));
232 #endif
233     neighbor_plus[dir*nsites+site]=next_site;
234     next_y=(y+ny-1)%ny;
235 #ifndef __F4RootLattice__
236     next_site=next_y+ny*(next_x+nx*next_t);
237 #if dim==4
238     next_site*=nz;
239     next_site+=next_z;
240 #endif
241 #else
242     next_site=next_halfInt+2*(next_z+nz*(next_y+ny*(next_x+nx*
243     next_t)));
244 #endif
245     neighbor_minus[dir*nsites+site]=next_site;
246     break;
247 }
248 #if dim==4
249 case 3: {
250     next_z=(z+1)%nz;
251 #ifndef __F4RootLattice__
252     next_site=next_z+nz*(next_y+ny*(next_x+nx*next_t));
253 #else
254     next_site=next_halfInt+2*(next_z+nz*(next_y+ny*(next_x+nx*
255     next_t)));
256 #endif
257     neighbor_plus[dir*nsites+site]=next_site;
258     next_z=(z+nz-1)%nz;
259 #ifndef __F4RootLattice__
260     next_site=next_z+nz*(next_y+ny*(next_x+nx*next_t));
261 #else
262 
```

```

258         next_site=next_halfInt+2*(next_z+nz*(next_y+ny*(next_x+nx*
        next_t)));
259 #endif
260         neighbor_minus[dir*nsites+site]=next_site;
261         break;
262     }
263 #endif
264 #ifdef __F4RootLattice__
265     case 4: { // (+0.5 +0.5 +0.5 +0.5)
266         if(halfInt){
267             next_halfInt=0;
268             next_t=(t+1)%nt;
269             next_x=(x+1)%nx;
270             next_y=(y+1)%ny;
271             next_z=(z+1)%nz;
272         }
273         else next_halfInt=1;
274         next_site=next_halfInt+2*(next_z+nz*(next_y+ny*(next_x+nx*
        next_t)));
275         neighbor_plus[dir*nsites+site]=next_site;
276         next_halfInt=halfInt;
277         next_t=t;
278         next_x=x;
279         next_y=y;
280         next_z=z;
281         if(halfInt) next_halfInt=0;
282         else{
283             next_halfInt=1;
284             next_t=(t+nt-1)%nt;
285             next_x=(x+nx-1)%nx;
286             next_y=(y+ny-1)%ny;
287             next_z=(z+nz-1)%nz;
288         }
289         next_site=next_halfInt+2*(next_z+nz*(next_y+ny*(next_x+nx*
        next_t)));
290         neighbor_minus[dir*nsites+site]=next_site;
291         break;
292     }
293
294     case 5: { // (+0.5 +0.5 +0.5 -0.5)
295         if(halfInt){
296             next_halfInt=0;
297             next_t=(t+1)%nt;
298             next_x=(x+1)%nx;
299             next_y=(y+1)%ny;
300         }
301         else{
302             next_halfInt=1;
303             next_z=(z+nz-1)%nz;
304         }
305         next_site=next_halfInt+2*(next_z+nz*(next_y+ny*(next_x+nx*
        next_t)));
306         neighbor_plus[dir*nsites+site]=next_site;
307         next_halfInt=halfInt;
308         next_t=t;

```

```

309     next_x=x;
310     next_y=y;
311     next_z=z;
312     if(halfInt){
313         next_halfInt=0;
314         next_z=(z+1)%nz;
315     }
316     else{
317         next_halfInt=1;
318         next_t=(t+nt-1)%nt;
319         next_x=(x+nx-1)%nx;
320         next_y=(y+ny-1)%ny;
321     }
322     next_site=next_halfInt+2*(next_z+nz*(next_y+ny*(next_x+nx*
next_t)));
323     neighbor_minus[dir*nsites+site]=next_site;
324     break;
325 }
326
327 case 6: { // (+0.5 +0.5 -0.5 +0.5)
328     if(halfInt){
329         next_halfInt=0;
330         next_t=(t+1)%nt;
331         next_x=(x+1)%nx;
332         next_z=(z+1)%nz;
333     }
334     else{
335         next_halfInt=1;
336         next_y=(y+ny-1)%ny;
337     }
338     next_site=next_halfInt+2*(next_z+nz*(next_y+ny*(next_x+nx*
next_t)));
339     neighbor_plus[dir*nsites+site]=next_site;
340     next_halfInt=halfInt;
341     next_t=t;
342     next_x=x;
343     next_y=y;
344     next_z=z;
345     if(halfInt){
346         next_halfInt=0;
347         next_y=(y+1)%ny;
348     }
349     else{
350         next_halfInt=1;
351         next_t=(t+nt-1)%nt;
352         next_x=(x+nx-1)%nx;
353         next_z=(z+nz-1)%nz;
354     }
355     next_site=next_halfInt+2*(next_z+nz*(next_y+ny*(next_x+nx*
next_t)));
356     neighbor_minus[dir*nsites+site]=next_site;
357     break;
358 }
359
360 case 7: { // (+0.5 -0.5 +0.5 +0.5)

```

```

361         if(halfInt){
362             next_halfInt=0;
363             next_t=(t+1)%nt;
364             next_y=(y+1)%ny;
365             next_z=(z+1)%nz;
366         }
367         else{
368             next_halfInt=1;
369             next_x=(x+nx-1)%nx;
370         }
371         next_site=next_halfInt+2*(next_z+nz*(next_y+ny*(next_x+nx*
next_t)));
372         neighbor_plus[dir*nsites+site]=next_site;
373         next_halfInt=halfInt;
374         next_t=t;
375         next_x=x;
376         next_y=y;
377         next_z=z;
378         if(halfInt){
379             next_halfInt=0;
380             next_x=(x+1)%nx;
381         }
382         else{
383             next_halfInt=1;
384             next_t=(t+nt-1)%nt;
385             next_y=(y+ny-1)%ny;
386             next_z=(z+nz-1)%nz;
387         }
388         next_site=next_halfInt+2*(next_z+nz*(next_y+ny*(next_x+nx*
next_t)));
389         neighbor_minus[dir*nsites+site]=next_site;
390         break;
391     }
392
393     case 8: { // (-0.5 +0.5 +0.5 +0.5)
394         if(halfInt){
395             next_halfInt=0;
396             next_x=(x+1)%nx;
397             next_y=(y+1)%ny;
398             next_z=(z+1)%nz;
399         }
400         else{
401             next_halfInt=1;
402             next_t=(t+nt-1)%nt;
403         }
404         next_site=next_halfInt+2*(next_z+nz*(next_y+ny*(next_x+nx*
next_t)));
405         neighbor_plus[dir*nsites+site]=next_site;
406         next_halfInt=halfInt;
407         next_t=t;
408         next_x=x;
409         next_y=y;
410         next_z=z;
411         if(halfInt){
412             next_halfInt=0;

```

```

413         next_t=(t+1)%nt;
414     }
415     else{
416         next_halfInt=1;
417         next_x=(x+nx-1)%nx;
418         next_y=(y+ny-1)%ny;
419         next_z=(z+nz-1)%nz;
420     }
421     next_site=next_halfInt+2*(next_z+nz*(next_y+ny*(next_x+nx*
next_t)));
422     neighbor_minus[dir*nsites+site]=next_site;
423     break;
424 }
425
426 case 9: { // (+0.5 +0.5 -0.5 -0.5)
427     if(halfInt){
428         next_halfInt=0;
429         next_t=(t+1)%nt;
430         next_x=(x+1)%nx;
431     }
432     else{
433         next_halfInt=1;
434         next_y=(y+ny-1)%ny;
435         next_z=(z+nz-1)%nz;
436     }
437     next_site=next_halfInt+2*(next_z+nz*(next_y+ny*(next_x+nx*
next_t)));
438     neighbor_plus[dir*nsites+site]=next_site;
439     next_halfInt=halfInt;
440     next_t=t;
441     next_x=x;
442     next_y=y;
443     next_z=z;
444     if(halfInt){
445         next_halfInt=0;
446         next_y=(y+1)%ny;
447         next_z=(z+1)%nz;
448     }
449     else{
450         next_halfInt=1;
451         next_t=(t+nt-1)%nt;
452         next_x=(x+nx-1)%nx;
453     }
454     next_site=next_halfInt+2*(next_z+nz*(next_y+ny*(next_x+nx*
next_t)));
455     neighbor_minus[dir*nsites+site]=next_site;
456     break;
457 }
458
459 case 10: { // (+0.5 -0.5 -0.5 +0.5)
460     if(halfInt){
461         next_halfInt=0;
462         next_t=(t+1)%nt;
463         next_z=(z+1)%nz;
464     }

```

```

465     else{
466         next_halfInt=1;
467         next_x=(x+nx-1)%nx;
468         next_y=(y+ny-1)%ny;
469     }
470     next_site=next_halfInt+2*(next_z+nz*(next_y+ny*(next_x+nx*
next_t)));
471     neighbor_plus[dir*nsites+site]=next_site;
472     next_halfInt=halfInt;
473     next_t=t;
474     next_x=x;
475     next_y=y;
476     next_z=z;
477     if(halfInt){
478         next_halfInt=0;
479         next_x=(x+1)%nx;
480         next_y=(y+1)%ny;
481     }
482     else{
483         next_halfInt=1;
484         next_t=(t+nt-1)%nt;
485         next_z=(z+nz-1)%nz;
486     }
487     next_site=next_halfInt+2*(next_z+nz*(next_y+ny*(next_x+nx*
next_t)));
488     neighbor_minus[dir*nsites+site]=next_site;
489     break;
490 }
491
492 case 11: { // (+0.5 -0.5 +0.5 -0.5)
493     if(halfInt){
494         next_halfInt=0;
495         next_t=(t+1)%nt;
496         next_y=(y+1)%ny;
497     }
498     else{
499         next_halfInt=1;
500         next_x=(x+nx-1)%nx;
501         next_z=(z+nz-1)%nz;
502     }
503     next_site=next_halfInt+2*(next_z+nz*(next_y+ny*(next_x+nx*
next_t)));
504     neighbor_plus[dir*nsites+site]=next_site;
505     next_halfInt=halfInt;
506     next_t=t;
507     next_x=x;
508     next_y=y;
509     next_z=z;
510     if(halfInt){
511         next_halfInt=0;
512         next_x=(x+1)%nx;
513         next_z=(z+1)%nz;
514     }
515     else{
516         next_halfInt=1;

```



```

517         next_t=(t+nt-1)%nt;
518         next_y=(y+ny-1)%ny;
519     }
520     next_site=next_halfInt+2*(next_z+nz*(next_y+ny*(next_x+nx*
next_t)));
521     neighbor_minus[dir*nsites+site]=next_site;
522     break;
523 }

668 #endif
669     default: {
670         break;
671     }
672 }

677     if (type_of_start==0) {
678         for (i=0;i<Ncolsquare;i++) {
679             v[i]=dc(1.-2.*random_real(xx),1.-2.*random_real(xx));
680         }
681         norm_sun(v);
682         for (i=0;i<Ncolsquare;i++) {
683             ufield[(dir*nsites+site)*Ncolsquare+i]=v[i];
684         }
685     }

686     if (type_of_start==1) {
687         for (i=0;i<Ncolsquare;i++) {
688             ufield[(dir*nsites+site)*Ncolsquare+i]=dc(0.,0.);
689         }
690         for (i=0;i<Ncolsquare;i+=Ncol_plus_one) {
691             ufield[(dir*nsites+site)*Ncolsquare+i]=dc(1.,0.);
692         }
693     }
694 }

743 #ifdef __F4RootLattice__ // initializing vector with all possible
744 staples in each direction
745 for(dir=0; dir<maxDir; dir++) switch(dir){
746     case 0:{
747         stapleDirs[dir*nStaples+ 0]= +4; stapleDirs[dir*nStaples+ 1]= -8;
748         stapleDirs[dir*nStaples+ 2]= +5; stapleDirs[dir*nStaples+ 3]=+10;
749         stapleDirs[dir*nStaples+ 4]= +6; stapleDirs[dir*nStaples+ 5]=+11;
750         stapleDirs[dir*nStaples+ 6]= +7; stapleDirs[dir*nStaples+ 7]= +9;

751         break;
752     }
753     case 1:{
754         stapleDirs[dir*nStaples+ 0]= +4; stapleDirs[dir*nStaples+ 1]= -7;
755         stapleDirs[dir*nStaples+ 2]= +5; stapleDirs[dir*nStaples+ 3]=-11;
756         stapleDirs[dir*nStaples+ 4]= +6; stapleDirs[dir*nStaples+ 5]=-10;
757         stapleDirs[dir*nStaples+ 6]= +8; stapleDirs[dir*nStaples+ 7]= +9;

758         break;
759     }
760     case 2:{
761         stapleDirs[dir*nStaples+ 0]= +4; stapleDirs[dir*nStaples+ 1]= -6;
762         stapleDirs[dir*nStaples+ 2]= +5; stapleDirs[dir*nStaples+ 3]= -9;

```

```

778     stapleDirs[dir*nStaples+ 4]= +7; stapleDirs[dir*nStaples+ 5]=-10;
779     stapleDirs[dir*nStaples+ 6]= +8; stapleDirs[dir*nStaples+ 7]=+11;

788     break;
789 }
790 case 3:{
791     stapleDirs[dir*nStaples+ 0]= +4; stapleDirs[dir*nStaples+ 1]= -5;
792     stapleDirs[dir*nStaples+ 2]= +6; stapleDirs[dir*nStaples+ 3]= -9;
793     stapleDirs[dir*nStaples+ 4]= +7; stapleDirs[dir*nStaples+ 5]=-11;
794     stapleDirs[dir*nStaples+ 6]= +8; stapleDirs[dir*nStaples+ 7]=+10;

758     break;
759 }
760 case 1:{
761     stapleDirs[dir*nStaples+ 0]= +4; stapleDirs[dir*nStaples+ 1]= -7;
762     stapleDirs[dir*nStaples+ 2]= +5; stapleDirs[dir*nStaples+ 3]=-11;
763     stapleDirs[dir*nStaples+ 4]= +6; stapleDirs[dir*nStaples+ 5]=-10;
764     stapleDirs[dir*nStaples+ 6]= +8; stapleDirs[dir*nStaples+ 7]= +9;

803     break;
804 }
805 case 4:{
806     stapleDirs[dir*nStaples+ 0]=+24; stapleDirs[dir*nStaples+ 1]= +8;
807     stapleDirs[dir*nStaples+ 2]= +1; stapleDirs[dir*nStaples+ 3]= +7;
808     stapleDirs[dir*nStaples+ 4]= +2; stapleDirs[dir*nStaples+ 5]= +6;
809     stapleDirs[dir*nStaples+ 6]= +3; stapleDirs[dir*nStaples+ 7]= +5;

818     break;
819 }
820 case 5:{
821     stapleDirs[dir*nStaples+ 0]=+24; stapleDirs[dir*nStaples+ 1]=-10;
822     stapleDirs[dir*nStaples+ 2]= +1; stapleDirs[dir*nStaples+ 3]=+11;
823     stapleDirs[dir*nStaples+ 4]= +2; stapleDirs[dir*nStaples+ 5]= +9;
824     stapleDirs[dir*nStaples+ 6]= -3; stapleDirs[dir*nStaples+ 7]= +4;

833     break;
834 }
835 case 6:{
836     stapleDirs[dir*nStaples+ 0]=+24; stapleDirs[dir*nStaples+ 1]=-11;
837     stapleDirs[dir*nStaples+ 2]= +1; stapleDirs[dir*nStaples+ 3]=+10;
838     stapleDirs[dir*nStaples+ 4]= -2; stapleDirs[dir*nStaples+ 5]= +4;
839     stapleDirs[dir*nStaples+ 6]= +3; stapleDirs[dir*nStaples+ 7]= +9;

848     break;
849 }
850 case 7:{
851     stapleDirs[dir*nStaples+ 0]=+24; stapleDirs[dir*nStaples+ 1]= -9;
852     stapleDirs[dir*nStaples+ 2]= -1; stapleDirs[dir*nStaples+ 3]= +4;
853     stapleDirs[dir*nStaples+ 4]= +2; stapleDirs[dir*nStaples+ 5]=+10;
854     stapleDirs[dir*nStaples+ 6]= +3; stapleDirs[dir*nStaples+ 7]=+11;

863     break;
864 }
865 case 8:{
866     stapleDirs[dir*nStaples+ 0]=-24; stapleDirs[dir*nStaples+ 1]= +4;
867     stapleDirs[dir*nStaples+ 2]= +1; stapleDirs[dir*nStaples+ 3]= -9;

```

```

868     stapleDirs[dir*nStaples+ 4]= +2; stapleDirs[dir*nStaples+ 5]=-11;
869     stapleDirs[dir*nStaples+ 6]= +3; stapleDirs[dir*nStaples+ 7]=-10;

878     break;
879 }
880 case 9:{
881     stapleDirs[dir*nStaples+ 0]=+24; stapleDirs[dir*nStaples+ 1]= -7;
882     stapleDirs[dir*nStaples+ 2]= +1; stapleDirs[dir*nStaples+ 3]= -8;
883     stapleDirs[dir*nStaples+ 4]= -2; stapleDirs[dir*nStaples+ 5]= +5;
884     stapleDirs[dir*nStaples+ 6]= -3; stapleDirs[dir*nStaples+ 7]= +6;

893     break;
894 }
895 case 10:{
896     stapleDirs[dir*nStaples+ 0]=+24; stapleDirs[dir*nStaples+ 1]= -5;
897     stapleDirs[dir*nStaples+ 2]= -1; stapleDirs[dir*nStaples+ 3]= +6;
898     stapleDirs[dir*nStaples+ 4]= -2; stapleDirs[dir*nStaples+ 5]= +7;
899     stapleDirs[dir*nStaples+ 6]= +3; stapleDirs[dir*nStaples+ 7]= -8;

908     break;
909 }
910 case 11:{
911     stapleDirs[dir*nStaples+ 0]=+24; stapleDirs[dir*nStaples+ 1]= -6;
912     stapleDirs[dir*nStaples+ 2]= -1; stapleDirs[dir*nStaples+ 3]= +5;
913     stapleDirs[dir*nStaples+ 4]= +2; stapleDirs[dir*nStaples+ 5]= -8;
914     stapleDirs[dir*nStaples+ 6]= -3; stapleDirs[dir*nStaples+ 7]= +7;

923     break;
924 }

1059     default:{
1060         break;
1061     }
1062 }
1063 #endif

1141
1142 }
```

File f4pstaple.cc:

```

1 void f4pstaple(dc *stot, int site, int mu, int first_direction, int
   use_smeared) {
2
3 // Computes the triangular staples from (site) to (site+a mu) in the
   positive
4 // directions (starting from first_direction) and stores their sum into
   stot.
5
6 int i, j, nextSite, vertex, dir1, dir2, aux_ind1, aux_ind2;
7 bool dagger1, dagger2;
8
9 nextSite=neighbor_plus[mu*nsites+site];
10
11 for (i=0;i<Ncolsquare;i++) {
12     stot[i]=dc(0.,0.);
13 }
14
15 for(j=0; j<nStaples; j+=2){
16     dir1 = stapleDirs[mu*nStaples+j];
17     dir2 = stapleDirs[mu*nStaples+j+1];
18     if(dir1==0 || dir2==0) break;
19     if(dir1>0){
20         dagger1 = false;
21         dir1 = dir1%24; // because direction 0 was indicated as 24
22         vertex = neighbor_plus[dir1*nsites+site];
23         aux_ind1 = (dir1*nsites+site)*Ncolsquare;
24     }
25     else{
26         dagger1 = true;
27         dir1 = -dir1%24;
28         vertex = neighbor_minus[dir1*nsites+site];
29         aux_ind1 = (dir1*nsites+vertex)*Ncolsquare;
30     }
31     if(dir2>0){
32         dagger2 = false;
33         dir2 = dir2%24;
34         aux_ind2 = (dir2*nsites+vertex)*Ncolsquare;
35 #ifdef __DebugStaple__
36         if(nextSite!=neighbor_plus[dir2*nsites+vertex]){
37             fprintf(stderr, "[ERROR]: pStaple does not close: site=%d, mu=%d
   , dir1=", site, mu);
38             if(dagger1) fprintf(stderr, "-");
39             fprintf(stderr, "%d, dir2=-%d\n", dir1, dir2);
40             fprintf(stderr, "                Link points to %d, staple ends in %d\n
   ", nextSite, neighbor_plus[dir2*nsites+vertex]);
41             exit(1);
42         }
43 #endif
44     }
45     else{
46         dagger2 = true;
47         dir2 = -dir2%24;
48         aux_ind2 = (dir2*nsites+nextSite)*Ncolsquare;
49 #ifdef __DebugStaple__

```

```

50     if(nextSite!=neighbor_minus[dir2*nsites+vertex]){
51         fprintf(stderr, "[ERROR]: pStaple does not close: site=%d, mu=%d
, dir1=", site, mu);
52         if(dagger1) fprintf(stderr, "-");
53         fprintf(stderr, "%d, dir2=-%d\n", dir1, dir2);
54         fprintf(stderr, "          Link points to %d, staple ends in %d\n
", nextSite, neighbor_minus[dir2*nsites+vertex]);
55         exit(1);
56     }
57 #endif
58     }
59     for(i=0; i<Ncolsquare; i++){
60         u1[i] = ufield[aux_ind1+i];
61         u2[i] = ufield[aux_ind2+i];
62     }
63
64     if(dagger1){
65         if(dagger2){ // u1^dag * u2^dag = (u2 * u1)^dag
66             mult_C_equals_AB(u3,u2,u1);
67             hermitian_conjugate(product,u3);
68         }
69         else{ // u1^dag * u2
70             mult_C_equals_AdaggerB(product,u1,u2);
71         }
72     }
73     else{
74         if(dagger2){ // u1 * u2^dag
75             mult_C_equals_ABdagger(product,u1,u2);
76         }
77         else{ // u1 * u2
78             mult_C_equals_AB(product,u1,u2);
79         }
80     }
81
82     for(i=0; i<Ncolsquare; i++) stot[i] += product[i];
83 }
84 }

```

File f4nstaple.cc:

```

1 void f4nstaple(dc *stot, int site, int mu, int first_direction, int
   use_smeared) {
2
3 // Computes the triangular staples from (site) to (site+a mu) in the
   negative
4 // directions (starting from first_direction) and adds them to stot.
5 //
6 // Important: As opposed to pstaple, here the result is ADDED to stot!
7
8 int i, j, nextSite, vertex, dir1, dir2, aux_ind1, aux_ind2;
9 bool dagger1, dagger2;
10
11 nextSite=neighbor_plus[mu*nsites+site];
12
13 for(j=0; j<nStaples; j+=2){
14     dir1 = stapleDirs[mu*nStaples+j];
15     dir2 = stapleDirs[mu*nStaples+j+1];
16     if(dir1==0 || dir2==0) break;
17     if(dir1>0){
18         dagger1 = false;
19         dir1 = dir1%24; // because direction 0 was indicated as 24
20         vertex = neighbor_plus[dir1*nsites+site];
21         aux_ind1 = (dir1*nsites+site)*Ncolsquare;
22     }
23     else{
24         dagger1 = true;
25         dir1 = -dir1%24;
26         vertex = neighbor_minus[dir1*nsites+site];
27         aux_ind1 = (dir1*nsites+vertex)*Ncolsquare;
28     }
29     if(dir2>0){
30         dagger2 = false;
31         dir2 = dir2%24;
32         aux_ind2 = (dir2*nsites+vertex)*Ncolsquare;
33 #ifdef __DebugStaple__
34         if(nextSite!=neighbor_plus[dir2*nsites+vertex]){
35             fprintf(stderr, "[ERROR]: nStaple does not close: site=%d, mu=%d
   , dir1=", site, mu);
36             if(dagger1) fprintf(stderr, "-");
37             fprintf(stderr, "%d, dir2=-%d\n", dir1, dir2);
38             fprintf(stderr, "                Link points to %d, staple ends in %d\n
   ", nextSite, neighbor_plus[dir2*nsites+vertex]);
39             exit(1);
40         }
41 #endif
42     }
43     else{
44         dagger2 = true;
45         dir2 = -dir2%24;
46         aux_ind2 = (dir2*nsites+nextSite)*Ncolsquare;
47 #ifdef __DebugStaple__
48         if(nextSite!=neighbor_minus[dir2*nsites+vertex]){
49             fprintf(stderr, "[ERROR]: nStaple does not close: site=%d, mu=%d
   , dir1=", site, mu);

```

```

50     if(dagger1) fprintf(stderr, "-");
51     fprintf(stderr, "%d, dir2=-%d\n", dir1, dir2);
52     fprintf(stderr, "          Link points to %d, staple ends in %d\n",
    nextSite, neighbor_minus[dir2*nsites+vertex]);
53     exit(1);
54 }
55 #endif
56 }
57 for(i=0; i<Ncolsquare; i++){
58     u1[i] = ufield[aux_ind1+i];
59     u2[i] = ufield[aux_ind2+i];
60 }
61
62 if(dagger1){
63     if(dagger2){ // u1^dag * u2^dag = (u2 * u1)^dag
64         mult_C_equals_AB(u3,u2,u1);
65         hermitian_conjugate(product,u3);
66     }
67     else{ // u1^dag * u2
68         mult_C_equals_AdaggerB(product,u1,u2);
69     }
70 }
71 else{
72     if(dagger2){ // u1 * u2^dag
73         mult_C_equals_ABdagger(product,u1,u2);
74     }
75     else{ // u1 * u2
76         mult_C_equals_AB(product,u1,u2);
77     }
78 }
79
80 for(i=0; i<Ncolsquare; i++) stot[i] += product[i];
81
82 }
83 }

```

File `plaquette.cc`:

```

1 double plaquette() {
2
3     int site, mu, i;
4     dc pltot=dc(0.,0.);
5
6
7
8
9
10
11    for (site=
12
13
14
15        0
16
17        ;site<
18
19
20
21        nsites
22
23        ;site++)
24    for (mu=0;mu<maxDir;mu++) {
25        for (i=0;i<Ncolsquare;i++) {
26            u4[i] = ufield[(mu*nsites+site)*Ncolsquare+i];
27        }
28    #ifdef __F4RootLattice__
29        f4pstaple(staple, site, mu, 0, 0);
30        f4nstaple(staple, site, mu, 0, 0);
31    #else
32        pstaple( staple, site, mu, 0, 0);
33    #endif
34    #ifdef __DebugUnitarity__
35        mult_C_equals_ABdagger(tempor, u4, u4);
36        fprintf(stdout, "[DEBUG]: Site=%d, mu=%d, uudag=(%f+i%f, %f+i%f, %f+
37            i%f, %f+i%f)\n",
38            site, mu, real(tempor[0]), imag(tempor[0]), real(tempor[1]),
39            imag(tempor[1]),
40            real(tempor[2]), imag(tempor[2]), real(tempor[3]), imag(
41            tempor[3]));
42    #endif
43    mult_C_equals_ABdagger(tempor, u4, staple);
44    for (i=0;i<Ncolsquare;i+=Ncol_plus_one) {
45        pltot+=tempor[i];
46    }
47
48
49
50
51
52
53
54
55
56
57
58 #elif defined __F4RootLattice__
59     return real(pltot)/(Ncol*nStaples*nlinks);
60 #else
61     return real(pltot)/(Ncol*dim*(dim-1.)*nsites);
62 #endif
63
64
65
66
67
68
69
70
71
72
73
74 }
```


Bibliography

- [1] Mark Srednicki. *Quantum Field Theory*. Cambridge: Cambridge University Press, 2007. ISBN: 9781139462761. URL: <https://books.google.com/books?id=50epxIG42B4C>.
- [2] M.E. Peskin and D.V. Schroeder. *An Introduction To Quantum Field Theory*. Frontiers in Physics. Avalon Publishing, 1995. ISBN: 9780813345437. URL: <https://books.google.it/books?id=EVeNNcslvX0C>.
- [3] S. Weinberg. *The Quantum Theory of Fields*. The Quantum Theory of Fields 3 Volume Hardback Set v. 2. Cambridge University Press, 1995. ISBN: 9780521550024. URL: <https://books.google.it/books?id=48xXMF1oHxkC>.
- [4] M. Kaku. *Quantum Field Theory: A Modern Introduction*. Oxford University Press, 1993. ISBN: 9780195091588. URL: <https://books.google.it/books?id=4m0McRFYnzQC>.
- [5] P. Ramond. *Field Theory: A Modern Primer*. Frontiers in Physics. Avalon Publishing, 1997. ISBN: 9780201304503. URL: https://books.google.it/books?id=EJ4%5C_BAAAQBAJ.
- [6] Dirk Schlingemann. “From Euclidean Field Theory to Quantum Field Theory”. In: *Reviews in Mathematical Physics* 11.09 (Oct. 1999), pp. 1151–1178. DOI: [10.1142/s0129055x99000362](https://doi.org/10.1142/s0129055x99000362). URL: <https://doi.org/10.1142/s0129055x99000362>.
- [7] I. Montvay and G. Münster. *Quantum fields on a lattice*. Cambridge Monographs on Mathematical Physics. Cambridge University Press, Mar. 1997. ISBN: 978-0-521-59917-7. DOI: [10.1017/CB09780511470783](https://doi.org/10.1017/CB09780511470783).
- [8] Christof Gatttringer and Christian B. Lang. *Quantum chromodynamics on the lattice*. Vol. 788. Berlin: Springer, 2010. ISBN: 978-3-642-01850-3. DOI: [10.1007/978-3-642-01850-3](https://doi.org/10.1007/978-3-642-01850-3).

- [9] Thomas DeGrand and Carleton E. Detar. *Lattice methods for quantum chromodynamics*. 2006.
- [10] H.S.M. Coxeter. *Regular Polytopes*. Dover Books on Mathematics. Dover Publications, 2012. ISBN: 9780486141589. URL: <https://books.google.it/books?id=2e7AQAAQBAJ>.
- [11] *The Art Of Computer Programming, Volume 2: Seminumerical Algorithms, 3/E*. v. 2. Pearson Education, 1998. ISBN: 9788177583359. URL: <https://books.google.it/books?id=0tLNKNVh1XoC>.
- [12] Martin Lüscher. “A portable high-quality random number generator for lattice field theory simulations”. In: *Computer physics communications* 79.1 (1994), pp. 100–110.
- [13] C. B. Lang and C. Rebbi. “Potential and Restoration of Rotational Symmetry in SU(2) Lattice Gauge Theory”. In: *Phys. Lett.* B115 (1982). [, 322 (1982)], p. 137. DOI: [10.1016/0370-2693\(82\)90813-9](https://doi.org/10.1016/0370-2693(82)90813-9).
- [14] J.F. Adams, Z. Mahmud, and M. Mimura. *Lectures on Exceptional Lie Groups*. Chicago Lectures in Mathematics. University of Chicago Press, 1996. ISBN: 978-0-22600-527-0. URL: <https://books.google.it/books?id=a3XCft6dubIC>.
- [15] Herbert Neuberger. “Spinless Fields on F_4 Lattices”. In: *Phys. Lett.* B199 (1987), p. 536. DOI: [10.1016/0370-2693\(87\)91623-6](https://doi.org/10.1016/0370-2693(87)91623-6).
- [16] G. Bhanot et al. “ ϕ^4 on F_4 : Analytical Results”. In: *Nucl. Phys.* B343 (1990), pp. 467–506. DOI: [10.1016/0550-3213\(90\)90479-W](https://doi.org/10.1016/0550-3213(90)90479-W).
- [17] Gyan Bhanot et al. “ ϕ^4 on F_4 : Numerical results”. In: *Nucl. Phys.* B353 (1991), pp. 551–564. DOI: [10.1016/0550-3213\(91\)90348-2](https://doi.org/10.1016/0550-3213(91)90348-2).
- [18] Urs M. Heller. “Higgs mass bound on an F(4) lattice”. In: *Nucl. Phys. B Proc. Suppl.* 20 (1991), pp. 609–612. DOI: [10.1016/0920-5632\(91\)90985-N](https://doi.org/10.1016/0920-5632(91)90985-N).
- [19] William Celmaster. “Gauge Theories on the Body - Centered Hypercubic Lattice”. In: *Phys. Rev.* D26 (1982), p. 2955. DOI: [10.1103/PhysRevD.26.2955](https://doi.org/10.1103/PhysRevD.26.2955).
- [20] William Celmaster. “Evidence for Improved Scaling on a Body Centered Hypercubic Lattice”. In: *Phys. Rev. Lett.* 52 (1984), p. 403. DOI: [10.1103/PhysRevLett.52.403](https://doi.org/10.1103/PhysRevLett.52.403).
- [21] William Celmaster. “The Average Plaquette of SU(2) Gauge Theory on a Body Centered Hypercubic Lattice”. In: *Phys. Rev.* D28 (1983), p. 2076. DOI: [10.1103/PhysRevD.28.2076](https://doi.org/10.1103/PhysRevD.28.2076).
- [22] W. Celmaster and K.J.M. Moriarty. “Monte Carlo simulation of pure SU(2) gauge theory on a body-centered hypercubic lattice”. In: *Computer Physics Communications* 34.4 (1985), pp. 415–425. ISSN: 0010-4655. DOI: [https://doi.org/10.1016/0010-4655\(85\)90069-4](https://doi.org/10.1016/0010-4655(85)90069-4). URL: <https://www.sciencedirect.com/science/article/pii/0010465585900694>.
- [23] A. König et al. “Correlation Length for SU(3) Gauge Theory on an 8×4 Lattice”. In: *Phys. Lett. B* 138 (1984), pp. 410–412. DOI: [10.1016/0370-2693\(84\)91929-4](https://doi.org/10.1016/0370-2693(84)91929-4).
- [24] Marco Panero. “Thermodynamics of the QCD plasma and the large-N limit”. In: *Phys. Rev. Lett.* 103 (2009), p. 232001. DOI: [10.1103/PhysRevLett.103.232001](https://doi.org/10.1103/PhysRevLett.103.232001). arXiv: [0907.3719](https://arxiv.org/abs/0907.3719) [hep-lat].

- [25] Anne Mykkänen, Marco Panero, and Kari Rummukainen. “Casimir scaling and renormalization of Polyakov loops in large-N gauge theories”. In: *JHEP* 1205 (2012), p. 069. DOI: [10.1007/JHEP05\(2012\)069](https://doi.org/10.1007/JHEP05(2012)069). arXiv: [1202.2762](https://arxiv.org/abs/1202.2762) [[hep-lat](#)].