Improved Actions in Lattice QCD

Frédéric D.R. Bonnet
Supervisors: Dr. Derek B. Leinweber and A/Prof. Anthony G. Williams.

May 4, 2008

Special Research Centre for the Subatomic Structure of Matter (CSSM) and Department of Physics and Mathematical Physics, University of Adelaide, 5005, Australia.

Thesis in Theoretical Physics submitted to fulfill the requirements of the the degree of Doctor of Philosophy.

Abstract

In this thesis I explore the physical effects of improved actions combined with improved operators in the framework of lattice QCD. All calculations are done in the quenched approximation, that is, when all of the dynamical fermion interactions have been suppressed by setting the determinant of the fermion matrix to a constant.

The thesis first briefly introduces lattice QCD to familiarize the reader with the basic concepts. It then describes the common numerical procedures used. It is made up of three major sections.

The first is the exploration of gauge field configurations and the study of the role of instantons in lattice QCD. In this work the Wilson gauge action and a standard 1 loop topological charge operator are used to determine the relative rates of standard cooling and smearing algorithms in pure $SU_c(3)$-color gauge theory. I consider representative gauge field configurations on $16^3 \times 32$ lattices at $\beta = 5.70$ and $24^3 \times 36$ lattices at $\beta = 6.00$. I find the relative rate of variation in the action and topological charge under various algorithms may be succinctly described in terms of simple formulae [1]. The results are in accord with recent suggestions from fat-link perturbation theory. This work is then extended to $\mathcal{O}(a^2)$-improved gauge action and $\mathcal{O}(a^2)$-improved operators [2]. In particular, an $\mathcal{O}(a^2)$-improved version of APE smearing is motivated by considerations of smeared link projection and cooling. The extent to which the established benefits of improved cooling carry over to improved smearing is critically examined. I consider representative gauge field configurations generated with an $\mathcal{O}(a^2)$-improved gauge field action on $16^3 \times 32$ lattices at $\beta = 4.38$ and $24^3 \times 36$ lattices at $\beta = 5.00$ having lattice spacings of 0.165(2) fm and 0.077(1) fm respectively. While the merits of improved algorithms are clearly displayed for the coarse lattice spacing, the fine lattice results put the various algorithms on a more equal footing and allow a quantitative calibration of the smoothing rates for the various algorithms. I find that the relative rate of variation in the action may also be described in terms of simple calibration formulae for $\mathcal{O}(a^2)$-improvement which accurately describes the relative smoothness of the gauge field configurations at a microscopic level.

In the second section the first calculation of the gluon propagator using an $\mathcal{O}(a^2)$-improved action with the corresponding $\mathcal{O}(a^2)$-improved Landau gauge fixing [3] condition is presented [4]. The gluon propagator obtained from the improved action and improved Landau gauge condition is compared with earlier unimproved results on similar physical lattice volumes of $3.2^3 \times 6.4$ fm. It is found that there is good agreement between the improved propagator calculated on a coarse lattice with lattice spacing $a = 0.35$ fm and the unimproved propagator calculated on a fine lattice with spacing $a = 0.10$ fm. This motivated us to calculate the gluon propagator on a coarse very large-volume lattice of $5.6^3 \times 11.2$ fm. The infrared behavior observed in previous studies is confirmed. The gluon propagator is enhanced at intermediate momenta and

[1] F. D. R. Bonnet, P. Fitzhenry, D. B. Leinweber, M. R. Stanford & A. G. Williams, *Phys. Rev. D* **62**, 094509 (2000) [hep-lat/0001018].

[2] F. D. R. Bonnet, D. B. Leinweber, A. G. Williams & J. M. Zanotti, *Submitted to Phys. Rev. D.* [hep-lat/0106023].

[3] F. D. R. Bonnet, P. O. Bowman, D. B. Leinweber, D. G. Richards & A. G. Williams, *Aust. J. Phys.* **52**, 939 (1999).

[4] F. D. R. Bonnet, P. O. Bowman, D. B. Leinweber & A. G. Williams, *Infrared behavior of the gluon propagator on a large volume lattice, Phys. Rev. D* **62**, 051501, (2000).

suppressed at infrared momenta. The observed infrared suppression of the Landau gauge gluon propagator is not a finite volume effect. This work is then extended to a variety of lattices with spacing ranging from $a = 0.17$ to $a = 0.4$ fm [5] to further explore finite volume and discretization effects. In this work a technique previously used for minimizing lattice artifacts, known as "tree-level correction", has also been extended. It is demonstrated that by using tree-level correction, determined by the tree-level behavior of the action being considered, it is possible to obtain scaling behavior over a very wide range of momenta and lattice spacings. This makes it possible to explore the infinite volume and continuum limits of the Landau-gauge gluon propagator.

As a final part of this thesis I present the first results for the quark propagator using an Overlap fermionic quark action [6]. I compare the results with those obtained from the standard Wilson fermion. The overlap quark action is $\mathcal{O}(a)$-improved compared with the Wilson fermion. This action realizes exact chiral symmetry on the lattice unlike the Wilson fermion and it demonstrates that the fastest way forward in this field is with improved lattice operators.

The idea of studying improved actions in lattice gauge theory was suggested to me by A/Prof. Anthony G. Williams during the *"Nonperturbative Methods in Quantum Field Theory"* workshop in early February 1998. Initially it was suggested to me that a calculation of the gluon propagator using improved action on large volumes, following a study just done with standard gauge action in Ref. [62]. The point of interest was to study the effect an improved gauge field action would have on the gluon propagator. This study would then be extended to quark actions. In the meantime when generating gauge field configurations using a computer code written in Fortran 77 (provided by Dr. Derek B. Leinweber), it occurred to me that it would be good to explore the content of these gauge field configurations. In order to do realistic calculations on large lattices we needed a gauge field configuration generator that would run on our CM5 computer and so Connection Machine Fortran (CMF) became the adopted language.

I started writing the computer code to generate the gauge field configuration in the $SU_c(2)$ with the help of Dr. Derek B. Leinweber, who introduced me to the basic concepts in lattice QCD. I then extended this code to the $SU_c(3)$ gauge group. This is commonly known as the standard Wilson gauge action. After investigating with some of the optimization possibilities, I moved on to code an $\mathcal{O}(a^2)$-improved gauge action. The code uses a masking procedure for the link update. I have generalized the masking procedure for any planar gauge field action in $SU_c(N)$, Ref. [18].

From there it was very obvious that by applying a continuous repetition of some sections of code that I written, that some bigger Wilson loops could easily be included in the action and hence highly improved actions could be easily constructed. The only difficulty was to calculate the improvement coefficients.

I then moved on to study smearing algorithms. I adapted the gauge field configuration code to a cooling and a 1×2 and 2×1 improved cooling code in which we inserted higher order loop operators. This was the tool used to explore gauge field configurations and their topological structures. Once the short range quantum fluctuations are removed it is possible to see instantons. Instantons are believed to play a crucial role in the spontaneous chiral symmetry breaking mechanism. We improved

[5]F. D. R. Bonnet, P. O. Bowman, D. B. Leinweber, A. G. Williams & J. M. Zanotti, *Infinite volume and continuum limits of the landau gauge gluon propagator*, *Phys. Rev. D* **64**, 034501 (2001) [hep-lat/0101013].

[6]F. D. R. Bonnet, P. O. Bowman, D. B. Leinweber, A. G. Williams & J. Zhang, *Overlap Propagator in Landau Gauge*, *to be Submitted to Phys. Rev. D.*

the topological charge operator from the clover term to an (1×2 and 2×1) $\mathcal{O}(a^2)$–improved topological charge operator (see Appendices, Sections E.16 and E.17). This code was subsequently adapted by Sundance Bilson-Thompson so that he could insert higher order loops. I have also inserted my $\mathcal{O}(a^2)$–improved operator to construct an $\mathcal{O}(a^2)$–improved smearing algorithm. Using these tools I have calibrated the relative rates of cooling and smearing.

Another piece of work on gauge fixing, reviewed in Chapter 8, was led by Dr. Patrick O. Bowman, Ref. [63]. There I supplied the gauge field configurations and checked some of the analytical work. For the gluon propagator work I supplied all of the lattice configurations with the exception of the $32^3 \times 64$ used in Ref. [62]. The analysis was primarily carried out by Dr. Patrick O. Bowman and partly inspired by the one carried out in hep-lat/0106023. While this gluon propagator work is not being presented here as my own Ph. D. qualifying work, I am a co–author on the subsequent papers and so I have therefore decided to include a review of this work in Chapter 9.

I have also made some contribution in the construction of the Fat–link quark action (with and without the clover term) developed by James M. Zanotti. These contributions involve the code for the Reunitarization of the smeared links, Appendix E.21. Because of the code developed for the improved lattice definition of the $F_{\mu\nu}(x)$ term I have also made some contribution to the Fat–link clover quark action although I will not discuss about this work in the following thesis.

My main contribution for the overlap quark propagator study was in the analysis of the propagator data. The overlap propagators were generated by Dr. Jianbo Zhang and the research was also carried out in collaboration with A/Prof. Anthony G. Williams and Dr. Derek B. Leinweber. The quark propagators for the Wilson fermion were generated by a computer code parallelized by James M. Zanotti and originally written by Prof. Frank X. Lee.

The anisotropic lattice code has not been used in any calculations yet although it has been tested and verified. The code was extended from the isotropic improved generator code in $SU_c(3)$. After a literature search, we decided to implement the action described in Ref. [31] for the anisotropic Wilson action and in Ref. [11, 32] for the improved anisotropic case.

Apart from the work on the gauge fixing and the gluon propagator, done in collaboration with Dr. Patrick O. Bowman, and which for completeness is briefly reviewed in Chapters 8 and 9 respectively, this thesis contains no material which has been accepted for the award of any other degree or diploma in any university or other institution and to the best of knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text.

I give consent to this copy of my thesis, when deposited in the University Library, being available for loan and photocopying.

Frédéric D. R. Bonnet　　　　Date: 20th of September 2001.

Acknowledgement

I would like to thank my supervisors Assoc. Prof. Anthony G. Williams and Dr. Derek B. Leinweber for their help and their guidance throughout my studies. I would like to give special thanks to Assoc. Prof. Anthony G. Williams for encouraging me to take a project in Lattice Gauge Theory and for his clear vision which lead to this thesis. I would also like to thank Dr. Derek B. Leinweber for helping me with most of the aspects of this thesis.

I would like to gratefully acknowledge Dr. Patrick O. Bowman for analysing all of the gauge field configurations in the study of the gluon propagator, James M. Zanotti for calculating the various lattice spacings and allowing me to use the quark propagator code, Dr. Jianbo Zhang for generating the overlap quark propagators, Francis Vaughan and Paul Coddington of the South Australian Centre for Parallel Computing and the Distributed High-Performance Computing Group for support in some of the technical problems and for keeping the CM5 running, and to Ramona Adorjan for her hard and dedicated work on daily computer maintenance.

Special thanks to Prof. Anthony W. Thomas and Assoc. Prof. Anthony G. Williams for working so hard to keep the CSSM running as a research centre and providing a stimulating and pleasant research environment.

I would also like to thank Olivier Leitner and Francois Bissey for their friendship and for sharing some of their research interests with me.

Finally I acknowledge all of the people at the CSSM for providing such a good environment for studying physics. I think that Adelaide University is extremely fortunate to have such a world–class research centre as the CSSM.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Standard model (SM) of particle interactions is a synthesis of three of the four forces of nature. These forces are described by gauge theories each of which is characterized by a coupling constant, g. For the strong interaction, $g_s \sim 1$. For the electromagnetic interaction the fine structure constant is $g_{\mathrm{em}} \sim 1/137$. This constant is known to a very high precision, the predictions of Quantum Electrodynamics (QED) agree with nature to extremely high accuracy making it the most successful theory in physics. The smallness of g is the reason why the perturbative approach has been very successful for QED. For the weak interactions $G_{\mathrm{F}} \sim 10^{-15}\,\mathrm{GeV}^2$.

It is almost universally accepted that Quantum Chromodynamics (QCD) is the underlying quantum field theory of the strong interaction [1, 2] which binds atomic nuclei and fuels the sun and the stars. Strongly interacting particles are referred to as hadrons which include for example, the protons and neutrons that make up atomic nuclei as well as a wide variety of particles that are produced in particle accelerators and from astrophysical sources. In Fig. 1.1, I show a naive picture of the proton, surrounded by the QCD vacuum. These hadrons are made up of quarks and gluons

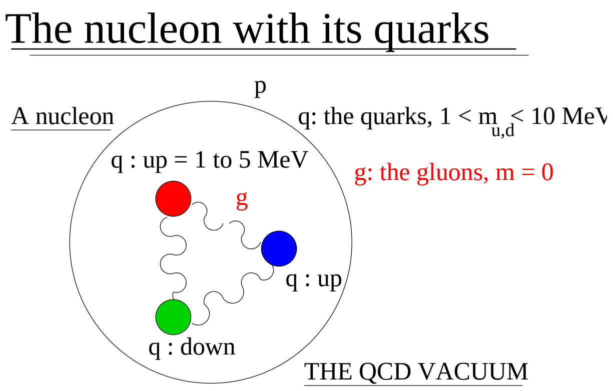


Figure 1.1: A naive diagram of the nucleon with its constituent quarks

which are the underlying constituents in QCD. Experimental work has confirmed that the basic constituents of matter are the six quarks: up, down, strange, charm, bottom and top (u,d,s,c,b and t respectively). The up and down quarks are light and have about the same mass, while the bottom and top quarks are much more massive, as it can be seen in Fig. 1.2. Each of these quarks posses an internal degree of freedom called colour. The six leptons, Fig. 1.3, are the remaining basic constituents of matter.

The concept of colour was first introduced in 1963 by Gell–Man and Zweig who proposed a model that explained the spectrum of strongly interacting particles in terms

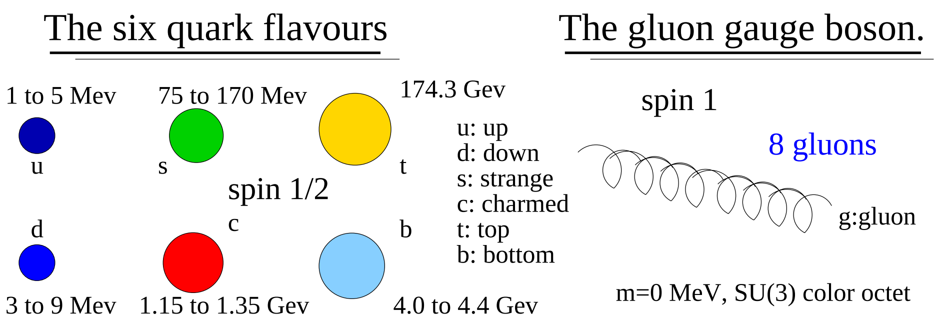


Figure 1.2: The six quarks all with spin 1/2 with their respective masses. The gluon has spin 1, it is a vector boson. The gluons are the mediator of the strong interaction.

of basic constituents called quarks. In this model the mesons were expected to be made up of a quark and an anti–quark while the baryons were understood to be bound states

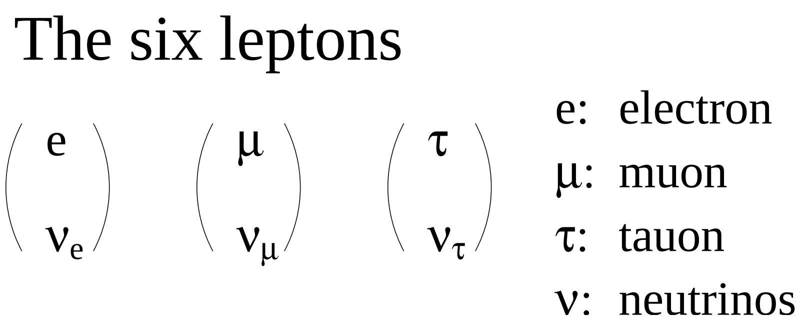


Figure 1.3: The six leptons.

made up of three quarks. To explain the electric charges and other quantum numbers Gell–Man and Zweig postulated the existence of three distinct species (called flavours) of quarks: the up (u), down (d) and strange (s) quarks. The discovery of additional hadrons confirmed the existence of the three other flavours: charm (c), bottom (b) and top (t). To respect the baryon symmetry, the quarks needed to be assigned fractional electric charges of +2/3 for the u, c, t and -1/3 for the d, s, b. Then the proton would be a bound state of uud, while the neutron a bound state of udd.

Phenomenologically the model was very successful. However, it had some serious problems. Firstly the problem of confinement. Free particles with fractional charges could not be found despite tremendous efforts. Secondly some of the lightest excited states of the nucleon, like the spin 3/2 with charge +2, the Δ^{++}, required a totally symmetric wave function under interchange of the quark spin and flavour quantum numbers. This contradicted the expectation that quarks, which must have spin 1/2, should obey Fermi–Dirac statistics.

The later problem was reconciled by Han and Nambu, Greenberg and Gell–Man, who proposed that the quarks carried an additional internal degree of freedom called "colour". This quantum number is fully represented by the non–Abelian gauge group, $SU_c(3)$. This gauge group possesses 8 generators, each of which may be viewed as a quanta of the $SU_c(3)$ gauge field. These quanta are called gluons.

The quarks are spin-1/2 particles (i.e., fermions) and the gluons are massless spin-1 particles (i.e., gauge bosons). The quarks interact strongly through their "colour" charge through the exchange of gluons. The 8 gluons of $SU_c(3)$ (i.e., one for each generator of $SU_c(3)$) themselves carry colour.

This theory of the strong interactions, Quantum Chromodynamics, can be fully defined in terms of quark and gluon fields, through a fundamental quantity of quantum field theory referred to as the action. The action is the integral of the Lagrangian density which is a function of the quark and gluon fields. This action in Minkowski space-time (up to gauge–fixing terms) is defined as:

$$\begin{aligned} S_{\mathrm{QCD}} &= \int d^4x\mathcal{L} \\ &= \int d^4x \left(-\frac{1}{4} F^a_{\mu\nu}(x) F^{a\mu\nu}(x) \sum_q \bar{\psi}^i_q(x)(i\gamma_\mu D_\mu(x) - m_q)_{ij} \psi^j_q(x) \right), \quad (1.1) \end{aligned}$$

with the non–Abelian electromagnetic and covariant derivative expressed as:

$$\begin{aligned} F^a_{\mu\nu}(x) &= \partial_\mu A^a_\nu(x) - \partial_\nu A^a_\mu(x) + g_s f_{abc} A^b_\mu(x) A^c_\nu(x), && (1.2) \\ (D_\mu(x))_{ij} &= \delta_{ij}\partial_\mu - ig_s \sum_a \frac{\lambda^a_{ij}}{2} A^a_\mu(x). && (1.3) \end{aligned}$$

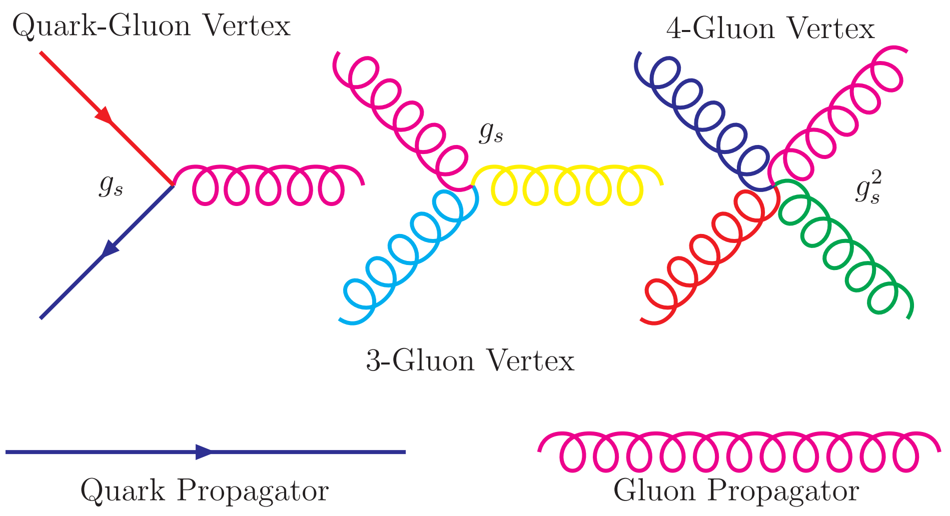
The indices, ij, are the spinor indices. The first term describes the dynamics of gluons while the second term involves both the quark and gluon fields and is summed over the number of quark flavours. In Eq. (1.1), g_s is the QCD coupling constant. There is only one coupling strength between all the quarks and gluons. The fully antisymmetric tensor, f_{abc}, is the structure constant of the colour octet $SU_c(3)$ gauge group (The Gell–Man matrices and the values for the f_{abc} can be found in Appendix A).

The $\psi^j_q(x)$ are the 4-component Dirac spinors associated with each quark field of 3 colours and the flavour q. The eight gluons fields $A^a_\mu(x)$ are contained in the colour octet $SU_c(3)$ Lie Algebra.

Expanding the terms of Lagrangian density one finds

$$\begin{aligned} \mathcal{L} &= -\frac{1}{4}\left(\partial_\mu A^a_\nu(x) - \partial_\nu A^a_\mu(x)\right)^2 \\ &- \frac{1}{2}\left[g_s f_{abc}\left(\partial_\mu A^a_\nu(x) - \partial_\nu A^a_\mu(x)\right) A^b_\mu(x) A^c_\nu(x) + \frac{1}{2}\left(g_s f_{abc} A^b_\mu(x) A^c_\nu(x)\right)^2 \right] \\ &+ \sum_q \bar{\psi}^i_q(x)(i\gamma_\mu\partial_\mu(x) - m_q)_{ij}\psi^j_q(x) + \bar{\psi}^i_q(x)\left[\gamma_\mu\left(g_s\sum_a \frac{\lambda^a_{ij}}{2} A^a_\mu(x)\right)\right]\psi^j_q(x). \quad (1.4) \end{aligned}$$

From this expansion we can see that we obtain terms involving different powers in the coupling. The terms at the zeroth order will give the gluon and quark propagators while at the first order we obtain the 3–gluon vertex for the term just involving the gluon field and the quark–gluon vertex when both the quark and gluon fields are mixed. At the second order we obtain the 4–gluon vertex. These green functions are drawn in the following Feynman diagrams:



Hence the gluons interact with themselves as well as with the quarks. It is these quantities that give rise to the non–perturbative physics and it is the essential difference between QCD and the corresponding theory of photons and electrons referred to as quantum electrodynamics (QED). The difference between these two has far reaching consequences since the theories have entirely different behaviour. One of them is that at shorter distances the effective coupling constant in QCD decreases. The property is known as asymptotic freedom.

There are many different ways to probe the dynamics of these particles. One way, which is an important theme in this thesis, is the study of quark and gluon propagators. For example for the gluon propagator there has been considerable interest in the infrared behaviour of the propagator as a probe into the mechanism of confinement [3, 4] and as input for other calculations. Another fundamental quantity of QCD is the quark propagator, Fig. 1.4.

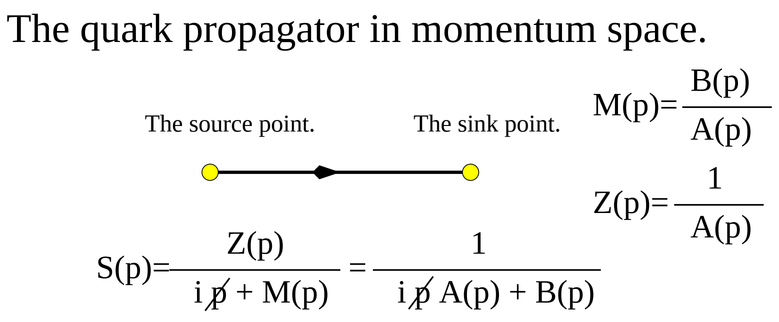

The quark propagator in momentum space.
The source point.
The sink point.
$M(p)=\frac{B(p)}{A(p)}$
$Z(p)=\frac{1}{A(p)}$
$S(p)=\frac{Z(p)}{i\not{p}+M(p)}=\frac{1}{i\not{p}A(p)+B(p)}$

Figure 1.4: Illustration of the Feynman graph for the quark propagator. The $M(p)$ is the quark mass function and $Z(p)$ is the quark renormalization momentum function. The propagator is created at a source point propagated along to a sink point where it is annihilated.

From the quark propagator it is possible to extract the quark mass function, $M(p)$, and the quark renormalization momentum function, $Z(p)$. This quantity is really a description of how the quark propagates in the QCD vacuum. By studying the momentum dependent quark mass function in the infrared region (the scalar part of the propagator) we can gain some insights into the mechanism of chiral symmetry breaking. Chiral symmetry is dynamically broken in the QCD vacuum. This gives rise to mass generation in the infrared. The sum of the current quark mass makes up less than 3%

of the nucleon mass, the rest of the mass comes from dynamical symmetry breaking and it is responsible for about 97% of the mass inside the protons and neutrons. So studying the quark propagator will help us to understand the mechanism that gives rise to the majority of the mass of strong interacting particles.

There are very few first-principles methods for studying QCD in the nonperturbative low-energy regime. Of these few, the most widely used is the so-called Lagrangian-based lattice field theory which formulates the field theory on a Euclidean space-time lattice [5, 6]. An alternative lattice approach is based on the Hamiltonian formulation of quantum field theory and makes use of cluster decompositions and again Monte Carlo methods to carry out the simulations [7]. In addition, there are numerous studies based on a light-front formulation of QCD [8] and much use has been made of Schwinger-Dyson equations [4] to assist with the construction of QCD-based quark models.

The Lagrangian-based lattice technique simulates the functional integral using a four-dimensional hypercubic Euclidean spacetime lattice together with Monte Carlo methods for generating an ensemble of gluon field configurations with the appropriate Boltzmann distribution $\exp(-S_G)$. The argument of the exponential, S_G is a discretized form of the QCD gluon action on the hypercubic lattice. The simplest discretizations of the QCD action involve only nearest neighbours on the lattice and have $\mathcal{O}(a^2)$ errors, where a is the lattice spacing. Improved actions represent a major advance for the field of lattice gauge theory, where by using increasingly non-local discretizations of the QCD action we can obtain the same accuracy with far fewer lattice points and hence far less computational time and effort. The purpose of this work is to use improved actions and improved operators to study observables close to their continuum limits. For further details on the state of the art lattice QCD techniques see for example Ref. [9]. Another related and equally important advance is the technique of nonperturbative improvement (e.g., mean-field improvement) which corrects for some of the major nonperturbative effects (the so-called tadpole contributions) and hence brings the lattice results to their continuum form more quickly by improving the match with perturbation theory at a given lattice spacing a [10]. It is the combination of improved actions and nonperturbative improvement that together have come to represent a significant advance for the field [9, 11].

Lattice QCD is based on a Monte Carlo treatment of the path integral formulation which makes it a computationally demanding method for calculating physical observables. The gluon field is represented by 3×3 complex $SU(3)$ matrices, where there is one such $SU(3)$ matrix associated with every link on the lattice. The links lie only along one of the four Cartesian directions and join neighbouring lattice sites. Since all lattice links require identical numerical calculations, lattice gauge theory is ideally suited for parallel computers.

There are various types of improved actions, and as explained above, these are all based on the idea of eliminating the discretization errors that occur when passing from continuum physics to the discretized lattice version. The simplest (i.e., non-improved) gluon action is the so called *standard Wilson* action and consists of 1×1 Wilson loops or, as they are frequently called, plaquettes. The Wilson loops used to build up lattice actions shall often be referred to, in this thesis, as plaquettes. The need to build the gluon action out of closed loops arises from the need to maintain exact $SU(3)$ gauge invariance in the discrete lattice action and from the fact that closed loops are gauge invariant. This 1×1 loop action was first proposed by Wilson [12] in the early 70's and has been used extensively over the years. It consists of taking an arbitrary starting

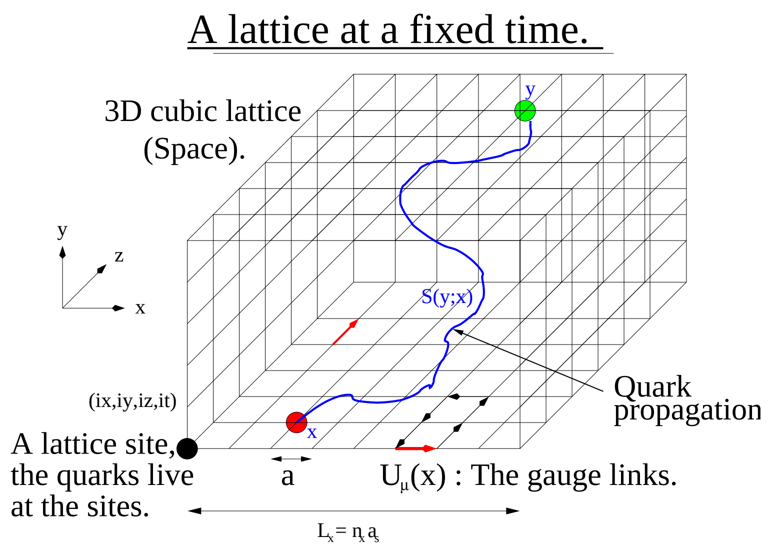

A lattice at a fixed time.
3D cubic lattice (Space).
y
z
x
y
S(y;x)
Quark propagation
(ix,iy,iz,it)
A lattice site, the quarks live at the sites.
x
a
U_μ(x) : The gauge links.
L_x = n_x a_s

Figure 1.5: A three dimensional lattice at a fixed time in Euclidean space, i.e., a single time "slice". The quarks live at the sites while the gluon fields reside on the gauge link $U_\mu(x)$. Each sites are separated by a space of dimension a. The spacing a is called the lattice spacing and can be tuned for a given simulation.

site, for example x, on the lattice and stepping around a 1×1 loop until returning to the starting point x. The 1×1 Wilson loop is illustrated in Fig. 1.6.

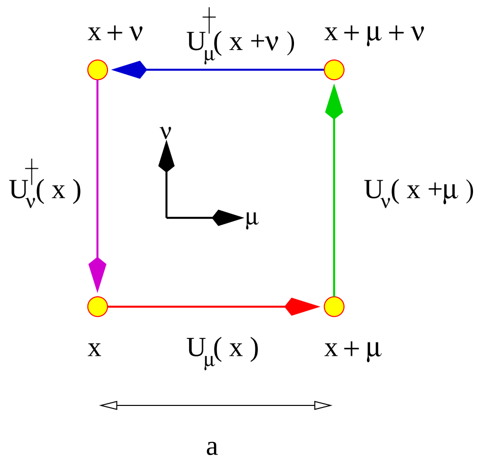


Figure 1.6: The 1×1 plaquette $U_{\rm sq}(x)$ with base at x lying in the $\mu\nu$-plane. The lattice spacing is denoted by a.

Improving the standard Wilson action is achieved by making use of larger loops (e.g., 1×2, 2×2, etc.) in the lattice gluon action [13] to cancel out finite lattice spacing artifacts to a given order in a. For an elegant and detailed discussion of these topics see Ref. [11].

As already mentioned, a central theme of this thesis is the study of the gluon and quark propagators. An effective study of the gluon propagator is done by improving the gluonic action to higher order in the lattice spacing. Similarly for the quark propagator, where a suitably improved choice of the fermion is crucial in order to extract meaningful results over the entire momentum range. In the deep infrared region, artifacts associated with the finite size of the lattice spacing become small for sufficiently small lattice spacing a. This is the most interesting region as nonperturbative physics lies here. However, the ultraviolet behaviour at large momenta of the propagator will in general strongly deviate from the correct continuum behaviour. This behaviour will be action dependent, and it is this region that really dictates the quality of the discretised action. Some interesting progress has been made in improving the ultraviolet behaviour of the propagator for certain quark actions, a method recently developed and referred to as tree-level correction.

This thesis is divided as follows:

Chapter 2 introduces the basic elements used in lattice QCD. The concept of the link variable represents the fundamental quantity of the theory. I then describe, in Chapter 3.1 the general procedure used in conducting numerical simulations. Chapter 4 goes through the algorithm that enables us to generate non–Abelian gauge field configurations as well as the technical related to the implementation of the computer code.

Chapter 5 describes the $\mathcal{O}(a^2)$–improved gauge action. In this chapter a general masking method is presented. This method is applicable to planar lattice gauge action in $SU_c(N)$ of any size. Chapter 6 briefly describes the anisotropic gauge action for two different types of improved gauge action. Chapter 7 uses a cooling method with both

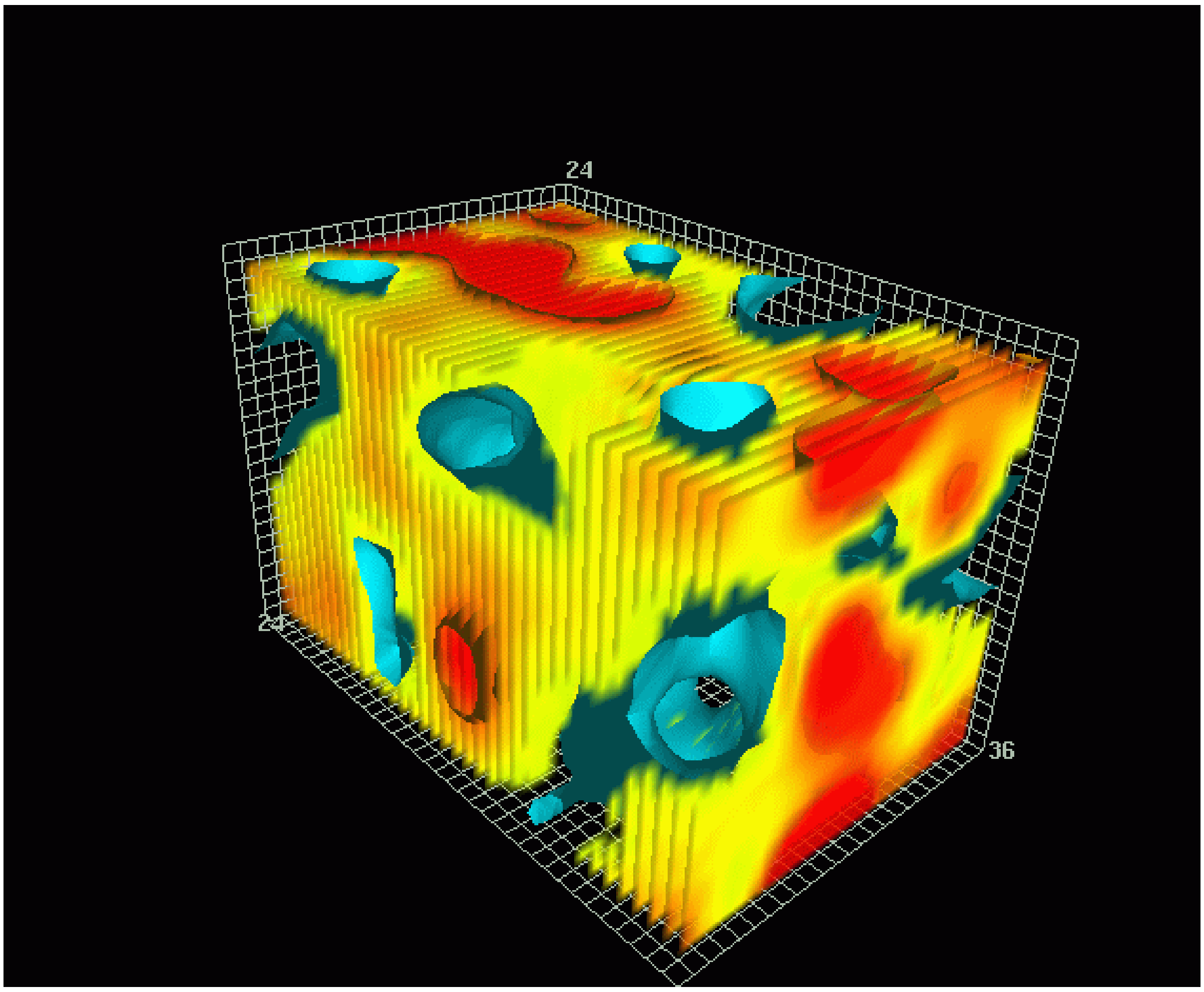


Figure 1.7: Graphical representation of the action density after 11 sweeps of improved cooling on a $24^3 \times 36$ lattice at $\beta = 5.00$

standard and improved operators to bring some insight into the various smoothing methods. In this chapter, I also construct an improved topological charge operator based on the improved definition of the field strength tensor. Using various smoothing algorithms I calibrate the relative smoothing rates between the different algorithms, using improved and non–improved operators. The goal is to study the distribution of action and topological charge density of typical smoothed gauge field configurations. This will allow us to probe the importance of topology and topological structures, such as instantons, for understanding the properties of QCD. This is done with the help of a graphical representation like the one shown in Fig. 1.7.

Chapter 8 introduces improved gauge fixing methods. Chapter 9 explores the gluon propagator in quenched lattice QCD with improved actions at coarse lattice spacing and in the infinite volume limit. Chapter 10 presents work on the quark propagator for various actions, including the overlap Wilson fermion.

Chapter 2

Lattice Gauge Theory

In the introduction I have briefly introduced the basic concept of a discretized gluonic action. In this chapter I shall discuss some of the fundamentals of lattice QCD. These concepts are of crucial importance, since the forthcoming chapters become more technical and revolve around the basic definitions that are explained in this chapter.

Lattice QCD is the best tool we have at the moment for calculating non–perturbative observables from first–principles. This field is rapidly expanding, which is strongly correlated with the fact that computer resources are rapidly growing. The basic idea is to replace infinite four dimensional continuous space–time by a discrete, finite volume one. The scale of the four dimensional volume is characterized by a parameter called the lattice spacing. The lattice spacing, a, plays the role of a cut–off parameter and, therefore, theories on the lattice do not contain ultraviolet divergences, i.e., they are regularized. On the other hand, when passing to a lattice theory, the relativistic invariance is violated, but the gauge invariance is preserved. In the continuum limit ($a \longrightarrow 0$) relativistic invariance is restored and in a renormalizable theory like QCD, the renormalized quantities approach finite limits.

2.1 QCD on the Lattice

2.1.1 The Basic Elements of Lattice QCD

For technical reason we pass from Minkowski space–time to Euclidean space–time, that is make the change of variable $x_0 \longrightarrow it$. The theory is constructed from few basic elements of the lattice which are the following:

- A *lattice site*, or a point of the lattice is specified by the coordinates $x_\mu = an_\mu$, where n_μ is a 4–vector with components n_x, n_y, n_z and n_t. In general the number of sites in the spacial directions are set to be equal and in the temporal direction a multiple of the spacial direction. The parameter a is the lattice spacing, i.e., the distance between neighbouring lattice sites. On isotropic lattices the spacing is the same in all directions. In the case of anisotropic lattices the spacing in the spatial and temporal directions are different.

  In order to study QCD one needs to attach an $SU_c(3)$ matrix to each lattice link. The generators of this gauge group are explicitly written in Appendix A.

- A *link variable*, $U_\mu(x)$, or a link connecting two neighbouring sites, is given by the coordinate of the x_μ of its origin and by a direction of the corresponding axis

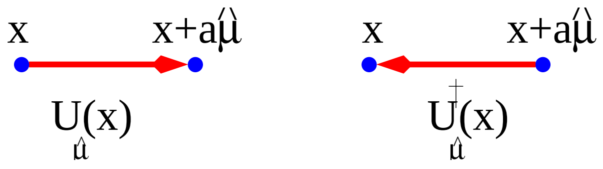


Figure 2.1: The link variable joining two lattice sites in the $\mu\nu$-plane. The lattice spacing is denoted by a.

in the space; the link variable connects lattice sites with the coordinates x and $x + a\hat{\mu}$, where $\hat{\mu}$ is a unit vector in the direction of μ. When the direction of the link variable is reversed we take the hermitian conjugate of $U_\mu(x)$, as shown in Fig. 2.1.

The gluon field lives on the gauge links which can be related to the continuum gluon field by

$$U_\mu(x) = \mathcal{P}e^{ig_0 \int_x^{x+\hat{\mu}} A_\mu(z)dz}\,. \tag{2.1}$$

The gluon field $A_\mu(x)$ is contained in the colour octet $SU_c(3)$ Lie Algebra, i.e. $A_\mu(x) = \sum_{a=1}^{8} t^a A_\mu^a(x)$ with $t^a = \lambda^a/2$ and λ^a are the Gell–Mann matrices, in the case of $SU_c(2)$ the generators are just the Pauli matrices σ^a, See Appendix A.

- A *plaquette*, or an elementary square bounded by four links, illustrated in Fig. 2.2 is given by the coordinate of the site lying in its corner and by the positive

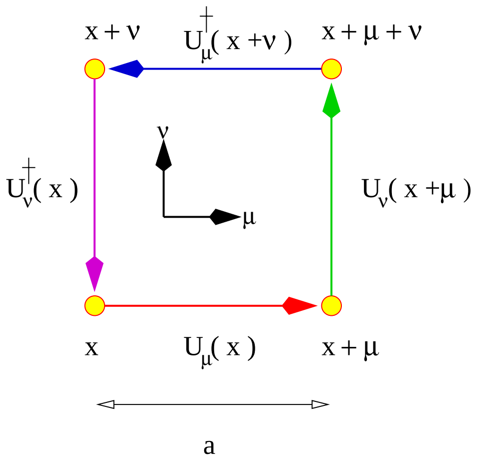


Figure 2.2: The 1×1 plaquette $U_{\rm sq}(x)$ with base at x lying in the $\mu\nu$-plane. The lattice spacing is denoted by a.

directions μ and ν of the adjacent links forming the sites of the square. The plaquette variable is written as:

$$P_{\mu\nu}(x) = U_\mu(x)\, U_\nu(x + \hat{\mu})\, U_\mu^\dagger(x + \hat{\nu})\, U_\nu^\dagger(x)\,. \tag{2.2}$$

A plaquette is the simplest closed Wilson loop that can be formulated on the lattice. More complicated closed loops may be constructed quite easily by attaching more links together.

- A *staple* is formed by three *links* joined edge–to–edge in one plane. In Fig. 2.3, I illustrate a *staple* in a given plane (right), and on the left of that figure I illustrate the sum of the six staples. The link joining the lattice x and $x + \hat{x}$ is pointing in the $\hat{x}$ direction.

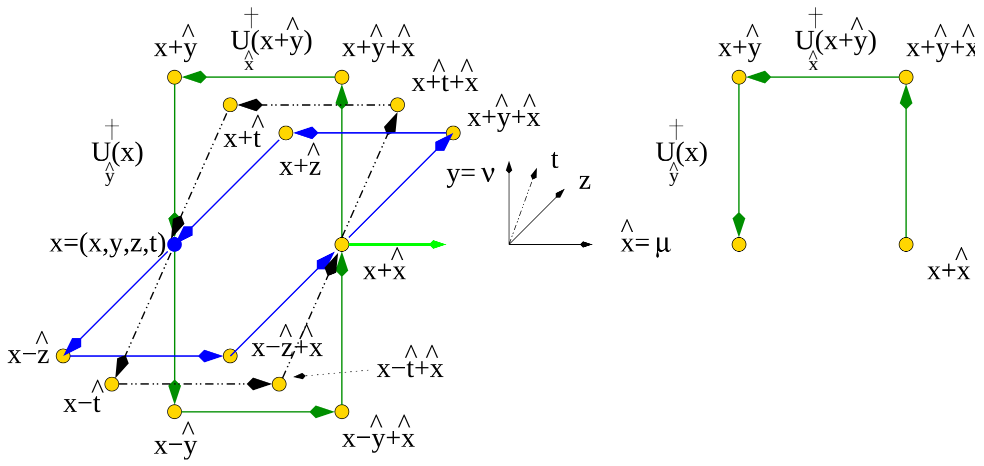


Figure 2.3: Rotating the 1×1 plaquette sitting in the $\hat{x}\hat{y}$ plane about the $\hat{x}$–axis into the $\hat{x}\hat{z}$ and $\hat{x}\hat{t}$ planes (left) and a staple (right).

2.1.2 The General Method in Lattice QCD

Lattice QCD calculations are a discretized non–perturbative implementation of field theory using the Feynman path integral, which in a quantum field theory should more appropriately be referred to as the Feynman functional integral. It is used when one is confronted in computing the expectation value of an observable or some n–point Greens function. The **Feynman path integral** definition of this is

$$\mathcal{G}(x_1, .., x_n) = \langle 0|T(\Phi(x_1)..\Phi(x_n))|0\rangle = \frac{\int \mathcal{D}\Phi(\prod_j^n \Phi(x_j))e^{iS_{\rm QCD}}}{\int \mathcal{D}\Phi e^{iS_{\rm QCD}}}. \tag{2.3}$$

In QCD, $\Phi(x)$ become the quark and gluon fields. The exponential factor $e^{iS_{\rm QCD}}$ acts as a probability, but is imaginary. So strong oscillations for large $S_{\rm QCD}$ makes the path integral numerically intractable in ordinary Minkowski space, however, numerical simulation is possible in Euclidean space. The transition from Minkowski to Euclidean space is done by transforming the time component of the fields to an imaginary one. So the n–point green function defined in Eq. (2.3) becomes

$$\langle 0|T(\Phi(x_1)..\Phi(x_n))|0\rangle \overset{t\to -it_{\rm E}}{\longmapsto} \langle 0|\Phi(x_1)..\Phi(x_n)|0\rangle_{\rm E}\,. \tag{2.4}$$

Such transformations transform the exponential factor from an imaginary to a real factor,

$$S_{\rm QCD} \longrightarrow iS^{\rm E}_{\rm QCD} \quad \text{this means} \quad e^{iS_{\rm QCD}} \longrightarrow e^{-S^{\rm E}_{\rm QCD}}. \tag{2.5}$$

Strong oscillations are then damped out at large $S^{\rm E}_{\rm QCD}$, making Monte Carlo simulations possible. So in Euclidean space our n–point green function becomes:

$$\begin{aligned}\mathcal{G}_{\rm E}(x_1, .., x_n) = \langle 0|\Phi(x_1)..\Phi(x_n)|0\rangle_{\rm E} &= \frac{\int \mathcal{D}\Phi(\prod_j^n \Phi(x_j))e^{-S^{\rm E}_{\rm QCD}}}{\int \mathcal{D}\Phi e^{-S^{\rm E}_{\rm QCD}}} \\ &= \frac{1}{\mathcal{Z}}\int \mathcal{D}\Phi(\prod_j^n \Phi(x_j))e^{-S^{\rm E}_{\rm QCD}}.\end{aligned} \tag{2.6}$$

Let's rewrite the partition function as:

$$\mathcal{Z} = \int \mathcal{D}A_\mu \mathcal{D}\psi\mathcal{D}\overline{\psi}\exp\left(-S_{\rm QCD}\right), \tag{2.7}$$

where $S_{\rm QCD}$ is the QCD action in Euclidean space–time (I have dropped the superscript on the action term, as from now all the quantities will be in Euclidean space–time unless stated otherwise)

$$
\begin{aligned}
S_{\rm QCD} = \int d^4x \mathcal{L} \; &= \; \int d^4x \left(\frac{1}{4} F^a_{\mu\nu}(x) F^{a\mu\nu}(x) + \sum_q \bar{\psi}^i_q(x) (\gamma_\mu D_\mu(x) + m_q)_{ij} \psi^j_q(x) \right) \\
&= \; \int d^4x \left(\frac{1}{4} F^a_{\mu\nu}(x) F^{a\mu\nu}(x) + \sum_q \bar{\psi}^i_q(x) [M_q]_{ij} \psi^j_q(x) \right), \qquad (2.8)
\end{aligned}
$$

and $A_\mu(x)$ and $\psi(x)$ are the gluon and quark fields respectively. The matrix M is the Dirac operator. The fermions are represented by Grassmann variables ψ and $\overline{\psi}$. These can be integrated out exactly giving

$$
\mathcal{Z} = \int \mathcal{D}A_\mu \det M \exp \left(\frac{1}{4} F^a_{\mu\nu}(x) F^{a\mu\nu}(x) \right). \qquad (2.9)
$$

The fermionic contribution is now contained in a highly non–local term $\det M$ and the partition function becomes an integral over only the background gauge group configurations. After integration over the fermions, the action may be written as:

$$
S_{\rm QCD} = S_{\rm gauge} + S_{\rm quarks} = \int d^4x \left(\frac{1}{4} F^a_{\mu\nu}(x) F^{a\mu\nu}(x) \right) - \sum_q \log \left(\det [M_q]_{ij} \right), \qquad (2.10)
$$

where the sum is over the quark flavours.

The fermion matrix plays a central role in full QCD simulations. This matrix describes all the quark and anti–quark interactions in the QCD vacuum (the sea quarks). These interactions may therefore be switched off by setting $\det M$ = constant. This approximation is called the quenched approximation and largely reduces the computational expense. As $m_q \longrightarrow \infty$ the sea quark simply renormalize the strong coupling constant.

Results for physical observables are obtained by calculating expectation values

$$
\begin{aligned}
\langle \mathcal{O}[U_\mu(x)] \rangle \; &= \; \frac{1}{\mathcal{Z}} \int \mathcal{D}U \mathcal{O}[U_\mu(x)] e^{-S[U_\mu(x)]} \\
&= \; \lim_{M_c \longrightarrow \infty} \frac{1}{M_c} \sum_{i=1}^{M_c} \mathcal{O}[(U_\mu(x))_i], \qquad (2.11)
\end{aligned}
$$

over an uncorrelated ensemble of gauge field configuration $(\{U_1(x), .., U_{M_c}(x)\})$ generated with a probability distribution P_i

$$
\text{where} \qquad P_i \propto \begin{cases} e^{-S[U_\mu(x)]} & \text{quenched QCD} \\ detMe^{-S[U_\mu(x)]} & \text{full QCD}. \end{cases}
$$

In general $\mathcal{O}$ is any given combination of operators expressed in terms of time ordered products of gauge and quark fields. The quark fields in $\mathcal{O}$ are expressed in terms of quark propagators using Wick's theorem for contracting fields.

The basic building block for fermionic quantities is the Feynman propagator which is obtained by calculating the inverse of the Dirac operator on a given background field. A given element of this matrix $(M^{-1})^{y,j,b}_{x,i,a}$ is the amplitude for the propagation of a quark from site x with spin–colour i, a to site–spin and colour y, j and b.

2.1.3 The Simplest Formulation of the Actions

The Gauge Action

In the continuum the gauge action in Euclidean space–time for QCD is given by

$$S_G = \frac{1}{4}\int d^4x F_{\mu\nu}(x)F_{\mu\nu}(x). \tag{2.12}$$

It is this quantity that we want our discretized lattice gauge action to approach in the continuum limit. Here $F_{\mu\nu}(x)$ is the usual non–Abelian field strength tensor

$$F_{\mu\nu}(x) = \partial_\mu A_\nu(x) - \partial_\nu A_\mu(x) + ig[A_\mu(x), A_\nu(x)]. \tag{2.13}$$

The discretization is not done directly via the value of the gluon field but rather through the link variable, Eq. (2.1), where the $\mathcal{P}$–operator path–orders the $A_\mu(x)$ gluon fields along the integration contour (a closed path called a Wilson loop). We use $U_\mu(x)$ instead of $A_\mu(x)$ on the lattice because it is impossible to formulate a lattice version of QCD directly in terms of $A_\mu(x)$ that has exact gauge invariance. The link variables on the other hand transform simply under a gauge transformation:

$$U_\mu(x) \longrightarrow U_\mu^g(x) = g(x)U_\mu(x)g^\dagger(x + a\hat{\mu}). \tag{2.14}$$

This quantity makes it possible to construct out of the product of link variables, under a closed loop, a gauge invariant action that will approach Eq. (2.12) in the continuum limit.

The gauge action is constructed from an arbitrary Wilson loop

$$\begin{aligned}\mathcal{C}_{\mu\nu}(x) &= \frac{1}{N_c}\mathcal{P}\left\{Tr\left(\exp\left(-ig\oint_{\mathcal{C}} A(x)\cdot dx\right)\right)\right\} \\ &= \mathcal{P}\left\{\frac{1}{N_c}Tr\left[1 - ig\left(\oint_{\mathcal{C}} A(x)\cdot dx\right) - \frac{g^2}{2!}\left(\oint_{\mathcal{C}} A(x)\cdot dx\right)^2 + \mathcal{O}(g^3)\right]\right\}.\end{aligned} \tag{2.15}$$

Using Stokes theorem and performing a Taylor expansion of $D_\mu A_\nu(x + x_0)$ around the centre point of the Wilson loop x_0 in coordinate gauge $A\cdot x = 0$, it can be shown that:

$$\begin{aligned}\oint_{\mathcal{C}} A(x)\cdot dx &= \int_a^b dx_\mu dx_\nu \left[D_\mu A_\nu(x + x_0) - D_\nu A_\mu(x + x_0)\right] \qquad (2.16)\\ &= \int_a^b dx_\mu dx_\nu \Big[F_{\mu\nu}(x_0) + (x_\mu D_\mu + x_\nu D_\nu) F_{\mu\nu}(x_0) \\ &\quad + \frac{1}{2}\left(x_\mu^2 D_\mu^2 + x_\nu^2 D_\nu^2\right) F_{\mu\nu}(x_0) + \mathcal{O}(a^2 g^2, a^4)\Big].\end{aligned}$$

The integration bounds are determined by the size of the Wilson loop. In the case of a simple plaquette contour variable we have for the Wilson action, for both x_μ and x_ν, $-a/2 \le x_\mu \le a/2$ and $-a/2 \le x_\nu \le a/2$ respectively. Hence we have

$$\oint_{1\times 1} A(x)\cdot dx = a^2 F_{\mu\nu}(x_0) + \frac{a^4}{24}\left(D_\mu^2 + D_\nu^2\right) F_{\mu\nu}(x_0) + \mathcal{O}(a^6, A^2). \tag{2.17}$$

The action is obtain by taking the real part of the trace of $\mathcal{C}_{\mu\nu}(x)$. Hence we may write for the Wilson action

$$\begin{aligned} P_{\mu\nu}(x) &= \frac{1}{N_c}\mathrm{Tr}\left[I - \frac{g^2}{2!}\left(\oint_{\mathcal{C}} A(x)\cdot dx\right)^2 + ...\right] \\ &= 1 - \frac{g^2}{6}a^4\mathrm{Tr}\left(F^2_{\mu\nu}(x_0)\right) - \frac{g^2}{6}\frac{a^6}{12}\mathrm{Tr}\left(F_{\mu\nu}(x_0)\left(D^2_\mu + D^2_\nu\right)F_{\mu\nu}(x_0)\right) + \dots . \end{aligned} \tag{2.18}$$

Using this expansion we obtain the commonly known standard Wilson gluonic action:

$$S_{\mathrm{Wil}} = \beta\sum_{x,\mu>\nu}\left(1 - P_{\mu\nu}(x)\right) \longrightarrow \int d^4x\frac{1}{2}\mathrm{Tr}\left(F^2_{\mu\nu}(x)\right) + \mathcal{O}(a^2), \tag{2.19}$$

where $\beta = 6/g^2$ is a tuneable variable in numerical simulations. We can see that this action has $\mathcal{O}(a^2)$ errors which may become significant during numerical simulation if the lattice is not fine enough.

The Fermionic Action

The fermionic part of the QCD action was first discretized by Wilson [14] by replacing the derivative with symmetrized difference operators and included appropriate gauge links to maintain gauge invariance. The discretized part of the action then becomes

$$\overline{\psi}\not{D}\psi = \frac{1}{2a}\overline{\psi}\sum_\mu \gamma_\mu\left[U_\mu(x)\psi(x+\hat{\mu}) - U^\dagger_\mu(x-\hat{\mu})\psi(x-\hat{\mu})\right]. \tag{2.20}$$

Performing a Taylor expansion of the gauge links $U_\mu(x) = 1 + iagA(x+\hat{\mu}/2) + \mathcal{O}$ in powers of the lattice spacing and similarly for the fermion fields $\psi(x+\hat{\mu}) = \psi + a\psi'(x) + \mathcal{O}$ one sees that in the limit $a \longrightarrow 0$ one recovers the kinetic part of the standard continuum Dirac action in Euclidean space–time. Hence one arrives at the simplest ("naive") lattice action for fermions,

$$\begin{aligned} S &= \sum_x\left[m_q\overline{\psi}(x)\psi(x) + \frac{1}{2a}\sum_\mu\overline{\psi}(x)\gamma_\mu\left[U_\mu(x)\psi(x+\hat{\mu}) - U^\dagger_\mu(x-\hat{\mu})\psi(x-\hat{\mu})\right]\right] \\ &\equiv \sum_x\overline{\psi}(x)M[U]_{xy}\psi(y), \end{aligned} \tag{2.21}$$

where the interaction matrix, $M[U]_{xy}$, is given by

$$M_{ij}[U] = m_q\delta_{ij} + \frac{1}{2a}\sum_\mu\gamma_\mu\left[U_\mu(x)\delta_{i\,j-\mu} - U^\dagger_\mu(x-\hat{\mu})\delta_{i\,j+\mu}\right], \tag{2.22}$$

and the Euclidean hermitian γ matrices satisfy $\{\gamma_\mu,\gamma_\nu\} = 2\delta_{\mu\nu}$. From this simple action we can easily extract the 2–point function, $S(p)$. It is given by the inverse of $M_{ij}[U]$. To see how this is done one can Fourier transform the quark fields over the Brillouin zone (BZ) $[-\pi/a, \pi/a]$ as

$$\psi(x) = \int_{-\frac{\pi}{a}}^{\frac{\pi}{a}}\frac{d^4p}{(2\pi)^4}\tilde{\psi}(p)e^{ip\cdot x}, \text{ and } \overline{\psi}(x) = \int_{-\frac{\pi}{a}}^{\frac{\pi}{a}}\frac{d^4p'}{(2\pi)^4}\tilde{\overline{\psi}}(p')e^{-ip'\cdot x}. \tag{2.23}$$

One then finds that for a 2–point function in momentum space that

$$S^{-1}(p) = m_q + i\sum_{\mu} \gamma_\mu k_\mu = M[U](p). \tag{2.24}$$

Note that we are working in the infinite volume limit here. Here, $k_\mu = (1/a)\sin(p_\mu a)$. A plot, Fig. 2.4, of k_μ vs p_μ shows the zeros at the edge of the BZ. These zeros destroy

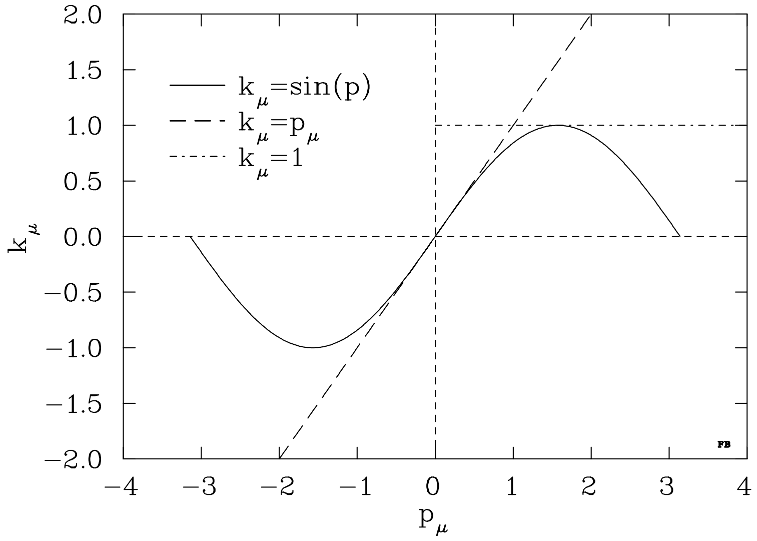


Figure 2.4: The lattice momentum k_μ plotted versus the discrete momentum p_μ over the the interval $[-\pi/a, \pi/a]$. The small dash–dot line is $k_\mu = 1$ and the dash line is line $k_\mu = p_\mu$. The continuum limit is determined by the momenta in the neighbourhood of $p_\mu = 0$ and $p_\mu = \pm\pi/a$.

the correct continuum limit in the fermionic case, because there exist sixteen regions of interest in the 2–point function where k_μ takes a finite value in the limit $a \longrightarrow 0$. Of these, fifteen regions involve high momentum excitations of the order of π/a, which gives rise to a momentum distribution function having the form resembling that of a single particle propagator. These excitations are pure lattice artifacts. In fact, in d–space the number would be 2^d (0 and π/a for each dimension), i.e. it doubles for each additional dimensions.

The fermion doubling problem can be overcome by cancelling out the lattice artifacts. This is done by introducing irrelevant operators at the expanse of an explicit breaking of chiral symmetry even for $m_q \longrightarrow 0$.

The naive lattice action for fermions can be modified by a term which vanishes in the continuum limit. This fermion action is known as Wilson fermion action, it has $\mathcal{O}(a)$ errors and is defined as

$$\begin{aligned} S_W[U,\overline{\psi},\psi] &= \sum_x \Big[(m_q + 4r)\overline{\psi}(x)\psi(x) \\ &- \frac{1}{2}\sum_\mu \overline{\psi}(x)(r-\gamma_\mu)U_\mu(x)\psi(x+a\hat{\mu}) + \overline{\psi}(x+a\hat{\mu})(r+\gamma_\mu)U^\dagger_\mu(x)\psi(x)\Big] \\ &= \sum_{xy}\overline{\psi}(x)D_W(x,y)\psi(y), \end{aligned} \tag{2.25}$$

with $D_W(x,y)$ the Wilson fermion operator:

$$D_W(x,y) = (m_q + 4r)\delta_{x,y} - \tag{2.26}$$

$$\frac{1}{2}\sum_{\mu}(r-\gamma_\mu)U_\mu(x)\delta_{y,x+a\hat{\mu}}+(r+\gamma_\mu)U_\mu^\dagger(x-a\hat{\mu})\delta_{y,x-a\hat{\mu}}.$$

Where r is the Wilson coefficient. The terms containing r are the irrelevant operators introduced to eliminate the fermion doubling problem. Here m_q is the lattice bare mass related to the hopping parameter κ via

$$\kappa=\frac{1}{2m+8r}\ \text{ and }\ m_q\equiv\frac{1}{2a}\left(\frac{1}{\kappa}-\frac{1}{\kappa_c}\right). \tag{2.27}$$

Chiral symmetry breaking gives rise to additive mass renormalization. When the gluonic interactions are turned off ($U_\mu(x)\equiv I$), the critical value for the hopping parameter is $\kappa_c^{(0)}=1/8$. When the gluons are present, κ_c is defined as the value of κ at which the pion mass vanishes. In this case the critical value tends to diverge away from its tree value of an eighth. The Wilson fermions, for $r\neq 0$, break chiral symmetry even for $m_q\longrightarrow 0$.

The Wilson gauge action, Eq. (2.19) and the Wilson fermionic action Eq. (2.25) are the simplest lattice actions which have the correct continuum limit ($a\longrightarrow 0$). These actions have been used in a large number of lattice simulations. Simulations in quenched QCD for the light hadron spectrum done by the CP-PACS Collaboration [15] suggests a deviation of 11% from experiment. There has been many other studies using these actions. The consistency of the results with the continuum were really limited by the computational power available at the time because people had to drive the lattice spacing to small values in order to get reasonable results. The computational factor is slowly becoming less of an issue these days, and we can expect to improve over the next few years. Increasing computational power is not really the fastest solution to the problem. We can also improve the operators, actions and algorithms to reduce the discretization errors to their minimum for a given lattice spacing. This is what the subsequent chapters of this thesis are primarily concerned with.

Chapter 3

Numerical Simulations and Markov Chains

3.1 Markov Chains

The goal in numerical lattice simulations is to approximate the Feynman path integral in Euclidean space–time

$$\mathcal{G}_{\mathrm{E}}(x_1,..,x_n) = \langle 0|\Phi(x_1)..\Phi(x_n)|0\rangle_{\mathrm{E}} = \frac{\int[\mathcal{D}\Phi](\prod_i^n \Phi(x_j))e^{-S^{\mathrm{E}}_{QCD}}}{\int[\mathcal{D}\Phi]e^{-S^{\mathrm{E}}_{QCD}}}. \tag{3.1}$$

Calculating Eq. (3.1), can be extremely time consuming.

For example, in the case of $SU_c(3)$ on a small lattice, hundred of thousands of integrations are required for just one configuration. One needs many of these configurations in order to represent properly the system under scrutiny. Hence direct evaluation of Eq. (3.1) is unthinkable for the lattice sizes that we will need to consider. However using statistical methods to evaluate Eq. (3.1) is possible. In fact, since most of the configurations will have an action which is very large, only a small fraction of them will contribute significantly to this path integral. Hence an efficient way of computing an ensemble average would consist in generating a sequence of uncorrelated field configurations with a probability distribution given by the Boltzmann factor $e^{-S^{\mathrm{E}}_{\mathrm{QCD}}}$. This is the technique of "**importance sampling**". If the configurations C_i, for $i = 1,..,N$, denote a representative sequence of configurations generated via the above technique, then the ensemble average approximation for Eq. (3.1) is

$$\mathcal{G}_{\mathrm{E}}(x_1,..,x_n)\approx\frac{1}{N}\sum_{i=1}^{N}\mathcal{G}(C_i). \tag{3.2}$$

What is required, is an algorithm that systematically generates such a sequence of configurations.

The configurations can be generated via a Markov process with each configuration being an element of a Markov chain.

Let us denote each of these configurations by a discrete index. Because of computer round–off limitations the set of all possible lattice gauge configurations is enormously large but finite. Hence, we may denote $C_1, C_2, ..$ to be a countable set of states of the system. Consider a stochastic process in which a finite set of configurations is generated one after the other according to some transition probability $P(C_i \longrightarrow C_j) = P_{ij}$ for going from configuration C_i to C_j.

Now given a set of configurations $\{C_{\tau_i}\}$ generated by the Markov process with mean recurence time τ_i (see Appendix C) and an observable evaluated on this set of states, $O(C_{\tau_i})$, we can perform an ensemble average of $O(C_{\tau_i})$ over the number of configurations, N, of the Markov chain by

$$\langle O \rangle_N = \frac{1}{N} \sum_{i=1}^{N} O(C_{\tau_i}). \tag{3.3}$$

It is this quantity which we compute, and which we want to equal the ensemble average corresponding to a given Boltzmann distribution. Here we restrict ourselves to **irreducible aperiodic** Markov chains whose states are **positive**, see Appendix C for a detailed definition of these terms. I now state some important results for **irreducible aperiodic** Markov chains with **positive** states [1].

Theorem 3.1 *If the chain is irreducible and the states are positive and aperiodic, then the limit $N \longrightarrow \infty$ of Eq. (C.1) exists, and is unique; in particular one can show* [2]

$$\lim_{N \to \infty} P_{ij}^{(N)} = \pi_j, \tag{3.4}$$

where $\pi_j > 0$ are the numbers which satisfy the following equations:

$$\sum_j \pi_j = 1 \text{ and } \pi_j = \sum_i \pi_i P_{ij}. \tag{3.5}$$

This theorem says that after a large number of Markov steps (i.e. as $N \longrightarrow \infty$) the obtained configuration is completely uncorrelated from the initial configuration used to start the Markov process. Furthermore, Eq. (3.5) states that the system is left unchanged after further updates with transition probabilities P_{ij}. When Eq. (3.5) and Eq. (3.4) are combined we see that it is just the condition that the system has reached equilibrium with π_i the probability of finding the configuration C_i. The second important theorem concerning the ensemble average is as follows:

Theorem 3.2 *If the chain is irreducible and its states are positive, and if*

$$\tau_i^{(2)} \equiv \sum_{n=1}^{\infty} n^2 P_{ii}^{(n)} < \infty \tag{3.6}$$

then the time average Eq. (3.3) approaches the ensemble average

$$\langle O \rangle_N = \frac{1}{N} \sum_{i=1}^{N} \pi_i O(C_i). \tag{3.7}$$

with a statistical uncertainty of order $O(1/\sqrt{N})$.

[1]The following two theorems are taken from [5], the proof of these theorems may be found in Hammersley and Handscomb (1975).

[2]For a detailed discussion, consult the books by Hammersley & Hamdscomb (1975), Clark & Disney (1985).

Theorems 3.1 and 3.2 provide the basis for using the Markov process to calculate the ensemble average Eq. (3.2). In fact it is possible to show [5, 16] that for a Markov process to sample a probability distribution of the type that is found in lattice gauge theory, $\exp(-S(C))$, it is sufficient to require that the transition probability P_{ij} satisfies detailed balance:

$$\exp(-S(C_i))P(C_i \longrightarrow C_j) = \exp(-S(C_j))P(C_j \longrightarrow C_i). \tag{3.8}$$

The transition probability P_{ij} should be understood as a rule for selecting the next configuration in the Markov chain according to the one directly preceeding it.

3.2 The Metropolis Algorithm

This method was originally proposed by Metropolis et al. [17] and is applicable in principle to any system.

Let C be any configuration which is to be updated. A new configuration C' is then suggested with a transition probability $P_0(C \longrightarrow C')$ which satisfies the following condition:

$$P_0(C \longrightarrow C') = P_0(C' \longrightarrow C). \tag{3.9}$$

Having suggested a configuration C', we must perform an accept/reject step on that configuration. The accept/reject step is based by comparing the action of these two configuration. If the transition probability of going from the configuration of C to configuration C' satisfies detail balance, then the decision is made as follows:

1. If $\exp(-S[C']) > \exp(-S[C])$, i.e. if the action decreases then the configuration C' is accepted.

2. If on the other hand $\exp(-S[C']) < \exp(-S[C])$, i.e. if the action increases, then accept C' only with a probability $P(C')$

$$P(C') = \frac{\exp(-S[C'])}{\exp(-S[C])} = \exp\left\{-(S[C'] - S[C])\right\}. \tag{3.10}$$

   For a random number $R \in [0, 1]$, if $R \leqslant \exp\{-(S[C'] - S[C])\}$ then accept C'. Otherwise reject C' and keep the old configuration C.

In order to be guaranteed that the system reaches equilibrium, we must ensure that the algorithm satisfies detailed balance. This is shown by using the fact that the transition probability $P(C \longrightarrow C')$ is the product of the probability $P_0(C \longrightarrow C')$ for suggesting C' as the new configuration and the probability of accepting it. We see that the algorithm for $P(C \longrightarrow C')$ implies the following statements:

$$\text{If } \exp(-S[C']) > \exp(-S[C]) \text{ then } P(C \longrightarrow C') = P_0(C \longrightarrow C'),$$

and

$$P(C' \longrightarrow C) = P_0(C' \longrightarrow C)\frac{\exp(-S[C])}{\exp(-S[C'])}.$$

Since $P_0(C \longrightarrow C') = P_0(C' \longrightarrow C)$, then we see that detailed balance is satisfied, i.e.

$$\exp(-S[C])P(C \longrightarrow C') = \exp(-S[C'])P(C' \longrightarrow C). \tag{3.11}$$

Conversely,

$$\text{if } \exp(-S[C']) < \exp(-S[C]) \text{ then } P(C \longrightarrow C') = P_0(C \longrightarrow C')\exp\{-(S[C'] - S[C])\},$$

and

$$P(C' \longrightarrow C) = P_0(C' \longrightarrow C), \tag{3.12}$$

which implies that detailed balance, Eq. (3.11), holds again.

3.2.1 The 2D Ising Model with the Metropolis Algorithm

In this simple model, I illustrate the use of the Metropolis algorithm. The Model can be taken as a crude description of magnetic material or a binary alloy. We use Monte Carlo methods to obtain a numerical result for the phase transition. The Ising model consists of a set of degrees of freedom interacting with each other via nearest neighbour interactions. These might represent, for example, the magnetic moments of the atoms in a solid. Here we do not consider the possibility of an external field. The spin variables are located on the sites of an $N_x \times N_y$ lattice. The spin can then be labelled as σ_{ij} where i, j are the indices for the spatial directions. These spin variables can take the values of,

$$\sigma_{ij} = \begin{cases} +1 & \text{spin up} \\ -1 & \text{spin down.} \end{cases} \tag{3.13}$$

So the action that we are interested in takes the following form

$$S(\sigma) = -\beta \sum_{i,j} \sigma_{ij}(\sigma_{i+1j} + \sigma_{ij+1}). \tag{3.14}$$

The idea is to sweep systematically through the lattice and considering whether or not to flip each spin one at the time. The lattice is updated using the Metropolis algorithm [17] as described in Sec. 3.2. Hence we consider two configurations S and S_t (S_t for S_{trial}) differing only by the flipping of one spin σ_{ij}. The acceptance ratio of the action is given by the formulae,

$$r = \frac{\exp(-S)}{\exp(-S_t)}. \tag{3.15}$$

If $r > 1$, the spin is automatically flipped. On the other hand if $r < 1$, that is the action has increased, then the trial configuration is accepted or equivalently the spin is flipped, if r is greater than a uniformly distributed pseudo-random number between 0 and 1. From Eq.(3.14) we see that only the terms involving σ_{ij} will contribute, so we have

$$r = \exp\left(-2\beta\sigma_{ij}(\sigma_{i+1j} + \sigma_{i-1j} + \sigma_{ij+1} + \sigma_{ij-1})\right). \tag{3.16}$$

This is illustrated in Fig. 3.1.

A periodic boundary condition on the lattice is assumed here, for example the lower neighbours of the spins with $i = N_x$ are those with $i = 1$ and left hand neighbours of

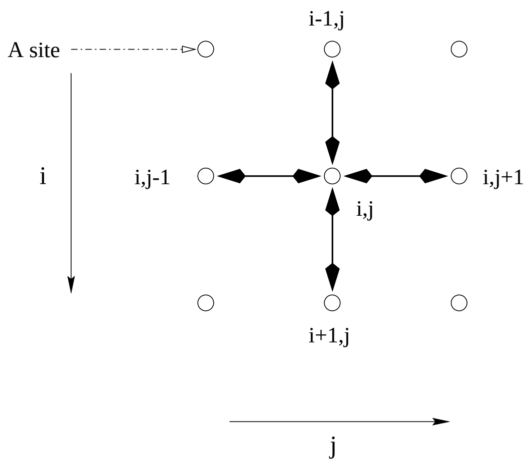


Figure 3.1: Schematic illustration of the 2D Ising Model.

those with $j = 1$ are those with $j = N_y$, the lattice therefore has the topology of a torus.

Here we are interested in measuring the magnetization of the system as it evolves for the coupling, β, between 0 and 1, i.e. $0 \leq \beta \leq 1$. Although this model has been solved analytically it is useful to obtain a numerical solution as an illustration of the Metropolis algorithm and Monte Carlo methods. This model was coded in Fortran 90. After a few thousands sweeps the system is thermalized. One can then extract physical quantities such as the magnetization. The resulting net magnetization, which is really the sum of the spins, is plotted in Fig. 3.2 for single hit Metropolis. From this plot one can observe a second order phase transition which means that there is a value of $\beta \cong 0.4406$ at which the slope of the magnetization approaches infinity as the number of states approaches infinity.

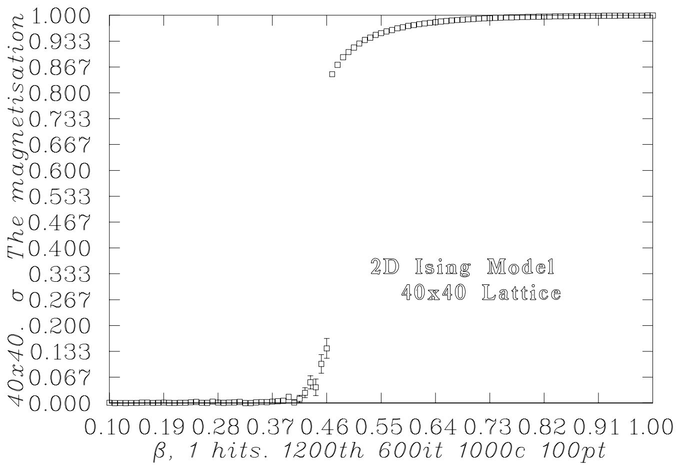


Figure 3.2: Numerical illustration of the 2D Ising Model for single hit Metropolis.

Further Numerical Result, Multiple Hit Metropolis

Further analysis can be done in the numerical study of the Metropolis algorithm. This simple technique is called Multiple Hit Metropolis. It consists of repeating the probability ratio, Eq.(3.16), and the random number comparison. For example when the number of hits is one, the comparison is done only once, so if a spin configuration is not updated then we move on hoping to update it on the next sweep. On the other hand if we hit 5 times we do the comparison 5 five times. Therefore increasing the chances for a spin update. For this model, the effect are subtle as illustrated in Fig. 3.3.

The data were obtained on an Alpha Workstation. The data plotted in Fig. 3.2 and Fig. 3.3, is for an 40×40 lattice. The system is thermalised in about 1200 hundred sweeps. I then obtain 1000 configurations separated by 600 sweeps. Since the errors scale as $\frac{1}{\sqrt{N}}$, N being the number of configurations, the more configurations we have the smaller the error bars are. This process is repeated for 100 different β values ranging from 0 and 1. For these numbers it takes from about three quarters of a full day for $single - hit$ to about three full days for $5 - hits$.

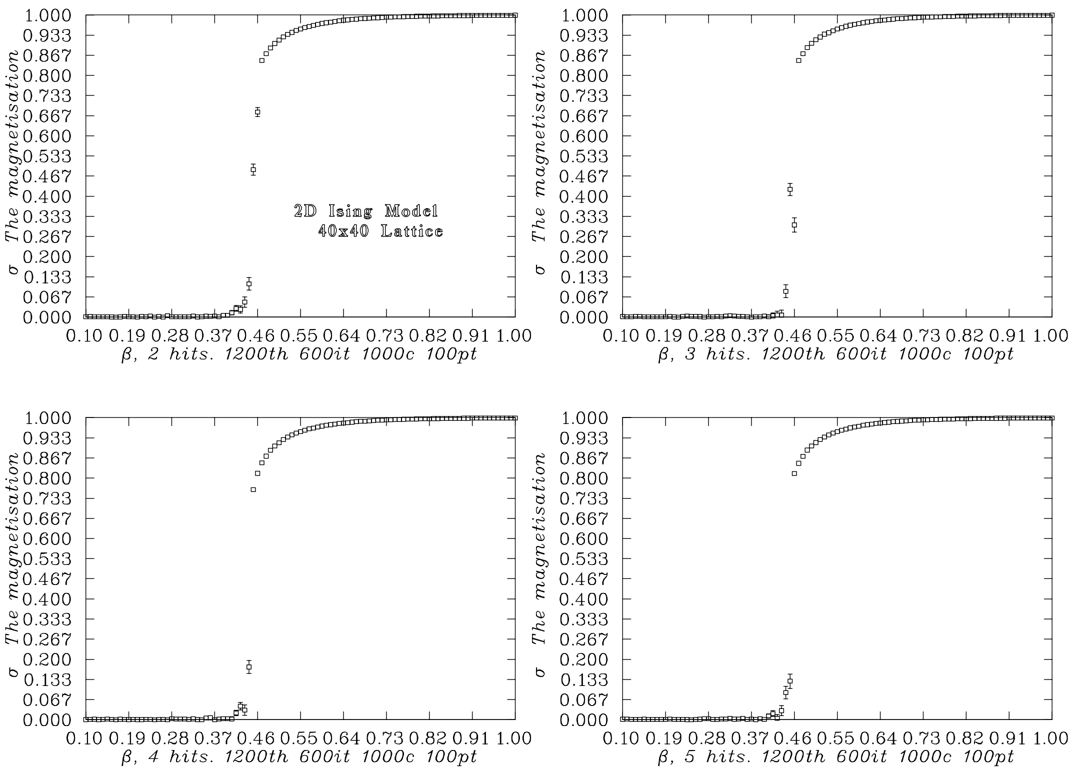


Figure 3.3: Numerical illustration of the 2D Ising Model for multiple hit Metropolis.

Chapter 4

Generating Gauge Field Configurations

The purpose of this chapter is to develop an algorithm that will generate an uncorrelated ensemble of gauge field configuration in a given gauge group, namely $SU_c(N)$. The link variable is an element of $SU_c(N)$ and is associated with each nearest-neighbour on the four dimensional simple hypercubic lattice. Here we start with the standard Wilson action (see Section 2.1.3) but the gauge action is totally arbitrary, that is the algorithm is also applicable to higher improved gauge actions. In this chapter I only consider the cases $N = 2$ and 3. The algorithm for $N \geqslant 3$ may be found in Ref. [21].

Here we are interested in the path integral,

$$\mathcal{Z} = \int \left(\prod_{\mu} dU_{\mu}(x) \right) \exp\left(-\beta S[U]\right), \tag{4.1}$$

which defines the quantum theory. It is the same partition function as Eq. (2.7) but written in the form of a statistical system at a temperature $T = \frac{1}{\beta}$. The system is treated by obtaining an ensemble of configurations which simulates an ensemble in equilibrium at this temperature T. Starting in some initial configuration (ordered or random start) we successively bring a heat bath into contact with each link variable. Each link, $U_\mu(x)$, in turn is replaced with a new link variable, $U'_\mu(x)$, chosen randomly from the entire group with probability density proportional to the Boltzmann factor

$$dP(U') \propto \exp\left(-\beta S[U']\right) dU', \tag{4.2}$$

where $S[U']$ is the action evaluated with the given link having value $U'(x)$ and all the other links fixed at their previous values.

4.1 The Algorithm for the $SU_c(2)$ Gauge Group

I now describe the algorithm for generating group element with weighting given by Eq. (4.2) for the $SU_c(2)$ gauge group [20]. One of the key aspects of the $SU_c(2)$ gauge group is that any group element may be parameterized in the form:

$$U(x) = a_0(x)I + i\vec{a}(x)\cdot\vec{\sigma}, \tag{4.3}$$

where $a_0(x), \vec{a}(x) \in \mathbf{R}$. The 2×2 matrices, σ_i, are the usual Pauli matrices, so at each lattice site we have an $SU_c(2)$ matrix. From now on, I will not explicitly write the lattice site dependence on the group elements but it is understood to be there.

These matrices have the properties that their trace is 0 and a determinant of 1. Since we require that the link variables are unitary we must have $U^\dagger U = UU^\dagger = I$, this implies that $a_\mu a_\mu = a_0^2 + \vec{a}^2 = 1$, and therefore the a_μ lie on the surface of the 3–sphere, S^3. In this notation I can obtain the invariant group measure,

$$dU = \frac{1}{2\pi^2}\delta(a^2-1)d^4a, \tag{4.4}$$

where $\frac{1}{2\pi^2}$ is a normalization factor and $2\pi^2$ is the solid angle for S^3 such that $\int dU = 1$. While working on a particular link, we need to consider only the contribution to the action coming from the six plaquettes containing that link, i.e., the staples. The staples are depicted in Fig. 2.3. If we denote the staples by $\widetilde{U}_{\alpha=1,..,6}(x)$, the six products of three links variables which interact with the link in question, one can show that the probability density can be written as

$$\begin{aligned} dP(U) &\propto \exp\left(-\beta S[U]\right) dU \\ &\propto \exp\left(-\beta \sum_{\Box} 1 - \frac{1}{N_c}\mathrm{Tr}(U_{\mu\nu}(x))\right) dU \\ &\propto \exp\left(\frac{\beta}{N_c}\mathrm{Tr}\left(U\sum_{\alpha=1}^{6}\widetilde{U_\alpha}(x)\right)\right) dU. \end{aligned} \tag{4.5}$$

Here $U_{\mu\nu}(x)$ is the product of the links defining the plaquette, i.e., Eq. (2.2). Using the useful property of the group $SU_c(2)$, namely that for $a, b \in SU_c(2)$ we have $a+b$ is proportional to an $SU_c(2)$ element, one can simplify Eq. (4.5) as follows:

$$\sum_{\alpha=1}^{6}\widetilde{U_\alpha} = k\overline{U}. \tag{4.6}$$

The space-time matrix $k \equiv k(x)$ is a constant of proportion, $\overline{U} \in SU_c(2)$ which implies that $\det(\overline{U}) = I$. To find k we take the determinant on both sides,

$$\det\left(\sum_{\alpha=1}^{6}\widetilde{U_\alpha}\right) = \det(k\overline{U}) = \det(kI)\det(\overline{U}) = k^2, \tag{4.7}$$

so we obtain,

$$k = \left(\det\left(\sum_{\alpha=1}^{6}\widetilde{U_\alpha}\right)\right)^{\frac{1}{2}}. \tag{4.8}$$

Using this result together with Eq. (4.5), we obtain the following density,

$$\begin{aligned} dP(U\overline{U}^{-1}) &\propto \exp\left(\frac{\beta}{N_c}\mathrm{Tr}\left(U\overline{U}^{-1}\sum_{\alpha=1}^{6}\widetilde{U_\alpha}\right)\right) dU \\ &= \exp\left(\frac{\beta}{N_c}k\mathrm{Tr}(U)\right) dU. \end{aligned} \tag{4.9}$$

Finally substituting the invariant group measure Eq. (4.4), and the trace of the link variable U, $\mathrm{Tr}(U) = \mathrm{Tr}(a_0 I + i\vec{a}\cdot\vec{\sigma}) = 2a_0$, into Eq. (4.9) we obtain,

$$dP(U\overline{U}^{-1}) \propto \frac{1}{2\pi^2}\delta(a^2-1)\exp\left(\frac{2\beta}{N_c}ka_0\right)d^4a. \tag{4.10}$$

The problem reduces to generating points randomly on the unit 3–sphere in a four dimensional space, $\mathbf{R^4}$, with exponential weighting along the a_0 direction. To generate an element U in this fashion, we replace the link variable on the lattice in Eq. (4.3) by

$$U \longrightarrow U' = U\overline{U}^{-1}. \tag{4.11}$$

Since for any $SU_c(2)$ matrices we have $\overline{U}^{\dagger} = \overline{U}^{-1}$, then

$$\overline{U}^{\dagger} = \frac{\left(\sum_{\alpha=1}^{6} \widetilde{U_\alpha}\right)^{\dagger}}{k}, \qquad \text{as} \quad k \in \mathbf{R}.$$

This leads to

$$U \longrightarrow U' = U\frac{\left(\sum_{\alpha=1}^{6} \widetilde{U_\alpha}\right)^{\dagger}}{k}, \qquad \text{where} \quad U = a_0 I + i\vec{a}\cdot\vec{\sigma}. \tag{4.12}$$

To generate the appropriately weighted points on S^3, we first note that the integration over $|\vec{a}|$ can be done using the delta function. So

$$\delta(a^2-1)\exp\left(\frac{2\beta}{N_c}ka_0\right)d^4a = \frac{1}{2}|\vec{a}|\exp\left(\frac{2\beta}{N_c}ka_0\right)da_0 d\Omega, \tag{4.13}$$

where $d\Omega$ is the differential solid angle of $\vec{a}$. The length of the vector $\vec{a}$ is just $\sqrt{a_1^2+a_2^2+a_3^2}$. Because a_μ lies on S^3 we can rewrite the length as $|\vec{a}| = \sqrt{1-a_0^2}$. Eq. (4.13) then becomes

$$\delta(a^2-1)\exp\left(\frac{2\beta}{N_c}ka_0\right)d^4a = \frac{1}{2}\left(\sqrt{1-a_0^2}\right)\exp\left(\frac{2\beta}{N_c}ka_0\right)da_0 d\Omega. \tag{4.14}$$

Thus we need to generate a_0 stochastically on $[-1,1]$ with probability:

$$P(a_0) \propto \left(\sqrt{1-a_0^2}\right)\exp\left(\frac{2\beta}{N_c}ka_0\right). \tag{4.15}$$

The spatial direction, the vector $\vec{a}$, is chosen totally randomly with a Gaussian probability distribution function as described in Section D.1. We are now able to write down a systematic approach to updating the new link variable,

1. The algorithm for the a_0 selection begins with a trial for a_0,

$$a_0 = 1 + \frac{N_c}{2\beta k}\ln(x) \qquad \Longrightarrow \qquad x = e^{-\frac{2\beta k}{N_c}}\exp\left(\frac{2\beta}{N_c}ka_0\right). \tag{4.16}$$

   The variable x is a random number uniformly distributed. We want

$$a_0 \in [-1,1] \Longrightarrow \left\{ \begin{array}{ll} a_0 = +1 & \longrightarrow x = 1 \\ a_0 = -1 & \longrightarrow x = e^{-\frac{4}{N_c}\beta k}. \end{array} \right\} \Longrightarrow x \in [e^{-\frac{4}{N_c}\beta k}, 1]. \tag{4.17}$$

   This generates a_0 distributed with exponential weight $e^{\frac{2}{N_c}\beta k a_0}$.

2. To correct for the factor $(1-a_0)^{\frac{1}{2}}$ in Eq. (4.15), reject this a_0 with probability $1-(1-a_0)^{\frac{1}{2}}$ and select a new trial a_0.

3. Repeat step.1 and .2 until an a_0 is accepted.

4.2 Computer Code for $SU_c(2)$

The difference between serial and parallel computation is that in the first case the computation is done by only one processor at a time, which therefore means that only one site can be updated at once. This is a contrast with parallel computation where multiple processors can perform updates at the same time. In that case, it is possible to select a number of sites which can be updated simultaneously.

In order to take advantage of this resource and not waste any computational time by leaving some of the processors idle, one must find a way to spread the information over the entire machine in such a way that the number of working processors is maximized at each flop. In lattice QCD it is usually accomplished by performing some masking procedure [18].

The variables contained in the computer code are separated into two distinct categories. The first one, called "global variables", contains the variables that are globally spread over the entire program (the link variable, $U_\mu(x)$ for example) and any variables that are in the argument list. The real and imaginary part of the link, $U_\mu(x)$, variables are explicitly separated into two different variables (called the *ur* and *ui*) to give better control over the data. The other category, called "local variables", only contains variables that are locally used within the routine.

4.2.1 Masking and Parallel Computing

When performing a Monte Carlo sweep of the entire lattice each lattice link must be updated individually using the particular gluon action of interest (e.g., S_G). The action is used in the Monte Carlo accept/reject step for that link in order that detailed balance is ensured at each link update and hence that it is ensured throughout the entire lattice sweep. It is the combination of randomness in the link updates, the maintenance of detailed balance, and decorrelation (ensured by large sweep numbers between the taking of samples) that ensures the desired ensemble of gauge field configurations are produced with the Boltzmann distribution $\exp(-S_G)$.

In the most naive procedure we move through each link on the lattice consecutively updating them one at a time until we have completed a "sweep" through the entire lattice. We then repeat these lattice sweeps as often as required. This simple procedure is highly inefficient on a parallel computing architecture, where we can be updating many links at the same time. However, there is a fundamental limitation to this parallelism, i.e., we will violate detailed balance and corrupt our data if we try to simultaneously update a link while information about that link is being used in the update of another link. It is crucial that we identify which links can be updated simultaneously and this is determined by the degree of non locality in the action. For example, for an action which contains only nearest neighbour interactions of the links, such as the Wilson action, we can use an efficient "checkerboard" algorithm, which will be described below. In general, the more non–local is the lattice gluon action the fewer are the links that can be simultaneously updated. We see that the improvement program is therefore more expensive to implement, but the benefit of improved actions far outweighs this drawback.

In order to facilitate our discussions we will refer to this concept as "masking", where the lattice links not eliminated by the mask are the ones that can be simultaneously updated in a parallel computing environment. The number of independent masks needed for a particular action determines an upper limit to the parallelism that can be used in a single lattice sweep. As we will see, the best that can be done is to have two masks per link direction and this is for the case of nearest neighbour interactions only.

We will simplify the presentation in the usual way by rescaling all dimensionful quantities by the lattice spacing a. This is equivalent to choosing our units $a = 1$.

4.2.2 Masking for the Standard Wilson Action

In the standard Wilson action, where only neighbouring links are connected by the action, we need only two masks for each of the four link directions. There are two different ways of implementing this masking as we will now discuss.

Checker Board Masking

The standard Wilson action only involves 1×1 Wilson loops (depicted in Fig. 2.2) and is the most fundamental lattice gluonic action. Whenever a given link is being updated, we must not be attempting to update any of the links within any of the 1×1 plaquettes which contains the given link. Consider the link from the lattice site x to $x + \mu$, where μ is one of the four Cartesian unit vectors $\hat{x}, \hat{y}, \hat{z}$, or $\hat{t}$. We see then that the plaquette in Fig. 2.2 forms a "staple" consisting of three links in the μ-ν plane which is attached to the link of interest $U_\mu(x)$. [Note that we are sometimes using x as a shorthand notation for the space-time lattice point $x^\mu \equiv (x, y, z, t)$ as well as for the x-coordinate on the $\hat{x}$ axis. The meaning should be clear from the context.] We could equally well consider the plaquette and staple below the link $U_\mu(x)$ in the figure, which also lies in the μ-ν plane. In addition, for a given Cartesian direction μ, there are three possible choices for ν, i.e., there are three orthogonal planes which contain the link and two staples per plane.

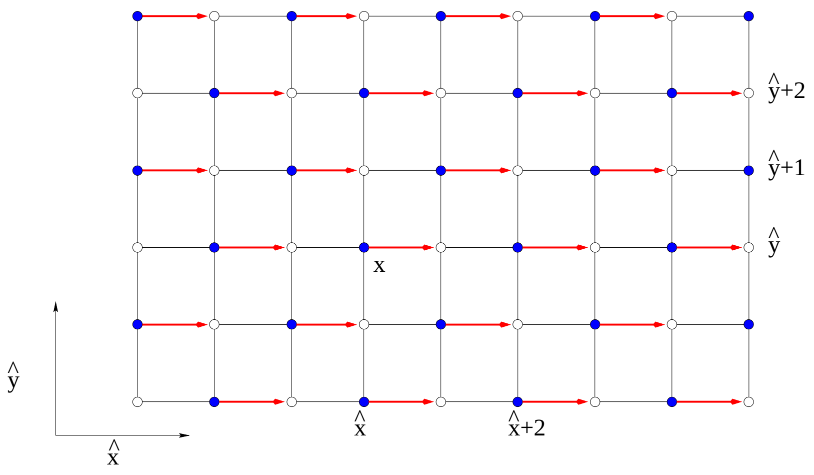


Figure 4.1: Checkerboard masking as seen in an $\hat{x}$–$\hat{y}$ plane of the lattice when using the standard Wilson action. The highlighted links with arrows can be updated simultaneously.

Let us consider, for example, all of the links in the $\hat{x}$-$\hat{y}$ plane which are oriented in the $\hat{x}$ direction. We can see from Fig. 4.1 that we can choose a "checkerboard" of such links that can be updated at the same time without interfering with each other. These links are indicated in the figure as highlighted links with arrows. It is easy to see that none of the links to be updated lie in any of the staples for the other links to be updated and that exactly half of the $\hat{x}$-oriented links in this plane can be simultaneously updated at one time.

We have identified one of the lattice sites in Fig. 4.1 as the site x. If the link variable $U_{\hat{x}}(x)$ is to be updated then from Fig. 4.1, it is observed that the link variables in the $\hat{x}$ direction that can be simultaneously updated are $U_{\hat{x}}(x+2\hat{x})$, $U_{\hat{x}}(x+4\hat{x})$ and so on. So every second link along the $\hat{x}$ direction can be updated at the same time. Now let us consider stepping in the $\hat{y}$ direction. We again see that every second link in that direction can be simultaneously updated. By symmetry the same must also be true for the $\hat{z}$ and $\hat{t}$ directions as depicted in Fig. 2.3, where we have used a broken dash-dot line to try to indicate the fourth dimension (i.e., for the links that lie in the $\hat{x}$–$\hat{t}$ plane). We see that for the link pointing in the $\hat{x}$ direction, the plaquettes (and staples) in the $\hat{x}$–$\hat{y}$, $\hat{x}$–$\hat{z}$, $\hat{x}$–$\hat{t}$ planes are all related by simple rotations about the link. Thus we see that we have now built up a four-dimensional mask for determining which links pointing in the $\hat{x}$ direction can be simultaneously updated.

Let us introduce some convenient shorthand notation. If for a given link pointing in the direction μ, we must take n steps in the direction ν before reaching the next updatable link pointing in the direction μ, we will use the notation $\mu : \nu \sim n\nu$. For our checkerboard masking we see that for a link pointing in the direction $\hat{x}$ we have to take two steps in each of the Cartesian directions before reaching the next updatable link. Hence we write

$$\hat{x}: \quad \hat{x} \sim 2\hat{x} \;\; , \quad \hat{y} \sim 2\hat{y} \;\; , \quad \hat{z} \sim 2\hat{z} \quad \text{and} \quad \hat{t} \sim 2\hat{t}. \tag{4.18}$$

We immediately see that this is also true for links oriented in the $\hat{y}$, $\hat{z}$, and $\hat{t}$ directions so that

$$\begin{aligned}
\hat{y}: &\quad \hat{x} \sim 2\hat{x} \;\; , \quad \hat{y} \sim 2\hat{y} \;\; , \quad \hat{z} \sim 2\hat{z} \quad \text{and} \quad \hat{t} \sim 2\hat{t}, && (4.19)\\
\hat{z}: &\quad \hat{x} \sim 2\hat{x} \;\; , \quad \hat{y} \sim 2\hat{y} \;\; , \quad \hat{z} \sim 2\hat{z} \quad \text{and} \quad \hat{t} \sim 2\hat{t}, && (4.20)\\
\hat{t}: &\quad \hat{x} \sim 2\hat{x} \;\; , \quad \hat{y} \sim 2\hat{y} \;\; , \quad \hat{z} \sim 2\hat{z} \quad \text{and} \quad \hat{t} \sim 2\hat{t}. && (4.21)
\end{aligned}$$

Finally, note that when we wish to update all of the links pointing in any one of the four Cartesian directions, say μ, we need only two four-dimensional masks. This is because exactly half of the μ-oriented links across the entire lattice are considered in each four-dimensional mask. To appreciate this we simply note that for any one of the Cartesian directions one mask can be turned into the checkerboard complement mask for that direction by shifting the mask by one step in any Cartesian direction, (see Fig. 4.1). So to update all of the links on the lattice we need a total of 8 four-dimensional masks, i.e., 2 masks for each of the four Cartesian directions. In other words, no matter how many nodes we have available on our parallel computing architecture a full lattice updating sweep will require 8 serial masked sweeps to complete with a nearest neighbour action (such as the Wilson action) and checkerboard masking. This is the conventional procedure for the standard Wilson action in lattice QCD studies. In closing this section on the standard Wilson action we observe that there is an alternative and equally good "linear" masking for this case.

Linear Masking

As an alternative approach to the checker board masking described in Sec. 4.2.2, one could partition the links over the lattice in a linear fashion as shown in Fig. 4.2. If the link variable of interest is $U_{\hat{x}}(x)$ then the next possible link variable in the $\hat{x}$ direction

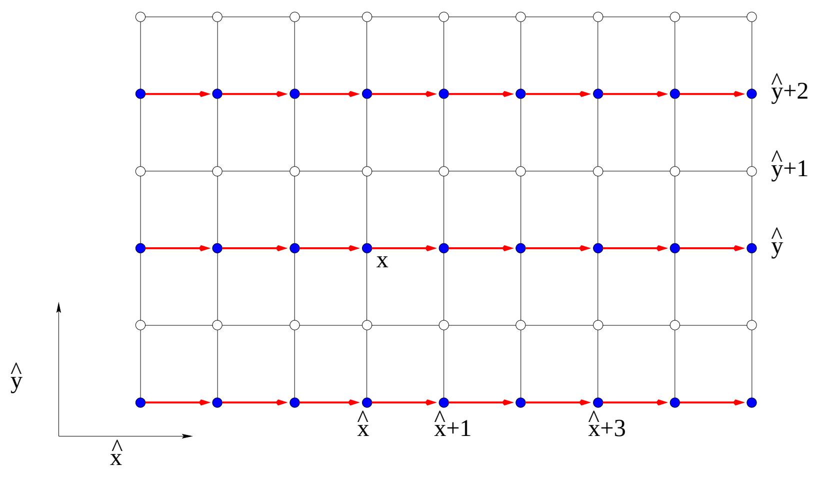


Figure 4.2: Linear masking of the lattice when using standard Wilson action. The highlighted arrows represents the link variable that can be updated simultaneously.

which can be updated is the $U_{\hat{x}}(x+\hat{x})$ link and then the $U_{\hat{x}}(x+2\hat{x})$ and so on. We see that all the links on the $\hat{x}$ line can be updated at the same time, since none of these links are contained in the 1×1 plaquettes for the other links in the line. Hence we have $\hat{x}:\hat{x}\sim 1\hat{x}$. Now looking in the $\hat{y}$ direction, we realize that we cannot touch the $U_{\hat{x}}(x+\hat{y})$ link because it is part of the Wilson loop containing the link variable $U_{\hat{x}}(x)$ which is being updated simultaneously. However, the links $U_{\hat{x}}(x+2\hat{y})$, $U_{\hat{x}}(x+4\hat{y})$, etc. can be updated. Consequently, we have $\hat{x}:\hat{y}\sim 2\hat{y}$ and similarly for steps in the $\hat{z}$ and $\hat{t}$ directions. For a link variable pointing in the $\hat{x}$ direction we then have that

$$\hat{x}: \quad \hat{x}\sim 1\hat{x} \quad , \quad \hat{y}\sim 2\hat{y} \quad , \quad \hat{z}\sim 2\hat{z} \quad \text{and} \quad \hat{t}\sim 2\hat{t}. \tag{4.22}$$

When the links to be updated are pointing in the other three directions we have

$$\hat{y}: \quad \hat{x}\sim 2\hat{x} \quad , \quad \hat{y}\sim 1\hat{y} \quad , \quad \hat{z}\sim 2\hat{z} \quad \text{and} \quad \hat{t}\sim 2\hat{t}, \tag{4.23}$$
$$\hat{z}: \quad \hat{x}\sim 2\hat{x} \quad , \quad \hat{y}\sim 2\hat{y} \quad , \quad \hat{z}\sim 1\hat{z} \quad \text{and} \quad \hat{t}\sim 2\hat{t}, \tag{4.24}$$
$$\hat{t}: \quad \hat{x}\sim 2\hat{x} \quad , \quad \hat{y}\sim 2\hat{y} \quad , \quad \hat{z}\sim 2\hat{z} \quad \text{and} \quad \hat{t}\sim 1\hat{t}, \tag{4.25}$$

for the $\hat{y}$, $\hat{z}$ and $\hat{t}$ directions respectively.

Again, we see that there are two complementary linear masks for links pointing in any given Cartesian direction μ. One mask can be obtained from the other by a shift of one step in any of the three Cartesian directions orthogonal to μ as can be appreciated from Fig. 4.2. Thus this linear masking is equally as efficient as the checkerboard masking of the previous section, since there are 2 masks for each of the 4 Cartesian directions giving a total of 8 masks.

The computer code implementing the above method may be found in Appendix E.1. Both the linear and checker board are implemented. During our production runs, the checker board was used.

The mask is setup as a six dimensional array. The first four dimensions are just space–time. The fifth dimension is the Lorentz index, that is the direction in which the link is pointing towards. Finally the sixth dimension is the number of masks. The variable *nmask* was set to sixteen so that it could be accommodated with the rest of the program. This routine gets only called once during the execution of the program.

4.2.3 The Program for the Pseudoheat Bath Algorithm

The routine that implements the pseudoheat bath algorithm is called *pseudoheat.fcm*. This routine can be found in Appendix E.2.

The algorithm tells us that only a_0 undergoes an accept/reject step, and that the rest of the 4–vector, a_μ, can therefore be randomly generated on $S^3 \in \mathbf{R}^4$. The routine commences by generating a uniformly distributed array of random numbers. This was done using the method [19] described in Appendix D.1.

The next step is to calculate the determinant $k = |\sum_{\alpha=1}^{6} \widetilde{U}_\alpha|^{1/2}$ where $\widetilde{U}_\alpha \equiv \{$ the product of the three links that interact with the link in question, and there are six planes$\}$, i.e., staples

Now the task is to update all the links in the lattice according to the mask. To ensure that the mask does not get corrupted during the procedure, it is copied into a temporary variable called *update*. The values of update are all true when $imask = 1$ and all false when $imask = 2$. The value of *imask* is passed into the argument list.

The accept/reject step is performed with a *do while*. The *do while* makes sure that all links are updated at every sweep. A space–time array of random double precision numbers dimensioned according to the volume of the lattice is stored in the variable x. The variable x is uniformly distributed within the region $\exp(-2\beta k) < x < 1$. This generates a_0 which is distributed with exponential weight $\exp(\beta k a_0)$.

To summarize, the algorithm begins with a trial $a_0 = 1 + ln(x)/(\beta k)$ where x is a random number uniformly distributed in the region $\exp(-2\beta k) < x < 1$. To correct for the factor $(1 - a_0^2)^{1/2}$ in the probability function $P(a_0) = (1 - a_0^2)^{1/2} \exp(\beta k a_0)$, reject this a_0 with probability $P_{\text{reject}}(a_0) = 1 - \sqrt{(1 - a_0^2)}$ (called *probrjk*) and select a new trial a_0. Repeat this until an a_0 is accepted. This a_0 is accepted by following the procedure.

The variable *update* is initialized with the same value as the *mask* variable. If $P_{\text{reject}}(a_0) = 1 - \sqrt{(1 - x^2)}$ is true, i.e. if $probrjk < y$ (y a random number $0 \leqslant y \leqslant 1$), and if $update = .true.$, which is true, then where ever this condition is true within the lattice then $a_0 = x$. Otherwise, we have a true with a false which is false which implies that a_0 is not updated. In that case $update = update\ .and.\ probrjk \geqslant random$ (that is $update \longrightarrow .not.\ update$) and we go around the loop to repeat this process until all the links are updated.

Here is the section of code that performs the accept/reject procedure:

```
do while( any(update) )
   counter = counter + 1
   call CMF_random( x )
   x = ( 1.0d0 - exp( -2.0d0 * beta * k ) ) * x + exp( -2.0d0 * beta * k )
   x = 1.0d0 + log( x ) / ( beta * k )
   probrjk = 1.0d0 - sqrt( 1.0d0 - x**2 )
   call CMF_random( random )
   where( update .and. probrjk < random ) a4vector(:,:,:,:,4) = x(:,:,:,:)
   update = ( update .and. probrjk >= random )
end do
```

This iterative procedure is dependent on the lattice spacing which means that as β increases the probability $P(a_0)$ decreases, which implies that $P_{\text{reject}}(a_0)$ increases. Therefore more iterations around the *do while* loop are required.

Once we have all of the a_μ, it is then possible to calculate the link variable $U_\mu(x) = a_0 I + i\vec{a}\cdot\vec{\sigma}$ in a particular direction. Using this randomly generated link variable, the existing link $U_\mu(x)$ is replaced with $U_\mu(x) \longrightarrow U' = U_\mu(x)\left(\sum_{\alpha=1}^{6} \widetilde{U_\alpha}\right)^\dagger / k$, as in Eq. (4.12).

4.2.4 The Staples with the C-shifts

The staple is constructed by taking the product of the gauge links around a closed contour. The multiplication is performed at the level of the gauge group for all space–time sites. For this program the contour was the standard Wilson loop namely a 1×1, see Fig. 2.3. I shall postpone this discussion until I come to discuss improved actions in Chapter 5.2.1.

4.3 Numerical Results for $SU_c(2)$

In Fig. 4.3 and Fig. 4.4, I illustrate the convergence as a function of the iteration process using two different initial states for two different Monte Carlo procedures: the Pseudo–heatbath algorithm [20] and the Metropolis algorithm [17]. The first state is called an ordered state, that is, where all the links are set to the identity ($U_\mu(x) = I$). The second one is called a random state, where a set of purely random $SU_c(2)$ matrices is attached at each lattice site (see Appendix E.3). Working in various β values, I plot the average action per plaquette:

$$\overline{P} = < S_\Box >, \tag{4.26}$$

for an $8 \times 8 \times 8 \times 8$ lattice. The size of the lattice is totally arbitrary. In fact, an 8^4 lattice is sufficient to get an overall view of the general behaviour.

In the top half of Fig. 4.3 we see that for the Pseudo–heatbath the points have disappeared from the graph after the tenth iteration. This is because at that point the 2×2 matrices are no longer part of the $SU_c(2)$ gauge group. This problem does not arise with the Metropolis algorithm. The Metropolis algorithm does not perform any accept/reject on any of the coefficients, a_μ, in the link variable, $U_\mu(x)$ Eq. (4.3).

In order to fix this problem the generated matrices need to be projected back onto the $SU_c(2)$ gauge group or reuniterized.

4.3.1 Reuniterizing the Links Every n Sweeps

The reunitarization of the $SU_c(2)$ matrix is done using the standard row by row orthonormalization procedure. The procedure begins by normalizing the first row of the matrix; then updates the second row by calculating $row2 = row2 - (row2 \cdot row1) row1$; finally the second row is normalized.

In the top half of Fig. 4.4, we can observe both the evolution of an ordered and a random initial state, when the links are reuniterized with every sweep. We can see the system evolves towards its equilibrium.

From the bottom half of the Fig. 4.4 one can observe that it takes longer for the Metropolis algorithm to converge for a high β value. In fact as shown in Fig 4.4 we can see that the Metropolis algorithm is ten times slower than the pseudo–heatbath method.

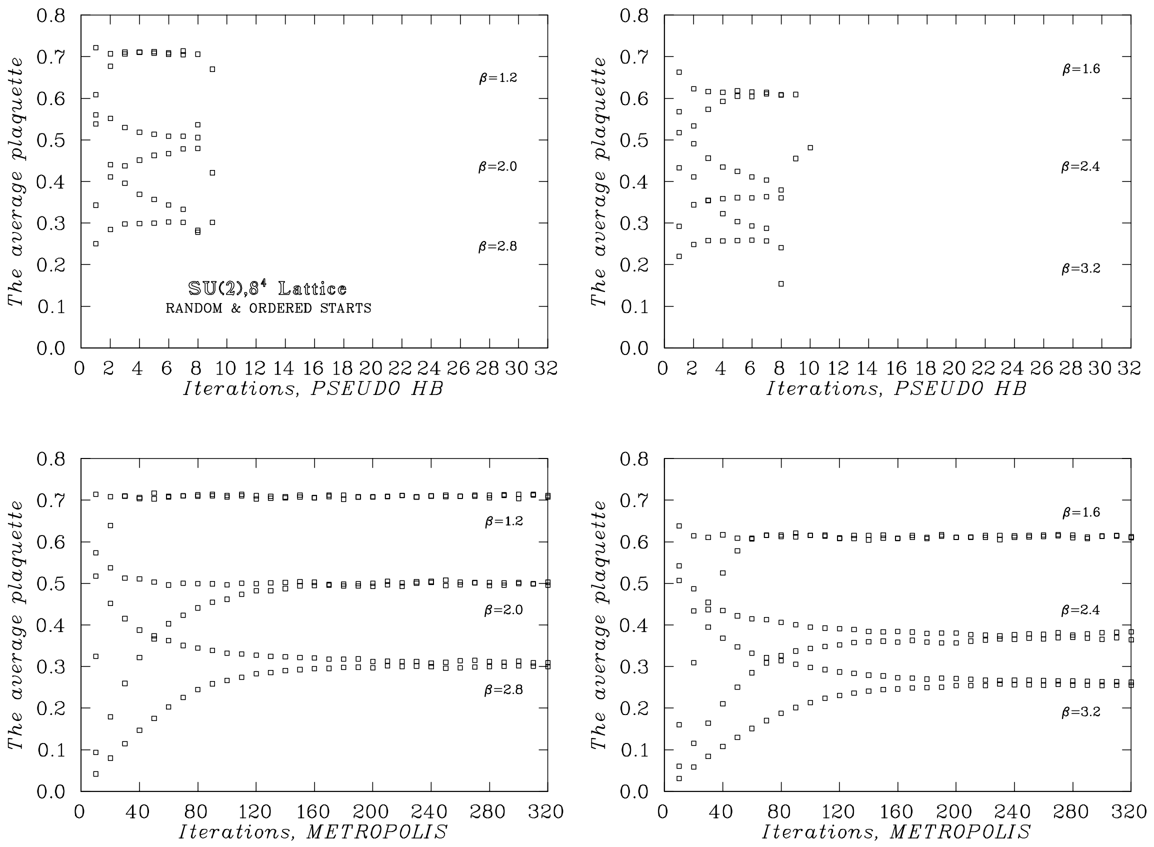


Figure 4.3: $SU_c(2)$ Gauge Configuration at different β. Pseudo-heatbath algorithm versus Metropolis algorithm without reunitarizing the links. Note that the Metropolis algorithm is ten times slower than the Pseudo–heatbath method.

High β means that we are in the weak regime where the typical quantum fluctuations are much smaller than in the strong regime (low β) where the fluctuations are large. This means that for high β the Metropolis algorithm spends a great deal of time generating unrealistic $SU_c(2)$ group elements. This effect is expected to be largely amplified as the number of colours, N_c, is increased. Although the Metropolis algorithm reaches thermalization about ten times slower than the pseudoheat–bath algorithm, it is actually possible to use it to generate gauge field configurations in the $SU_c(2)$ gauge group.

These results were obtained on both the $CM5$ and on an Alpha Workstation. It takes about $10sec$ of CPU time per full sweep on the Alpha station, while it takes about $1sec$ to $1.5sec$ on the $CM5$ with 64–Nodes for the Pseudo-heat bath on this small lattice. The length of time taken by the Metropolis algorithm for each iteration is approximately the same as that taken by the Pseudoheat–bath. This is because there are approximately the same amount of calculations for each full sweep.

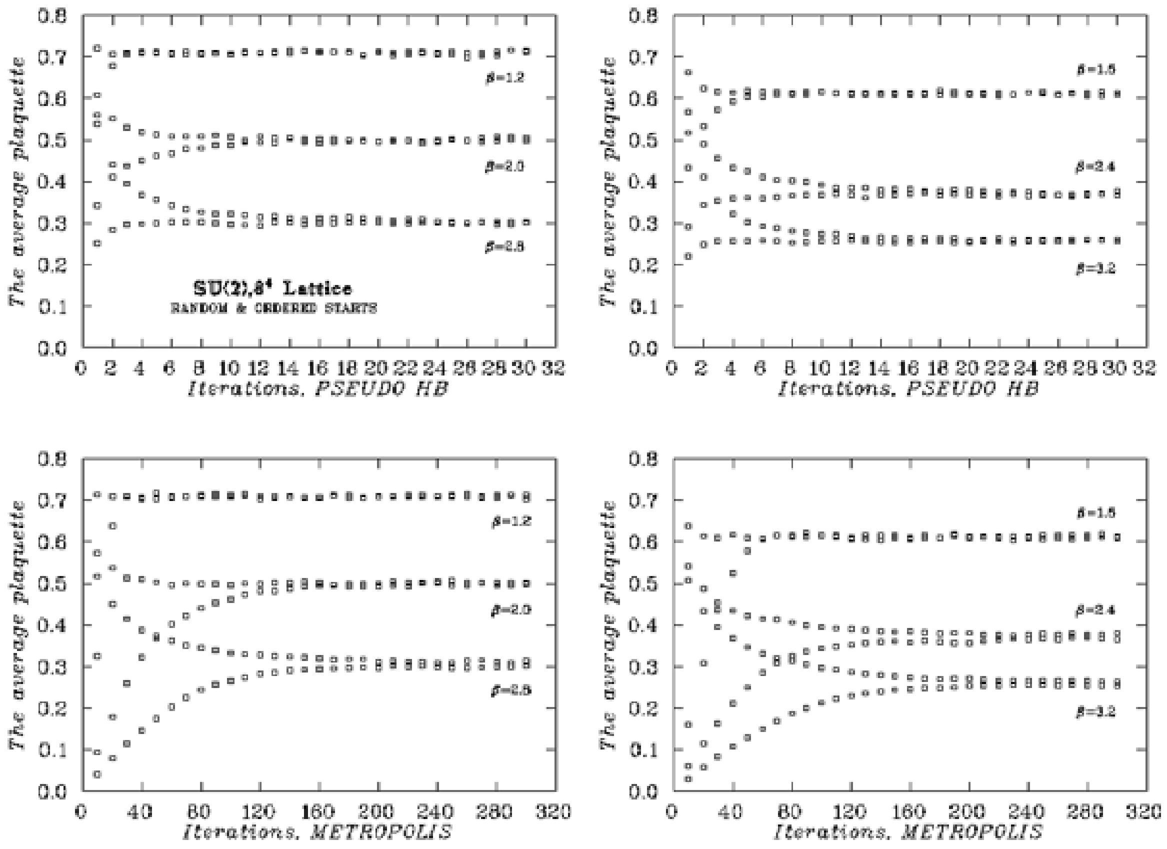


Figure 4.4: $SU_c(2)$ Gauge Configuration at different β. Pseudo-heatbath algorithm versus Metropolis algorithm with a reunitarization scheme. There we can see that the Metropolis algorithm is slower than the Pseudo–heatbath method by a factor of ten.

4.4 The Algorithm for the $SU_c(3)$ gauge group.

In this section a method for updating $SU_c(3)$ matrices is described. This algorithm is an extension of the Creutz method [20], mentioned in the previous Sec. 4.1, for updating the $SU_c(2)$ matrices, and could be applied with small programming effort to any number of colours [21]. It is composed of the following steps

1. Select a sufficiently large subset F of $SU_c(2)$ subgroups of $SU_c(N)$:

   $$F = \{SU_c(2)_k, k = 1, .., m = N - 1\} \tag{4.27}$$

   such that there is no left ideal, i.e. no subset of $SU_c(N)$ which is invariant under left multiplication by F except the unit element and the whole group. In the case of QCD, where only three colours are necessary, one could choose for the set F a matrix of the form:

   $$a_1 = \begin{pmatrix} \alpha_1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad and \qquad a_2 = \begin{pmatrix} 1 & 0 \\ 0 & \alpha_2 \end{pmatrix}, \tag{4.28}$$

   where $\alpha_k \in SU_c(2)$ located at the k^{th} and the $(k+1)^{th}$ rows and columns.

2. In each step of the iteration the new link variable is obtained by multiplying the previous value by $m = N - 1$ matrices (here $N = 3$ then $m = 2$) belonging to the m subgroups $SU_c(2)_k$

   $$U' = a_2 a_1 U, \qquad \text{with} \qquad a_k \in SU_c(2)_{k=1,2}, \tag{4.29}$$

   so that

   $$U' = U^{(m)}, \qquad \text{with} \qquad U^{(k)} = a_k U^{(k-1)} = a_k a_{k-1} .. a_1 U^{(0)}. \tag{4.30}$$

   When $k = 0$, the link variable is just the initial link, i.e. $U^{(0)} = U$.

The element a_k is chosen at random (a_1 is chosen first, then a_2) with the measure:

$$dP(a_k) = d^{(k)} a_k \frac{\exp\left(-\beta S[a_k U^{(k-1)}]\right)}{Z_k[U^{(k-1)}]}, \tag{4.31}$$

where $d^{(k)}a_k$ is the Haar measure on $SU_c(2)_k$, and the normalization factor $Z_k[U]$ is defined by:

$$Z_k[U] = \int_{SU_c(2)_k} da \exp(-\beta S[aU]). \tag{4.32}$$

Note that $Z_k[U]$ is invariant under left multiplication by an element of $SU_c(2)_k$, mathematically this means that

$$Z_k[bU] = Z_k[U], \qquad \text{if} \qquad b \in SU_c(2)_k. \tag{4.33}$$

Performing the generation of a_k consecutively for $k = 1, 2, .., m$, one obtains U' according to Eq.(4.30). N. Cabibbo and E. Marinari [21], have proved, using the invariance properties of the Haar measure under left multiplication and inversion, that the procedure leads to thermalization provided $U^{(k-1)}$ has a Boltzmann distribution, something that guarantees detailed balance. We can see that it is indeed the case, if we assume that $U^{(k-1)}$ has a Boltzmann distribution:

$$dP(U^{(k-1)}) = \frac{\exp\left(-\beta S[U^{(k-1)}]\right)}{Z_N} dU^{(k-1)}, \tag{4.34}$$

with Z_N a normalisation factor. Now since $U^{(k)} = a_k U^{(k-1)}$, it implies that $U^{(k-1)} = a_k^{-1} U^{(k)}$. Therefore, the probability density for $U^{(k-1)}$, Eq.(4.34), becomes

$$dP(U^{(k-1)}) = dP(a_k^{-1}U^{(k)}) = \frac{\exp\left(-\beta S[a_k^{-1}U^{(k)}]\right)}{Z_N} dU^{(k-1)}. \tag{4.35}$$

In this way, we can obtain a probability distribution function for the $U^{(k)}$. First we note that we have,

$$dP(U^{(k)}) = dP(a_k) dP(U^{(k-1)}). \tag{4.36}$$

Using Eq.(4.31) and Eq.(4.34), the probability distribution for $U^{(k)}$ can then be written as:

$$dP(U^{(k)}) = \int_{SU_c(2)_k} d^{(k)}a \frac{\exp\left(-\beta S[U^{(k)}]\right)}{Z_k[a^{-1}U^{(k)}]} \frac{\exp\left(-\beta S[a^{-1}U^{(k)}]\right)}{Z_N} d(a^{-1}U^{(k)}). \tag{4.37}$$

From Eq.(4.33) and the properties of the Haar measure:

$$d(a^{-1}U^{(k)}) = dU^{(k)}, \qquad \text{and} \qquad d^{(k)}(a) = d^{(k)}(a^{-1}), \tag{4.38}$$

it is easily seen that Eq.(4.37) coincides with the Boltzmann measure. So if $U^{(k)}$ has a Boltzmann distribution, so will U'. If the action S is identified with the Wilson action we can write

$$S[U] = \sum_{\Box} \Re e Tr(U\widetilde{U}_\Box) + \text{constant} = \Re e Tr(UR) \quad + \quad \text{constant}, \tag{4.39}$$

where $R = \sum_\Box \widetilde{U}_\Box$ arises from the staples depicted in Fig. 2.3. In order to generate the distribution in Eq.(4.31) for the minimal set F, Eq.(4.27), we note that

$$S[a_k U] = \Re e Tr(a_k U^{(k-1)} R) = \Re e Tr(\alpha_k r_k) \quad + \quad \text{terms independent of } \alpha_k. \tag{4.40}$$

Here the matrix r_k is the 2×2 submatrix of the link time staples matrix, $U^{(k-1)}R$. This submatrix must have exactly the same block structure as the matrix a_k.

We can write:

$$r_k = r_0 I + i\vec{r} \cdot \vec{\sigma}, \qquad \text{and} \qquad \alpha_k = \alpha_0 I + i\vec{\alpha} \cdot \vec{\sigma}, \tag{4.41}$$

so that

$$\Re e Tr(\alpha_k r_k) = 2(\alpha_0 \Re e(r_0) - \vec{\alpha} \cdot \Re e(\vec{r})). \tag{4.42}$$

Since $\alpha_\mu \in \mathbf{R}$, the real part of r_0 and $\vec{r}$ participate in the action. The problem reduces to the need to generate random $SU_c(2)$ submatrices α_k with a probability distribution given by:

$$dP(\alpha_k) = \delta(\alpha^2 - 1) \exp\left(-\frac{\beta}{N} \Re e Tr(\alpha_k r_k)\right) d^4\alpha_k. \tag{4.43}$$

These are generated using the algorithm described in Sec. 4.1.

To summarize the algorithm as it is coded (see Appendix E.9):

1. First calculate the staples with some arbitrary links (ordered or random start) in a particular direction, R.

2. Calculate the link times the staples, $U^{(0)}R$.

3. Extract the appropriate 2×2 submatrix from $U^{(0)}R$ and reform it as a $SU_c(2)$ matrix with $r_k = r_0 I + i\vec{r} \cdot \vec{\sigma}$, with $\Re e(r_\mu)$.

4. Pass this information into the Pseudo-heatbath algorithm to obtain some new $SU_c(2)$ matrix, the α_1.

5. Calculate $U^{(1)} = a_1 U^{(0)}$, where a_1 is as in Eq.(4.28).

6. Repeat step.1 to 5 to calculate the new link, $U^{(2)} = a_2 U^{(1)}$.

4.4.1 The Number of $SU_c(2)$ subgroups.

The algorithm suggests that a minimum of two $SU_c(2)$ subgroups is required to cover the $SU_c(3)$ gauge group. The positions of the entries for these two $SU_c(2)$ subgroups are displayed in the following matrices:

$$a_1 = \begin{pmatrix} \alpha_1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad \text{and} \qquad a_2 = \begin{pmatrix} 1 & 0 \\ 0 & \alpha_2 \end{pmatrix}. \tag{4.44}$$

The subgroups $\alpha_1, \alpha_2 \subseteq SU_c(2)$, are generated using the pseudoheat–bath algorithm described in Section 4.1.

Using cooling methods (the methods are described in subsequent chapters. In short, the method is designed to remove the short range quantum fluctuation from the gauge field) it was possible to demonstrate that a gauge field constructed from two $SU_c(2)$ subgroups had a rough nature as opposed to having three diagonal $SU_c(2)$ subgroups. Consequently, the $SU_c(3)$ gauge field configurations were constructed using the following structure:

$$a_1 = \begin{pmatrix} \alpha_1 & 0 \\ 0 & 1 \end{pmatrix}, a_2 = \begin{pmatrix} 1 & 0 \\ 0 & \alpha_2 \end{pmatrix}, \qquad \text{and} \qquad a_3 = \begin{pmatrix} \alpha_{11} & 0 & \alpha_{12} \\ 0 & 1 & 0 \\ \alpha_{21} & 0 & \alpha_{22} \end{pmatrix}. \tag{4.45}$$

To further smooth out the gauge field this structure was looped over twice, producing a gauge field constructed from six $SU_c(2)$ subgroups. See Appendix E.9.

Chapter 5

Improved Actions in the $SU_c(3)$ Gauge Group

In the previous section I described the pseudoheat–bath algorithm in the $SU_c(2)$ gauge group (Section 4.1) and how one can extend the algorithm to $SU_c(N)$ (Section 4.4). These methods were basically applied to the case of a simple 1×1 Wilson loop type action. I now turn to the case of improved actions where additional Wilson loops are added to construct an action which has smaller discretization errors. The main motivation behind using an improved action is that these types of action make it possible to run simulation at a coarser lattice spacing, hence larger volumes, while keeping the discretization errors well under control.

Over recent years computer power has tremendously increase therefore giving the possibility to study larger lattice with finer spacing. Not only these kind of simulations are very costly with respect to time, but also traditional perturbation theory for lattice QCD starts to fail at distances of order 0.1 fm. The idea was to do a perturbative expansion of the short–distance lattice quantities in terms of the bare coupling α_{lat} used in the lattice Lagrangian. This idea was motivated from the fact that the bare coupling in a cutoff theory is approximately equal to the running coupling evaluated at the cutoff scale, $\alpha_s(\pi/a)$. This means, in this case, that α_{lat} would be the appropriate coupling for the quantities dominated by the momenta near the cutoff. The problem with this idea is that the bare coupling in traditional lattice QCD is much smaller than an effective coupling at large lattice spacing namely:

$$\alpha_{lat} = \alpha_V(\frac{\pi}{a}) - 4.7\alpha_V^2 + ... \leq \frac{1}{2}\alpha_V(\frac{\pi}{a}), \qquad \text{for} \qquad a \geq 0.1\,\text{fm}, \tag{5.1}$$

where α_V is an effective coupling defined by the static–quark potential,

$$V_{\overline{Q}Q}(q) \equiv -4\pi C_F \frac{\alpha_V(q)}{q^2}. \tag{5.2}$$

Consequently the α_{lat} expansion, does not take into account the perturbative effects correctly and converges poorly [11].

The breakthrough, came in the early 1990's with the discovery of a trivial modification called "Tadpole Improvement". Tadpole contributions arise from higher powers of $agA_\mu(x)$, in the link operator expansion $U_\mu(x) \approx e^{-iagA_\mu(x)} = 1 - iagA_\mu(x) + \mathcal{O}(a^2)$, creating extra vertices in the gluon action, as well as generating divergent factors of $\mathcal{O}(g^2)$. Consequently the contribution generated by these extra vertices are suppressed by powers of g^2 and not a, this effect turns out to be significantly large. So the best way to eliminate these unwanted tadpole contribution is to find a quantity that can be computed during the simulation time, which is mostly made up of tadpole. The mean link $u_0 = (1/3 < \Re e\,\mathrm{Tr} U_{\mu\nu}(x) >)^{1/4}$, where $U_{\mu\nu}(x)$ is the 1×1 plaquette. To eliminate these unwanted contributions divide every lattice link by the mean link:

$$U_\mu(x) \longrightarrow \frac{U_\mu(x)}{u_0}, \qquad \text{such that} \qquad U_\mu(x) \longrightarrow u_0(1 - iagA_\mu(x)). \tag{5.3}$$

By making this simple change, the tadpole contributions are largely accounted for and perturbation theory converges more rapidly.

In the gluonic action the tadpole improved bare coupling, α_{TI} is enhanced by a factor of $1/u_0^4$ relative to the α_{lat} in the unimproved theory:

$$\alpha_{TI} = \frac{\alpha_{lat}}{u_0^4}. \tag{5.4}$$

If one expresses α_{TI} in terms of the continuum coupling α_V, we find that the tadpole improved bare coupling is approximately equal to the continuum coupling $\alpha_V(\pi/a)$:

$$\alpha_{TI} = \alpha_V(\frac{\pi}{a}) - 0.5\alpha_V^2 + ... \approx \alpha_V(\frac{\pi}{a}). \tag{5.5}$$

5.1 Gauge Action

The starting point in constructing an improved lattice gauge action is the same as in the case of the standard Wilson gauge action, Eq. (2.12), in Section 2.1.3. Recall that the QCD gauge action in continuum Euclidean space–time to be given by

$$S_G = \frac{1}{4}\int d^4x F_{\mu\nu}(x)F_{\mu\nu}(x). \tag{5.6}$$

Where $F_{\mu\nu}(x)$ is the usual non–Abelian gauge invariant field strength tensor defined in Eq. (2.13).

The Gauge action is constructed from the path ordered variable, around some arbitrary contour $\mathcal{C}_{\mu\nu}(x)$, Eq. (2.15), where the contour integral is evaluated using Stoke's Theorem as in Eq. (2.16). The boundaries of the contour integral are determined by the size of the Wilson loop. The expansion point x_0 is located at the centre of the Wilson loop such that if one is considering the 1×1 plaquette, for both x_μ and x_ν, we have $-a/2 \le x_\mu \le a/2$ similarly for $-a/2 \le x_\nu \le a/2$. Now we would like to insert more Wilson loops, namely the 2×1 and 1×2 loops. In this case, the integration bounds are $-a \le x_\mu \le a$ and $-a/2 \le x_\nu \le a/2$ for the 2×1 loop. Similarly for the 1×2 improved plaquette we have $-a/2 \le x_\mu \le a/2$ and $-a \le x_\nu \le a$. The integration bounds are illustrated in Fig. 5.1. So, for the 1×1 Wilson contour and $2\times 1, 1\times 2$ improved contours we have respectively the following:

$$\oint_{1\times 1} A(x)\cdot dx = a^2 F_{\mu\nu}(x_0) + \frac{a^4}{24}\left(D_\mu^2 + D_\nu^2\right) F_{\mu\nu}(x_0) + \mathcal{O}(a^6; A^3), \tag{5.7}$$

$$\oint_{2\times 1} A(x)\cdot dx = 2a^2 F_{\mu\nu}(x_0) + \frac{a^4}{12}\left(4D_\mu^2 + D_\nu^2\right) F_{\mu\nu}(x_0) + \mathcal{O}(a^6; A^3), \tag{5.8}$$

$$\oint_{1\times 2} A(x)\cdot dx = 2a^2 F_{\mu\nu}(x_0) + \frac{a^4}{12}\left(D_\mu^2 + 4D_\nu^2\right) F_{\mu\nu}(x_0) + \mathcal{O}(a^6; A^3)\,. \tag{5.9}$$

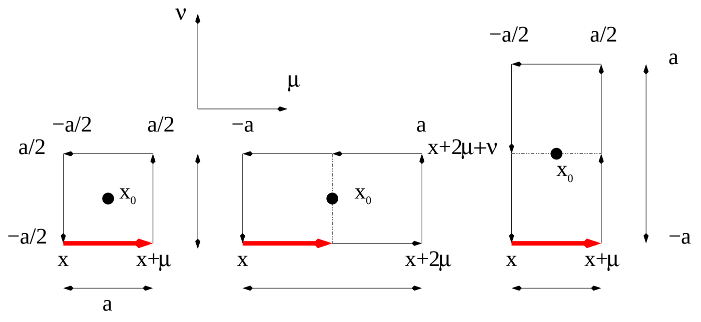


Figure 5.1: The integration contour for the Wilson loops used in a 1×1 contour and $2 \times 1, 1 \times 2$ improved contours. The expansion is done about x_0.

The action is obtained by taking the real part of the trace of Eq. (2.15), hence only terms of $\mathcal{O}(g^2)$ will contribute. We therefore need,

$$\left(\oint_{1\times1} A(x)\cdot dx\right)^2 = a^4 F^2_{\mu\nu}(x_0) + \frac{a^6}{12} F_{\mu\nu}(x_0)\left(D^2_\mu + D^2_\nu\right) F_{\mu\nu}(x_0) + \mathcal{O}(a^8; A^3), \tag{5.10}$$

$$\left(\oint_{2\times1} A(x)\cdot dx\right)^2 = 4a^4 F^2_{\mu\nu}(x_0) + \frac{4a^6}{12} F_{\mu\nu}(x_0)\left(4D^2_\mu + D^2_\nu\right) F_{\mu\nu}(x_0) + \mathcal{O}(a^8; A^3), \tag{5.11}$$

$$\left(\oint_{1\times2} A(x)\cdot dx\right)^2 = 4a^4 F^2_{\mu\nu}(x_0) + \frac{4a^6}{12} F_{\mu\nu}(x_0)\left(D^2_\mu + 4D^2_\nu\right) F_{\mu\nu}(x_0) + \mathcal{O}(a^8; A^3)\,. \tag{5.12}$$

The $\mathcal{O}(a^2)$ errors generated in the Wilson action, Eq. (2.19), are cancelled out by adding other rectangular Wilson loops. These loops are illustrated in Fig. 5.1 and 5.2, and

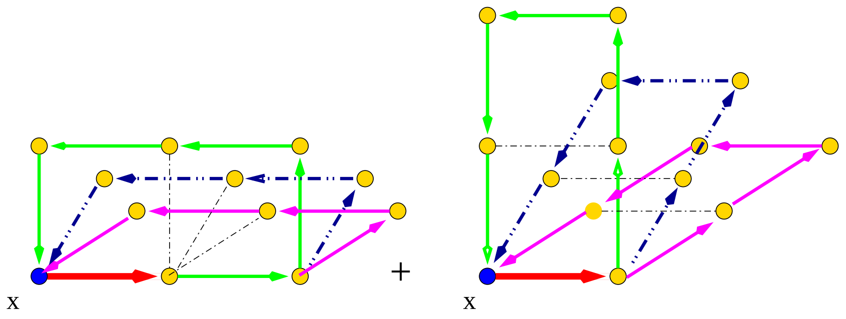


Figure 5.2: The staples containing the six elementary rectangular plaquettes with base at x.

have an expansion expressed by Eq. (5.11) for the 2×1 Wilson loop which we denote by $R_{\mu\nu}(x)$ and Eq. (5.12) for the 1×2 Wilson denoted $R_{\nu\mu}(x)$. These rectangular loops may be combined to obtain a lattice action which is accurate up to $\mathcal{O}(a^4)$, namely

$$\begin{aligned} \frac{1}{2}\left(R_{\mu\nu}(x) + R_{\nu\mu}(x)\right) &= \frac{1}{2!}\left(\left(\oint_{2\times1} A(x)\cdot dx\right)^2 + \left(\oint_{1\times2} A(x)\cdot dx\right)^2\right) \\ &= 4a^4 F^2_{\mu\nu}(x_0) + \frac{20a^6}{24} F_{\mu\nu}(x_0)\left(D^2_\mu + D^2_\nu\right) F_{\mu\nu}(x_0) + \mathcal{O}(a^8; A^3)\,. \end{aligned} \tag{5.13}$$

Using Fig. 5.1, we can write down an expression in terms of link variables which has an expansion of the form of Eq. (5.13):

$$\begin{aligned} U_\mu(x)\widetilde{\Sigma}_{\text{rect}}(x) &= U_\mu(x)U_\nu(x+\hat\mu)U_\nu(x+\hat\nu+\hat\mu)U^\dagger_\mu(x+2\hat\nu)U^\dagger_\nu(x+\hat\nu)U^\dagger_\nu(x) \\ &+ U_\mu(x)U_\mu(x+\hat\mu)U_\nu(x+2\hat\mu)U^\dagger_\mu(x+\hat\mu+\hat\nu)U^\dagger_\mu(x+\hat\nu)U^\dagger_\nu(x) \end{aligned} \tag{5.14}$$

Of course there are no mathematical restriction on the number of Wilson loops that one can assemble together, the problem with this approach is that the computational time required to calculate these loop considerably increases. Now by adding the squares and rectangles in the proper way with the correct factors, one finds that the improved classical lattice action is accurate up to $\mathcal{O}(a^4)$ [22, 23],

$$\begin{aligned} S_G^{\text{Imp}}(x) &= -\beta_0\left(\sum_{x,\mu>\nu}\frac{5}{3}P_{\mu\nu}(x)-\frac{1}{12}(R_{\mu\nu}(x)+R_{\nu\mu}(x))\right) \\ &\xrightarrow{\text{cont}} \int d^4x\sum_{\mu,\nu}\frac{1}{2}\text{Tr}\,F^2_{\mu\nu}(x)+\mathcal{O}(a^4), \end{aligned} \tag{5.15}$$

for $\beta_0 = 6/g^2$. The quantum version of the gluonic action is mainly obtained by adding the tadpole improvement. The tree–level $\mathcal{O}(a^2)$–improved action is then defined as,

$$S_G = \frac{5\beta}{3}\sum_{\text{sq}}\Re e\,\text{Tr}(1-U_\mu(x)\widetilde{\Sigma}_{sq}(x)) - \frac{\beta}{12u_0^2}\sum_{rect}\Re e\,\text{Tr}(1-U_\mu(x)\widetilde{\Sigma}_{\text{rect}}(x)), \tag{5.16}$$

where the operators $\widetilde{\Sigma}_{sq}(x)$ and $\widetilde{\Sigma}_{rect}(x)$ are defined as follows,

$$\begin{aligned} \widetilde{\Sigma}_{\text{sq}}(x) &= U_\nu(x+\hat{\mu})U^\dagger_\mu(x+\hat{\nu})U^\dagger_\nu(x), \quad \text{and} \quad P_{\mu\nu}(x) = U_\mu(x)\widetilde{\Sigma}_{sq}(x), \qquad (5.17) \\ \widetilde{\Sigma}_{\text{rect}}(x) &= U_\nu(x+\hat{\mu})U_\nu(x+\hat{\nu}+\hat{\mu})U^\dagger_\mu(x+2\hat{\nu})U^\dagger_\nu(x+\hat{\nu})U^\dagger_\nu(x) \\ &+ U_\mu(x+\hat{\mu})U_\nu(x+2\hat{\mu})U^\dagger_\mu(x+\hat{\mu}+\hat{\nu})U^\dagger_\mu(x+\hat{\nu})U^\dagger_\nu(x). \qquad (5.18) \end{aligned}$$

u_0 is the tadpole improvement factor that largely corrects for large quantum renormalization of the links. We employ the plaquette measure:

$$u_0 = \left(\frac{1}{3}\Re e\,\text{Tr}\left\langle\widetilde{\Sigma}_{sq}(x)\right\rangle\right)^{\frac{1}{4}}. \tag{5.19}$$

$\mathcal{O}(g^2a^2)$ correction to this action are estimated to be of the order of two to three percent [25]. Note that $\beta_0 = 6/g^2$ differs from the one used in [25, 26, 27]. Multiplication of β in Eq.(5.16) by a factor of 5/3 reproduces their definition. The u_0 factor depends only on the lattice spacing and $u_0\longrightarrow 1$ as $a\longrightarrow 0$.

5.2 Code for the improved action in $SU_c(3)$

5.2.1 Factors for the Rectangles and Staples

The implementation of the loop calculation is done using three distinct routines. The calculation of the staples, Routine. E.6, the 1×1 loops (Routine. E.7) and of the 1×2, 2×1 loops in Routine. E.8. The subroutine Staples acts as an interface routine which decides which routines are required in the calculation. The decision is done according to the action. In the case of the standard Wilson action only the squares will be required. However, in the case of an improved action both the Squares and Rectangles routines are required. The calculation of the plaquettes is done first, and this is then added on to the rectangular contribution with the appropriate coefficients. The idea is to calculate the contributions one plane at a time and then to amalgamate the results into a temporary variable for each Lorentz index.

The Rectangles routine is a continuous repetition of the squares routine. Enlarging the Wilson loop to a bigger mensuration would require a continuous copy paste of the link product as it is done in Squares and Rectangles. The only difficulty is to calculate the correct improvement factors. Rectangles involves the product of five links while Squares only involve those of three links, and is therefore more computationally expansive .

The matrix multiplication is done at the $SU_c(3)$ level, that is, multiplying the 3×3 complex matrices at all lattice sites simultaneously. To optimize the computation, the matrix movement (the cshift operations) of the $SU_c(3)$ matrices is done outside of the colour loops. For example the multiplication of $U_{\hat{y}}(x+\hat{x})U^{\dagger}_{\hat{x}}(x+\hat{y})$ is coded as

```
    tr = 0.0d0
    ti = 0.0d0
    tsr = cshift(ur(:,:,:,:,yhat(i),:,:),dim=xhat,shift=1)
    tsi = cshift(ui(:,:,:,:,yhat(i),:,:),dim=xhat,shift=1)
    usqr = cshift(ur(:,:,:,:,xhat,:,:),dim=yhat(i),shift=1)
    usqi = cshift(ui(:,:,:,:,xhat,:,:),dim=yhat(i),shift=1)
    do ic=1,nc
       do jc=1,nc
          do kc=1,nc
             tr(:,:,:,:,ic,jc) = tr(:,:,:,:,ic,jc)        +
   &           ( tsr(:,:,:,:,ic,kc) * usqr(:,:,:,:,jc,kc) +
   &             tsi(:,:,:,:,ic,kc) * usqi(:,:,:,:,jc,kc) )
             ti(:,:,:,:,ic,jc) = ti(:,:,:,:,ic,jc)        +
   &           ( tsi(:,:,:,:,ic,kc) * usqr(:,:,:,:,jc,kc) -
   &             tsr(:,:,:,:,ic,kc) * usqi(:,:,:,:,jc,kc) )
          end do
       end do
    end do
```

This is the most efficient way of multiplying the $SU_c(3)$ matrices together.

When the local action is calculated at a given link, the six loops are calculated, otherwise only the positive loops are considered.

5.2.2 Masking the Lattice When Using Improved Action

In this section, we describe the necessary masking procedure for a first-level improved action involving 1×1 and 1×2 Wilson loops. In particular, in this section we are describing the masking suitable for the improved gauge action of Eq. (5.16), which has been used extensively by us [53, 63, 28, 29, 30]. Let us again begin by considering the link variable beginning at some lattice site x and pointing in the $\hat{x}$ direction, i.e., $U_{\hat{x}}(x)$. We now need to consider both Fig. 7.26 for the elementary 1×1 square plaquette and Fig. 5.3 for the 1×2 rectangular plaquette. In Fig. 5.3 we show all of the 1×2 rectangular plaquettes which contain the link $U_{\hat{x}}(x)$, which is shown as the highlighted horizontal link in the three parts of this figure. Depicting a four dimensional object on a flat piece of paper is an artistic challenge and so we have again used a dash–dot line to indicate links lying in the $\hat{x}$–$\hat{t}$ plane. There are three distinguishable ways to include this link in a 1×2 plaquette (the three parts of the figure) and for each of these there are two (mirror-image) rectangles per Cartesian plane and four Cartesian planes.

All links in Figs. 2.3 and 5.3 with arrows (other than the link $U_{\hat{x}}(x)$ itself) must be omitted from the mask when updating this link with our improved action. We see that there are many excluded links.

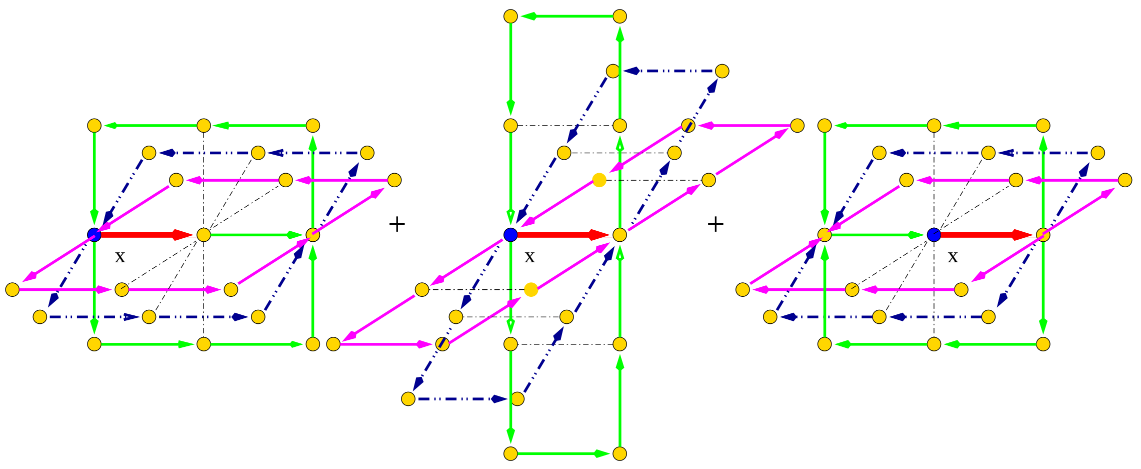


Figure 5.3: The set of all possible 1×2 plaquettes containing the link $U_{\hat{x}}(x)$. The dashed-dotted line is to be understood as being in the $\hat{x}$–$\hat{t}$ plane.

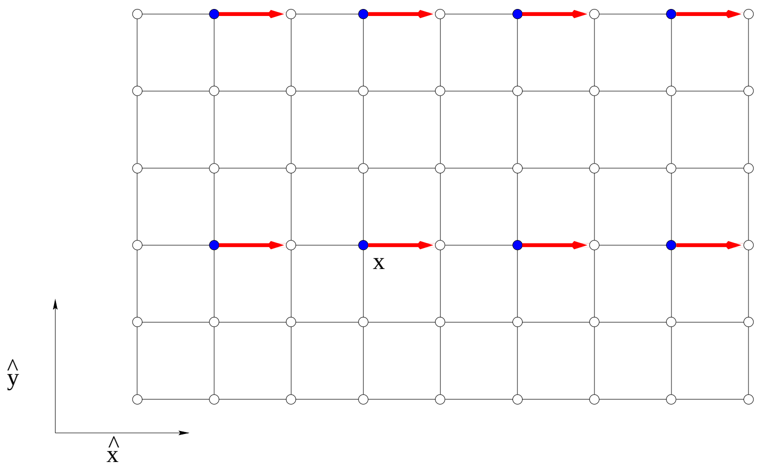


Figure 5.4: The highlighted links with arrows are the ones that can be simultaneously updated for an action containing both 1×1 and 1×2 plaquettes.

In Fig. 5.4 we show which links can be simultaneously updated with the link $U_{\hat{x}}(x)$. We can immediately write down by inspection from this figure that

$$\hat{x}: \quad \hat{x} \sim 2\hat{x} \;, \quad \hat{y} \sim 3\hat{y} \;, \quad \hat{z} \sim 3\hat{z} \quad \text{and} \quad \hat{t} \sim 3\hat{t}\,. \tag{5.20}$$

This follows since the $\hat{z}$ and $\hat{t}$ cases are identical to the $\hat{y}$ case for this $\hat{x}$–oriented link. The generalization to the other orientations of the links to be updated is straightforward by symmetry

$$\hat{y}: \quad \hat{x} \sim 3\hat{x} \;, \quad \hat{y} \sim 2\hat{y} \;, \quad \hat{z} \sim 3\hat{z} \quad \text{and} \quad \hat{t} \sim 3\hat{t}, \tag{5.21}$$

$$\hat{z}: \quad \hat{x} \sim 3\hat{x} \;, \quad \hat{y} \sim 3\hat{y} \;, \quad \hat{z} \sim 2\hat{z} \quad \text{and} \quad \hat{t} \sim 3\hat{t}, \tag{5.22}$$

$$\hat{t}: \quad \hat{x} \sim 3\hat{x} \;, \quad \hat{y} \sim 3\hat{y} \;, \quad \hat{z} \sim 3\hat{z} \quad \text{and} \quad \hat{t} \sim 2\hat{t}\,. \tag{5.23}$$

Let us return to the particular case of the masking for $\hat{x}$-oriented links. From Eq. (5.20) we see that there is symmetry between the $\hat{y}$, $\hat{z}$, and $\hat{t}$ directions and so we will begin by constructing suitable masks for any given equal-x hyper-plane, i.e., for the three dimensional space spanned by the unit vectors $\hat{y}$, $\hat{z}$, and $\hat{t}$.

Before attempting this, let us first consider Fig. 5.4 and extend this to three dimensions by imagining that the $\hat{z}$-axis is pointing directly out from the page. We shall temporarily neglect the $\hat{t}$ direction, which is equivalent to simply taking a slice of the four-dimensional lattice with the same value of t, (i.e., an equal-t hyper-plane). Now let us view this three-dimensional lattice by looking along the $\hat{x}$-axis at one particular equal-x plane. We will then be presented with end-views of updatable links in the $\hat{y}$–$\hat{z}$ plane. For every fixed value of z there are three different masks needed for y and vice versa. Also, there is no restriction on simultaneously updating *diagonally* shifted links, since we are only considering planar actions at this point.

Let's suppose now that we rotate the set of axes such that the entire $\hat{x} \in [1, nx]$ line is just a point going into and out of the page such as in Fig.(5.5). Doing this,

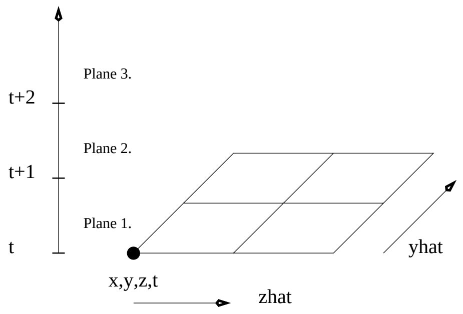


Figure 5.5: Rotating the two dimensional plane onto the $\hat{x}$ axis to give visual access to the other dimensions.

enables one to visually access the hidden fourth dimension, and see which link that can be updated. Since $\hat{y} \sim 3\hat{y}$, $\hat{z} \sim 3\hat{z}$ and $\hat{t} \sim 3\hat{t}$, this implies that the step number n equals to three, we therefore have three different planes, starting at $\hat{t}$ and finishing at $\hat{t}+2$, with three distinct points in each plane. Each filled black circle corresponds to a link that can be updated simultaneously. From Fig.(5.5), we see that any point along just the $\hat{z}$ axis, that is, points that are just to the right of the $(\hat{x}, \hat{y}, \hat{z}, \hat{t})$ point, cannot be updated as well as any of the points along the $\hat{y}$ axis (because of $\hat{y} \sim 3\hat{y}$ and $\hat{z} \sim 3\hat{z}$). On the other hand the link positioned at $(\hat{x}, \hat{y}+1, \hat{z}+1, \hat{t})$ and $(\hat{x}, \hat{y}+2, \hat{z}+2, \hat{t})$ can themselves be updated. The combination of these three points will constitute the first $\hat{t}$ plane. The second plane, the $\hat{t}+1$ plane, is obtained by incrementing over the $\hat{z}$ direction in relation to the $(\hat{x}, \hat{y}, \hat{z}, \hat{t})$ point by one unit to the right. The starting point is therefore $(\hat{x}, \hat{y}, \hat{z}+1, \hat{t})$. Eliminating the appropriate points according to the rule defined in Eq.(5.20), tells us that the next two allowed points in that plane are: $(\hat{x}, \hat{y}+1, \hat{z}+2, \hat{t})$ and $(\hat{x}, \hat{y}+2, \hat{z}, \hat{t})$. Similarly for the third plane, namely the $\hat{t}+2$ plane. The set of these three planes, consisting of the combination of three points over three planes (making up a total of nine points), covering every points of the 3×3 lattice sites make up the first mask. It is not difficult to see that we can cover all of the nine lattice links that need to be updated with three orthogonal masks as shown in Fig. 5.6. In this figure x-oriented links which can be updated at the same time are indicated by a solid dot. Note that each of these masks is related by a diagonal shift of the nine-point lattice "window".

We can now also extend this thinking to include the t direction, by stacking the

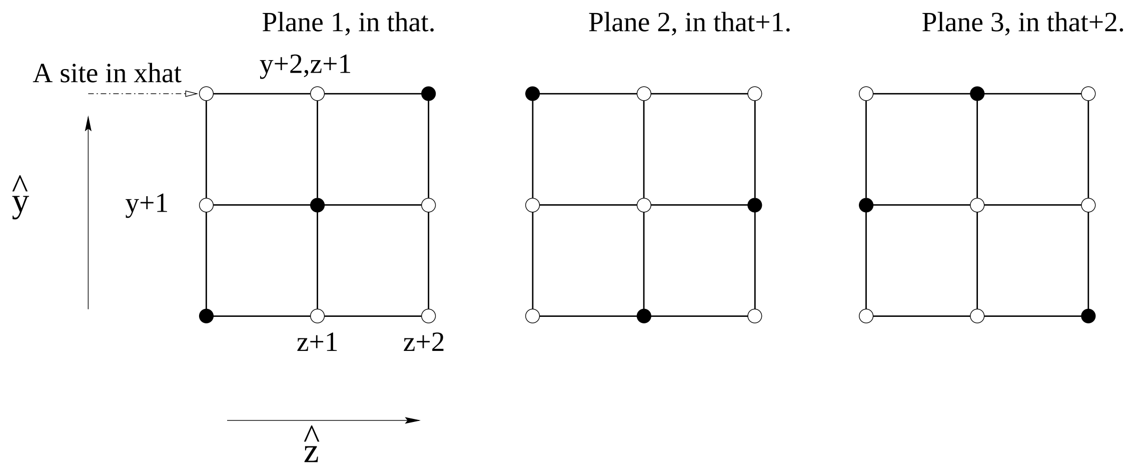


Figure 5.6: Schematic illustration of the lattice masking when using the 1×2 plaquette improved action.

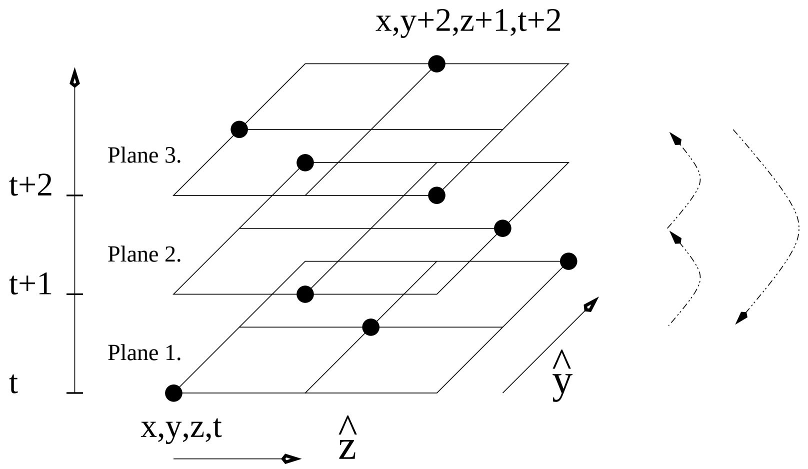


Figure 5.7: Illustration of the cyclic plane rotation in the improved masking.

three two-dimensional y–z masks on top of each other as shown in Fig. 5.7. We must stack the planes so that when viewed along any of the three axes the solid dots in any one Cartesian planes always have the appearance of one of the planes in Fig. 5.6. We see that this can be achieved in three ways by the stacking in Fig. 5.7 and its two cyclic permutations. These three three-dimensional masks when summed give the identity (i.e., the sum includes all points) and are orthogonal to each other (i.e., the sum includes all points only once).

We can now give a simple geometrical picture of what we are doing, which will simplify the generalization that we give in the next section. For $\hat{x}$-oriented links, the directions $\hat{y}$, $\hat{z}$, and $\hat{t}$ directions are all symmetrical and each direction requires a step of 3 to reach the next updatable link. Hence, we need to construct a complete set of orthogonal masks in three dimensions for a $3 \times 3 \times 3$ cube, where no two points in the cube lie on the same Cartesian axis (i.e., only diagonally related points). This is simple to do. Let us consider the bottom plane (i.e., plane 1) of Fig. 5.7 and connect the three solid dots by a diagonal line. We see that plane 2 is obtained from plane 1 by a diagonal shift of this line by one diagonal half-step, and similarly for plane 3. In visualizing this it may help to imagine surrounding the cube by many identical copies of itself and moving the diagonal line through diagonal half-steps across all of these cubes simultaneously. All three three-dimensional masks are obtained in the same way but start with plane 1, plane 2, and plane 3 respectively.

So for the $\hat{x}$-oriented links we need 3 masks for each equal-x hyper-plane (i.e., a three-volume here) and we have two independent equal-x hyper-planes, giving a total of 6 masks for each Cartesian direction for the link orientation. Since there are 4 orientations, then there is a total of 24 masks needed for an action containing both 1×1 and 1×2 plaquettes. Thus a single lattice sweep must take at least 24 sequential serial calculations even on the most parallel computing architecture.

The masking procedure outlined here for this action can only be implemented when the number of lattice points in each dimension is a multiple of three. Inspection of Fig. 5.7 reveals the periodicity of three is required to maintain separation of links at the boundary. Since simulations are usually carried out on lattices with even numbered sides, this restricts the length of the lattice sides to multiples of six. Fortunately, multiples of four are easily obtained as described in the next section. Moreover, Sec. 5.2.4 reports a high-performance mask for this action with a periodicity of four.

It is interesting to note that when implementing this masking procedure on the CM-5 we achieved optimum performance by calculating the updates for all links on the lattice and by then only implementing those updates that were appropriate for the particular mask being used at the time. In other words for the lattices that we have studied so far on the CM-5 it was more efficient to calculate link updates that were never used, than it was to split the masked links over the various processor nodes and update only these masked links. This was due to the fact that there was a large overhead of communication time in assigning the masked links across the processors. The point of this observation is that the optimal use of the masks will in general depend on the details of the parallel computing architecture being used.

5.2.3 Generalization of the Masking Procedure for an $n\times m$ Wilson Loop

We can now generalize the algorithm presented in Sec. 5.2.2 for arbitrarily improved planar actions. Let us begin as before by considering the update of links oriented in the $\hat{x}$-direction. Let us assume that we have an action with $n\times m$ links where the n refers to the $\hat{x}$ direction and the m refers to the $\hat{y}$, $\hat{z}$, $\hat{t}$ directions. We will eventually argue that only the $n_{\max}\times n_{\max}$ case, where $n_{\max}$ is the greater of n and m, is necessary in the general case. As shown in Fig. 5.8 the nearest simultaneously updatable links are separated by n steps in the $\hat{x}$ direction and $(m+1)$ steps in the other three Cartesian directions.

Hence we see that we can write in our notation for the four Cartesian orientations of the links that

$$\hat{x}: \quad \hat{x}\sim n\hat{x}\ ,\quad \hat{y}\sim(m+1)\hat{y}\ ,\quad \hat{z}\sim(m+1)\hat{z}\ \text{ and }\ \hat{t}\sim(m+1)\hat{t}, \tag{5.24}$$
$$\hat{y}: \quad \hat{x}\sim(m+1)\hat{x}\ ,\quad \hat{y}\sim n\hat{y}\ ,\quad \hat{z}\sim(m+1)\hat{z}\ \text{ and }\ \hat{t}\sim(m+1)\hat{t}, \tag{5.25}$$
$$\hat{z}: \quad \hat{x}\sim(m+1)\hat{x}\ ,\quad \hat{y}\sim(m+1)\hat{y}\ ,\quad \hat{z}\sim n\hat{z}\ \text{ and }\ \hat{t}\sim(m+1)\hat{t}, \tag{5.26}$$
$$\hat{t}: \quad \hat{x}\sim(m+1)\hat{x}\ ,\quad \hat{y}\sim(m+1)\hat{y}\ ,\quad \hat{z}\sim(m+1)\hat{z}\ \text{ and }\ \hat{t}\sim n\hat{t}\,. \tag{5.27}$$

We can now follow the arguments of the previous section. Let us consider a fixed-x hyper-plane (i.e., three-volume). In place of a 3×3 three-volume we will now need an $(m+1)\times(m+1)\times(m+1)$ three-volume. Furthermore, we will need a complete set of orthogonal and diagonal masks for this. Let us again look along the $\hat{x}$ direction at

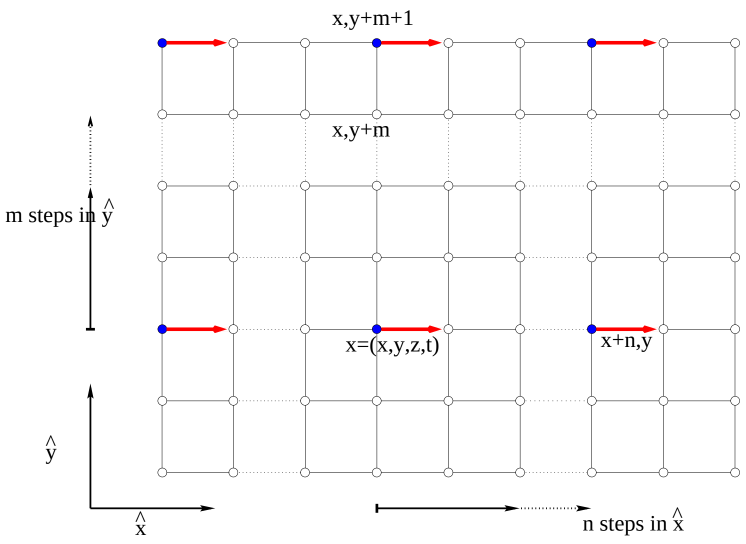


Figure 5.8: The highlighted links with arrows are the ones that can be simultaneously updated for an action containing up to $n \times m$ plaquettes, where here n refers to the $\hat{x}$ direction and m applies to the other three Cartesian directions.

a fixed t plane for now, i.e., we are looking at a $\hat{y}$–$\hat{z}$ plane as in Fig. 5.9. Let us refer to the $(m + 1) \times (m + 1)$ two-dimensional plane with the updatable links (solid dots) along the diagonal as plane 1. Then we can generate the other m two-dimensional planes by diagonal half-shifts as before as depicted in Figs. 5.10 and 5.11. We can then sequentially stack these planes in the $\hat{t}$ direction as before to form the first of the three-dimensional masks. The other m three-dimensional masks are then generated from this first mask by the cyclic permutations of the $m + 1$ planes as in Sec. 5.2.2. Hence we have generated the desired complete set of $(m + 1)$ orthogonal three-dimensional diagonal masks.

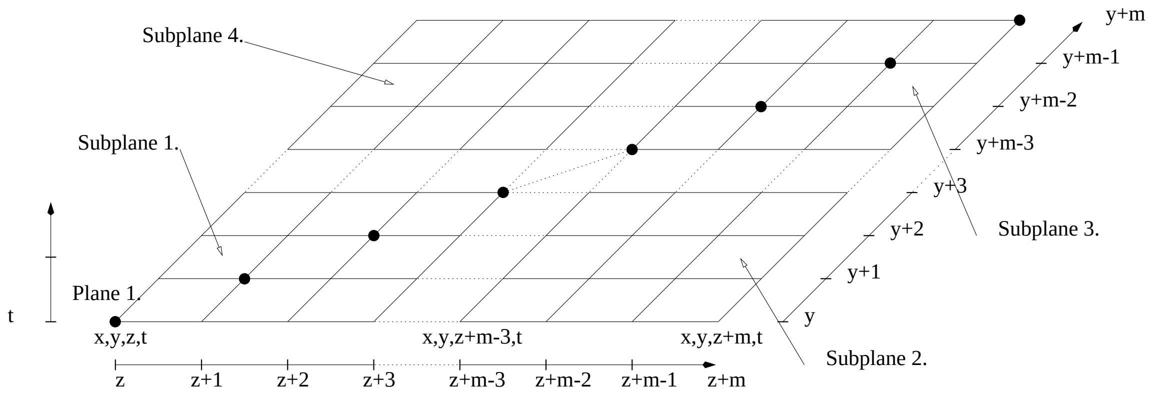


Figure 5.9: Plane 1 with the $(m+1)$ updatable sites on the main diagonal of the $\hat{y}-\hat{z}$ plane.

So for each fixed x-hyper-plane (i.e., three volume) we need $(m+1)$ masks. We will need such a set of masks for the n values of x. The general result is that for updating the links oriented in the $\hat{x}$ direction we need a total of $n \times (m + 1)$ masks and we have seen that the construction of these masks is straightforward. The construction of the masks for the other Cartesian orientations of the links proceeds identically. This total number of masks is $n_{\text{mask}} = 4 \times n \times (m+1)$. The periodicity of the mask is governed by

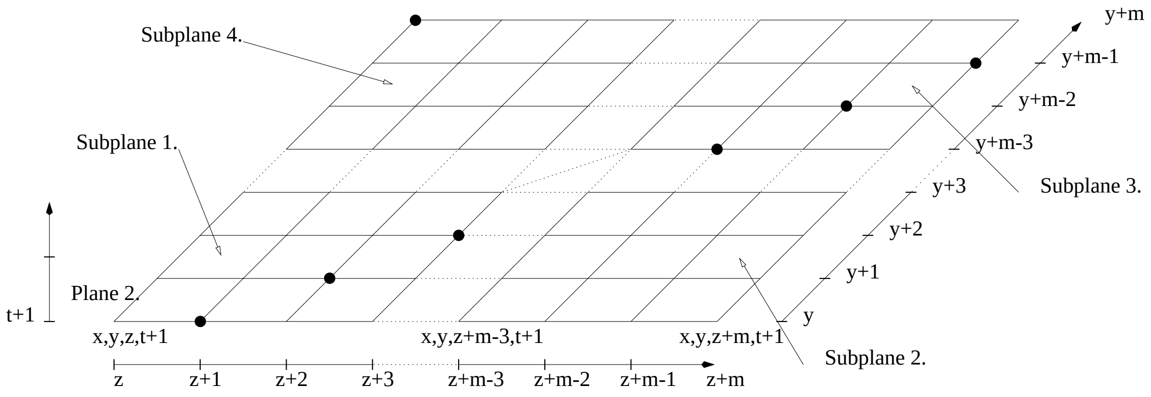


Figure 5.10: Plane 2.

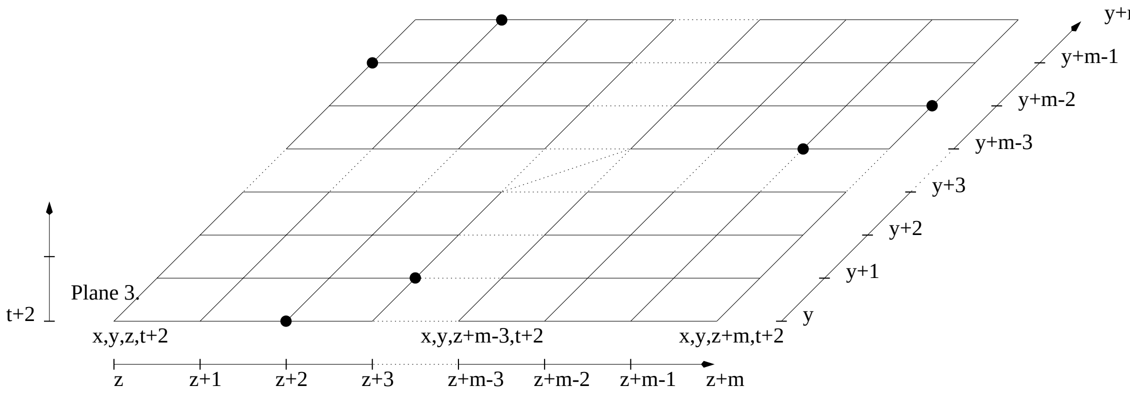


Figure 5.11: Plane 3.

the last factor, $(m+1)$, and the lengths of the lattice dimensions must be a multiple of this number. The reason for this is that if this were not the case then the imposition of the necessary periodic boundary conditions would cause link collisions, where a link being updated uses one or more other links which are simultaneously being updated.

Any improved lattice action of physical interest must be both Z_4-symmetric (i.e., symmetric under the arbitrary interchange of the four Cartesian directions) and translationally invariant. Thus for such actions every link will find itself occurring in every possible position for every plaquette in the improved action. We then see, as we did in Sec. 5.2.2 and Fig. 5.3, that the number of steps needed in each direction is determined by the longest plaquette side appearing in the action. Let us denote the longest plaquette side appearing in the action as n_{max}. Then we see that the number of steps needed in the various Cartesian directions is given by

$$\begin{array}{lllll}
\hat{x}: & \hat{x} \sim n_{\text{max}}\hat{x}\ , & \hat{y} \sim (n_{\text{max}}+1)\hat{y}\ , & \hat{z} \sim (n_{\text{max}}+1)\hat{z} \text{ and } & \hat{t} \sim (n_{\text{max}}+1)\hat{t}, \\
\hat{y}: & \hat{x} \sim (n_{\text{max}}+1)\hat{x}\ , & \hat{y} \sim n_{\text{max}}\hat{y}\ , & \hat{z} \sim (n_{\text{max}}+1)\hat{z} \text{ and } & \hat{t} \sim (n_{\text{max}}+1)\hat{t}, \\
\hat{z}: & \hat{x} \sim (n_{\text{max}}+1)\hat{x}\ , & \hat{y} \sim (n_{\text{max}}+1)\hat{y}\ , & \hat{z} \sim n_{\text{max}}\hat{z} \text{ and } & \hat{t} \sim (n_{\text{max}}+1)\hat{t}, \\
\hat{t}: & \hat{x} \sim (n_{\text{max}}+1)\hat{x}\ , & \hat{y} \sim (n_{\text{max}}+1)\hat{y}\ , & \hat{z} \sim (n_{\text{max}}+1)\hat{z} \text{ and } & \hat{t} \sim n_{\text{max}}\hat{t}\,.
\end{array}$$

Hence the number of masks in general for an improved action will then be given by

$$n_{\text{mask}} = 4 \times n_{\text{max}} \times (n_{\text{max}}+1), \tag{5.28}$$

and the lattice will need the length in each dimension to be an integral multiple of $(n_{\text{max}}+1)$.

It is useful to note that the *linear* masking for the standard Wilson action is the one that is extended initially in Sec. 5.2.2 and is subsequently generalized in this section. For the standard Wilson action (i.e., 1×1 plaquettes only) we see that $n_{\max} = 1$ and hence $n_{\text{mask}} = 4 \times 1 \times 2 = 8$ as we found for the linear (and checkerboard) mask. For the improved action that we have studied (i.e., 1×1 and 1×2 plaquettes) we have $n_{\max} = 2$ and hence $n_{\text{mask}} = 4 \times 2 \times 3 = 24$ or 6 masks per link direction as found in Sec. 5.2.2. However, this way of proceeding for the plaquette plus rectangle improved action would require each lattice dimension be a multiple of $(n_{\max} + 1) = 3$, but since we also typically want our lattices to have even lengths then that means each side of the lattice would need to be a multiple of 6 in length. Since the result in Eq. (5.28) is a lower bound, we can of course always choose to enlarge the period of our masking by choosing $n_{\max} + 2$ for the last factor in Eq. 5.28 rather than $n_{\max} + 1$. This will still ensure that no link collisions occur. For example, for the plaquette plus rectangle improved action we can use $(n_{\max} + 2) = 4$ instead of $(n_{\max} + 1) = 3$ in Eq. (5.28), so that any lattice lengths which are multiples of 4 become available at the cost of requiring 32 masks rather than 24. Fortunately, for this case a more efficient mask can be realized and will be presented in the next section.

5.2.4 Non-planar Considerations

We have presented a method for identifying links which may be simultaneously updated during Monte-Carlo updates or cooling sweeps. The generality of the algorithm allows one to parallelize link updates for planar actions of any degree of non locality. In this section we extend this analysis to a few special cases of actions in which out-of-plane considerations are necessary. Both cases are centred around the plaquette plus rectangle action of Eq. (5.16) in which 1×1 and 1×2 Wilson loops are considered in the action. Such actions dominate current improved gauge action analyses.

In Sec. 5.2.2 we illustrated how such an action can be masked through the consideration of an elementary $3 \times 3 \times 3$ cube in which one-third of the links may be simultaneously updated. However, only every second link in the direction of the links is updated simultaneously as illustrated in Fig. 5.4. Hence six masks per link direction are required.

Here we consider an alternative masking specialized to the 1×1 and 1×2 Wilson loop actions. Fig. 5.12 illustrates the manner in which these Wilson loops may be nested, such that one need not restrict the mask to every second link in the direction of the links being updated. This technique will reduce the number of masks by a factor of two, at the expense of considering an elementary $4 \times 4 \times 4$ cube in which one-quarter of the links may be simultaneously updated. Fig. 5.13 displays the four planes to be cycled through in which the links to be updated simultaneously are indicated by the solid dot. Hence only four masks per link direction are required. Moreover, the lattice dimensions (usually even numbers) can now be multiples of four as opposed to six.

The out-of-plane considerations required for the nested action are also indicated in Fig. 5.12. Hence it becomes apparent that not only the three links at $(x, y+1)$, $(x, y+2)$, and $(x, y+3)$, be avoided, but also the links two-steps in a direction orthogonal to the link direction and one step in a third direction (similar to moves of a Knight on a chess board) must be avoided.

Inspection of the four planes to be cycled through in the elementary $4 \times 4 \times 4$ cube displayed in Fig. 5.13 indicates that such Knight moves are already avoided in this mask. However, it also becomes clear that the ordering of the planes is crucial. For

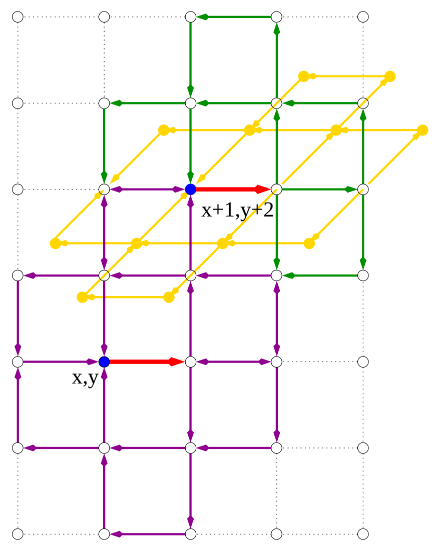


Figure 5.12: Two elementary cells for an action involving 1×1 and 1×2 Wilson loops are nested together such that one need not restrict the mask to every second link in the direction of the links being updated. The links with the positions labelled are the ones that can be simultaneously updated. The out of plane plaquette-plus-rectangle illustrates additional links that cannot be simultaneously updated.

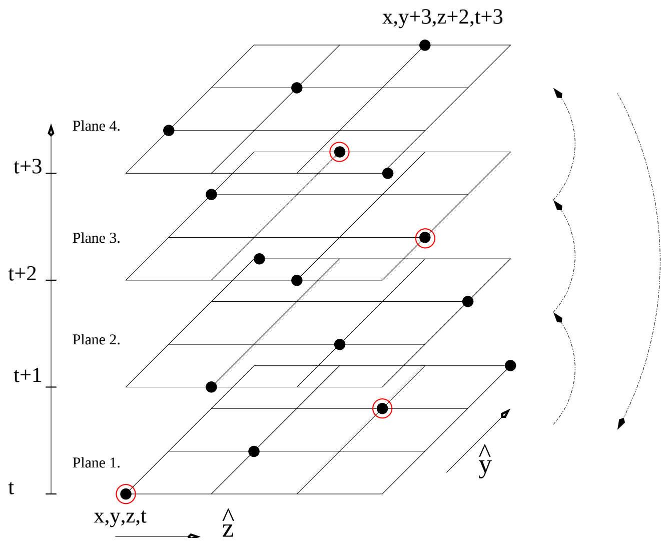


Figure 5.13: The four planes to be cycled through in the elementary $4 \times 4 \times 4$ cube. One-quarter of the links may be updated simultaneously and are indicated by the solid dot. The circled sites are an example of the sites surviving when the out of plane "chair" or "parallelogram" link paths are included in the action.

example interchanging the positions of planes 2 and 3 would cause "link collisions" within the nested mask.

Finally we consider non-planar actions in which one step out of the plane of the 1×1 and 1×2 Wilson loops is required. Such non-planar paths are introduced to eliminate small but finite $\mathcal{O}(g^2a^2)$ errors where g is the gauge coupling constant. The

Table 5.1: Parameters used in the fit of $a(\beta)$. The conversion of the $\beta_{\rm Lee}$ definition is done through $\beta = (3/5)\beta_{\rm Lee}$.

Action	β	Volume	$\beta_{\rm Lee}$	a fm	u_0^4	Physical Volume fm
Improved	3.75	$6^3 \times 12$	6.25	0.40(3)	0.4423	$2.40^3 \times 4.80$
Improved	4.08	$8^3 \times 14$	6.80	0.27(1)	0.5372	$2.16^3 \times 3.78$
Improved	4.20	$10^3 \times 16$	7.00	0.24(1)	0.5658	$2.40^3 \times 3.84$

six-link paths commonly referred to as the "chair" and "parallelogram" [25] introduce a link parallel to that being updated which is one-step orthogonal to the link direction and one step in a third direction.

Inspection of Fig. 5.13 indicates that such 1 by 1 moves eliminates fully two of the four planes and half of the parallel sites on each surviving plane. An example of four of the sites which may still be updated in parallel are indicated by the circled sites in Fig. 5.13. As a result there are now 16 masks required per link direction instead of 4. Now a total of 64 masks is required for this action which is still regarded as rather local.

The introduction of even the most local non-planar paths can have a serious detrimental effect on the level of parallelism that is possible. It is easy to see that one can rapidly eliminate all sites in an elementary $n \times n \times n$ cube with non-planar loops, leading to n^3 masks per link direction.

5.3 Estimating the Lattice Spacing

Before starting any sort of simulation it is necessary to have an idea on the scale one is working on. The lattice spacing as a function of the coupling β can be estimated from the two loop perturbative β function expressing the lattice spacing a as a function of β is

$$a(\beta) = \frac{1}{\Lambda}\left(\frac{12\pi}{11\,N\,\alpha}\right)^{51/121} \exp\left(-\frac{6\pi}{11\,N\,\alpha}\right) , \qquad (5.29)$$

where $N = 3$ for $SU_c(3)$ and

$$\alpha_0 = \frac{g_0^2}{4\pi} = \frac{2\,N}{4\pi}\frac{1}{\beta_0} . \qquad (5.30)$$

One must be careful in making contact with the correct definition of β as defined above ($\beta_0 = 2\,N/g_0^2$) when considering the plaquette plus rectangle action. I found that a direct fit from Eq. (5.29) worked best in estimating the lattice spacing. Using data already published in Ref. [25, 26, 27] and taking note of the fact that a multiplication of β in Eq.(5.16) by a factor of 5/3 reproduces their definition. I was able to obtain a fit for the lattice spacing. The details of the parameter used in the fit are shown in Table 5.1.

The single parameter fit on the scalar variable Λ, was carried out using Eqs. (5.30) and (5.29), which lead to

$$a(\beta) = \frac{1}{\Lambda}\left(\frac{24\pi^2\beta}{11\,N_c^2}\right)^{51/121} \exp\left(-\frac{12\pi^2\beta}{11\,N_c^2}\right) . \qquad (5.31)$$

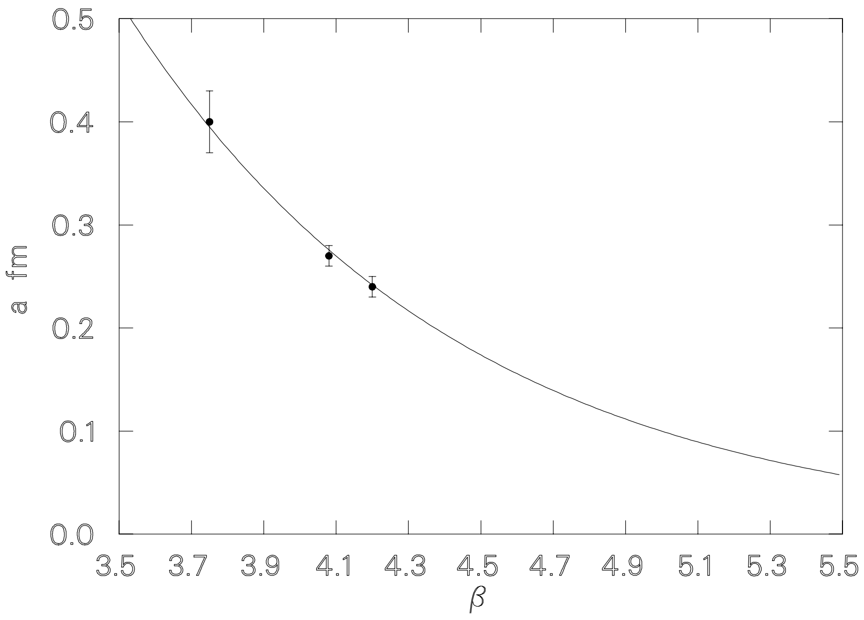


Figure 5.14: Graph to extract the running coupling using a fitted curve: the 2 loop perturbative β function. The fitted value for Λ is $\Lambda = 0.0719152$ fm^{-1}.

The resulting fit for Λ gave a value of $\Lambda = 0.0719152$ fm^{-1}, and shown in Fig. 5.14.

It is now possible to use this fitted curve to get an estimate for the lattice spacing as a function of the lattice coupling β. So for example, a lattice spacing of 0.2 fm would estimate a coupling of $\beta = 4.38$.

This way of estimating the lattice spacing may not be very accurate, in fact an explicit calculation using Wilson loops to extract the spacing and to calculate the static quark potential reports a value at $\beta = 4.38$ of $a = 0.165(2)$ [28]. Nevertheless, it is a good way to get a first estimate on the spacing.

Chapter 6

Anisotropic Lattices with Standard Wilson and Improved Action

One of the advantages of anisotropic lattices with temporal lattice spacing a_t that are smaller than spatial lattice spacing a_s, is that they allow one to extract results on coarse lattice as if one was working on fine ones. Such lattices are also useful in the determination of signal to noise ratio correlation functions calculated in Monte Carlo simulations of hadrons. This is because they give more time slices with an accurate signal.

6.1 Wilson Action

6.1.1 With Time Improvement

The simplest case of anisotropic gauge action is the anisotropic version of the standard Wilson gauge action described in Sec. 2.1.3. For this action no coefficients have to be tuned to restore space–time exchange symmetry up to $\mathcal{O}(a^2)$. An important step in any action is to be able to take the proper continuum limit. One then has to know the true or renormalized anisotropy $\xi = a_s/a_t$ as a function of the bare parameters.

The Anisotropic Wilson action for $SU_c(N)$ is written as

$$S_{\text{Wil}}^{\text{Ani}} = \frac{\beta}{N} \sum_{x,s>s'} \frac{1}{\xi_0} \Re e\, \text{Tr}(1 - U_{ss'}^{\text{sq}}(x)) + \xi_0 \Re e\, \text{Tr}(1 - U_{0s}^{\text{sq}}(x)). \tag{6.1}$$

where the coupling is still the bare coupling $\beta = 6/g^2$. Here the link operator, $P_{\mu\nu}(x)$, is the 1×1 plaquette operator defined in Eq. (2.2). At the classical level the renormalized anisotropy ξ is proportional to the ratio of the two lattice spacing, the temporal and spatial spacing i.e. $\xi = a_s/a_t$. It was suggested in Ref. [31] that the renormalized anisotropy, ξ, may be related to the bare anisotropy, ξ_0 using a method based on the ratios of Wilson loops. Here we use mean field improvement terms to get a direct estimate of the renormalized anisotropy. We then rewrite the Anisotropic Wilson action Eq. (6.1) as

$$S_{\text{Wil}}^{\text{Ani}} = \frac{\beta}{N\, u_s^4} \sum_{x,s>s'} \frac{1}{\xi_0} \Re e\, \text{Tr}(1 - U_{ss'}^{\text{sq}}(x)) + \xi_0 \Re e\, \text{Tr}(1 - U_{0s}^{\text{sq}}(x)), \tag{6.2}$$

In order to reproduce the results in Section 4.4 and 5 just set the renormalized anisotropy $\xi = a_s/a_t$ to 1.

6.2 Improved Action

6.2.1 With Time Improvement

We saw in Chapter 5 that perturbation theory by itself does not reliably determine the coupling in the improved action. It is known that a superior estimate of the coupling is obtained when perturbation theory is combined with mean field improvement. Mean field theory is introduced when the gauge links variables are renormalized by the mean field factor. In the spatial direction $U_j(x) \longrightarrow U_j(x)/u_s$ and in the temporal direction $U_t(x) \longrightarrow U_t(x)/u_t$, where u_s and u_t are the spatial and temporal mean links factors:

$$u_s = \left\langle \frac{1}{N_c} \Re e \, \mathrm{Tr} U_{\mathrm{sp}}(x) \right\rangle^{\frac{1}{4}}, \tag{6.3}$$

$$u_t = \frac{\left\langle \frac{1}{N_c} \Re e \, \mathrm{Tr} U_{\mathrm{tp}}(x) \right\rangle^{\frac{1}{2}}}{u_s}. \tag{6.4}$$

The power of the mean link increases by the number of links the Wilson loop is connected by. The anisotropic version of the gauge action defined in Eq. (5.1) is given by [11, 33]

$$S_I[U] = \beta \left\{ \frac{5}{3\xi u_s^4} W_{\mathrm{sp}} + \frac{5\xi}{3u_s^2 u_t^2} W_{\mathrm{tp}} - \frac{1}{12\xi^2 u_s^6} W_{\mathrm{sr}} - \frac{\xi}{12 u_s^4 u_t^2} W_{\mathrm{str}} - \frac{\xi}{12 u_s^2 u_t^4} W_{\mathrm{ttr}} \right\}. \tag{6.5}$$

Where $\beta = 6/g^2$ is the usual coupling and $W_c = 1/3 \sum_c \Re e \, \mathrm{Tr}(1 - U_c)$ are the usual contours. U_{sp} denotes the spatial plaquettes, U_{tp} refers to the temporal plaquette, U_{sr} expresses the product of link variables about a planar 2×1 spatial rectangular loop, U_{str} indicates the short temporal rectangles (one temporal, two spatial) and U_{ttr} refers to all temporal rectangles (two temporal spacing one spatial). When the temporal lattice spacing, a_t, is significantly smaller than the spacial one, a_s, we expect the temporal mean link u_t to be very close to 1 or close to its continuum value. It is then common practice to set the temporal mean link to its continuum value $u_t = 1$ and only update u_s.

6.2.2 Without Time Improvement

It was pointed out [33] that the gluon spectrum calculated from Eq. (6.5) had some high energy states arising from W_{ttr} term, which spans two time slices. These modes occur at energies of order $2/a_t$. although they have little effect on the gluon spectrum, they could cause problems when applying the variational method to extract masses from short time correlation functions.

These modes may be cancelled out by not doing any time improvement, i.e., "relaxing" the time improvement. The improved anisotropic gauge action then takes the form

$$S_{II}[U] = \beta \left\{ \frac{5}{3\xi u_s^4} W_{\mathrm{sp}} + \frac{4\xi}{3u_s^2 u_t^2} W_{\mathrm{tp}} - \frac{1}{12\xi u_s^6} W_{\mathrm{sr}} - \frac{\xi}{12\xi^2 u_s^4 u_t^2} W_{\mathrm{str}} \right\}. \tag{6.6}$$

Where the contour loops have the same meaning as in Eq. (6.5), explicitly we have

$$W_{\mathrm{sp}} = \frac{1}{3} \sum_x \sum_{i \neq j} \Re e \, \mathrm{Tr} \left[1 - U_i(x) U_j(x+\hat{\imath}) U_i^\dagger(x+\hat{\jmath}) U_j^\dagger(x) \right],$$

$$
\begin{aligned}
W_{\rm tp} &= \frac{1}{3}\sum_x\sum_i \Re e\,\mathrm{Tr}\left[1 - U_t(x)U_i(x+\hat{t})U_t^\dagger(x+\hat{i})U_i^\dagger(x)\right],\\
W_{\rm sr} &= \frac{1}{3}\sum_x\sum_{i\neq j} \Re e\,\mathrm{Tr}\left[1 - U_i(x)U_i(x+\hat{i})U_j(x+2\hat{i})U_i^\dagger(x+\hat{i}+\hat{j})U_i^\dagger(x+\hat{j})U_j^\dagger(x)\right],\\
W_{\rm str} &= \frac{1}{3}\sum_x\sum_i \Re e\,\mathrm{Tr}\left[1 - U_i(x)U_i(x+\hat{i})U_t(x+2\hat{i})U_i^\dagger(x+\hat{i}+\hat{t})U_i^\dagger(x+\hat{t})U_t^\dagger(x)\right].
\end{aligned}
$$

Where x labels the lattice sites and the indices i, j are the spatial indices. This action, designed for use with the temporal lattice spacing much less that the spacial one, has $\mathcal{O}(a_s^4, a_t^2, \alpha_s a_s^2)$.

6.3 The Computer Code for the Anisotropic Lattices

I now describe the computer code that implements the generation of gauge field configuration for the gauge actions described by Eqs. 6.1, 6.5 and 6.6. The implementation is done by just modifying the gauge field generator code for the improved action described in Sects. 4.2 and 5.2. Here I shall describe the key routines in more depth.

The main differences with the isotropic code are the calculations of the staples, the tadpole improvement factors and few extra switches in the front end of the program. The masking, the accept/reject steps and the construction of the $SU_c(3)$ matrices remain unchanged.

6.3.1 The Various Switches

The switches are set in the front end of the program, a copy of the code is shown in Appendix. E.10. The main switches are the coupling constant β, the renormalized anisotropy ξ, the choice of the action, a switch that decides if time improvement needs to be done or not (when the switch is turned off, it means that we are calculating Eq. (6.5) else we are calculating Eq. (6.6)), and two other switches for the tadpole factors.

The renormalized anisotropy is what sets the scale of the anisotropy. It can be set to any positive value, however most calculation set the anisotropy from around 3 to 5. The bare anisotropy is usually set as an integer. Once the anisotropy has been set the value gets passed into other routines. The only time it comes into play is for the calculation of the Wilson loops. When the time improvement is switched off the time switch parameter takes a value of 0. In that case the term $W_{\rm ttr}$ appearing in Eq. (6.5) is not calculated and the factor in front of $W_{\rm tp}$ (in the squares routine) is set to 4/3, we are therefore considering Eq. (6.6). Else if the time improvement switch is turned on the $W_{\rm ttr}$ is calculated and the factor in front $W_{\rm tp}$ is set to 5/3.

During the thermalization process after each Monte Carlo update (i.e. a full sweep over the lattice) the tadpole improvement factors are calculated. The tadpole factors are averaged over the number of intermediate sweeps. The action is then calculated with the latest tadpole values. The resulting action and averaged mean link values are reported to an output file.

6.3.2 The calculation of the Anisotropic Squares and Rectangles

I now describe in more detail the anisotropic routines squares and rectangles, Appendix E.12 and E.13 respectively. The argument list for both of these routines consist of the link variables (all lattice arrays like the link variable $U_\mu(x)$ are made up of a real and imaginary part), the staples, the Lorentz index, a variable which acts as a flag if all the plaquettes around the link variable are to be calculated, the time improvement switch, the tadpole factors and finally the anisotropic factor.

The link variable is the input variable, and the staples are the variable being calculated. The first step in the routine is to setup a dimension 3 array for each of the directional indices. This index is passed into the routine by the argument list. This array will loop over each dimension of the array. The procedure is repeated for each of the four Lorentz index. The subroutine squares is also used for the calculation of the tadpole factors. In the case of the spatial tadpole calculation we only need to worry about the spatial indices. The loop is therefore over just the spatial indices as opposed to the temporal tadpole where the tadpole is only calculated in the time direction.

The next step is the calculation of the anisotropic factors. These factors need to be inserted on the right Wilson loop. This is dictated by the action. The rest of the routines are the same as in the isotropic case, described in Sec. 5.2.1.

6.4 Some Numerical results

Here I show some of the results obtained for the anisotropic action described above. The idea was to insert the mean field improvement factors in the code and compare it to the bare anisotropy calculated from the string tension.

6.4.1 Results for the Anisotropic Wilson Gauge Action

In this section I show some results obtained from Eq. (6.1), I then compare the results with those obtained by Klassen [31]. In Table 6.1 I show some of the results obtained by Klassen [31] for a handful of small lattices.

ξ	β_{kl}	η
2.0	5.4	1.2658(184)
	5.8	1.1905(92)
3.0	5.5	1.3351(134)
	5.6	1.3043(130)
	6.3	1.1947(38)
4.0	5.4	1.4126(245)
	5.6	1.3374(94)

Table 6.1: Simulation results for the renormalization of the anisotropy, $\eta = \xi/\xi_0$. Simulation was done a $16^3 \times 48$. The table has been carved from Table I in Ref. [31].

Here we are estimating the renormalized anisotropy $\eta = \xi/\xi_0$ using the mean field tadpole factors. So we have $\eta = u_t/u_s$. The coupling used in our code is related to the one defined in Ref. [31] by $\beta = \beta_{\mathrm{kl}}/\eta^3$.

In Table. 6.2 I report the tadpole factors for one lattice. Taking the ratio of the two tadpole factors gives $u_t/u_s = 1.296$. These results should be compared with the lattice generated at $\beta_{\mathrm{kl}} = 5.6$ with $\xi = 3.0$ where the renormalized anisotropy is estimated to be $\eta = 1.3043(130)$. We can see a very good agreement between the two different methods, showing that estimating the renormalised anisotropy with the tadpole improvement factors gives a result very close to the one calculated using the Wilson loops.

Table 6.2: Parameters used to generate the anisotropic lattices. The coupling value is denoted as before by β, ξ is the renormalized anisotropy and the spatial and temporal tadpole factors are denoted by u_s and u_t respectively.

Action	Volume	ξ	β	u_s	u_t
Wilson	$6^3 \times 30$	3.0	2.55	0.7715	1.00

6.4.2 Results for the Peardon and Morningstar Gauge Action

In Table 6.3, I show the results for four different lattices. Three of which were just used to get a value for the spatial tadpole factor after a small number of sweeps. These three lattices are compared with the results obtained in [32], (Table I). There I am comparing their obtained value for the spatial tadpole factor u_s^4 with the one I get from my code. For example, if I compare the result for spatial tadpole factor, u_s, obtained from the $6^3 \times 30$ with a renormalized anisotropy of $\xi = 5.0$ and a coupling of $\beta = 1.70$, I get $u_s = 0.7369$. Taking this value to the fourth power gives a value of $u_s^4 = 0.2948$, compared with a value of $u_s^4 = 0.295$[1].

I now compare the results obtained from the fourth lattice ($12^3 \times 36$) which is used to calculate the renormalized anisotropy. The renormalized anisotropy can be obtained by taking the ratio of the spacing, namely a_s/a_t, and then be compared with the input anisotropy ξ. Fifty configurations were generated and twenty were analysed to extract both the temporal and spatial lattice spacing. Taking the ratio of the spacing (a_s/a_t), shown in Tab. 6.3, gives a renormalized anisotropy of $a_s/a_t \simeq 3.03 \pm 0.04$ compared with an input value of exactly 3.0. Similarly for the $12^3 \times 48$, using just ten configurations, the renormalized anisotropy comes out as $a_s/a_t \simeq 4.00 \pm 0.34$. The mean field improved estimate for the renormalized anisotropy is therefore a very good estimate.

[1] $u_s^4 = 0.295$ may be found in Table I of Ref. [32]

Table 6.3: Parameters used to generate the anisotropic lattices. The coupling value is denoted as before by β, ξ is the renormalized anisotropy and the spatial and temporal tadpole factors are denoted by u_s and u_t respectively.

Action	Volume	ξ	β	a_s fm	a_t fm	u_s	u_t	Physical Volume (fm)
Improved	$6^3 \times 18$	3.0	1.90	-	-	0.7636	1.00	-
Improved	$6^3 \times 30$	5.0	1.70	-	-	0.7369	1.00	-
Improved	$6^3 \times 30$	5.0	1.90	-	-	0.7567	1.00	-
Improved	$12^3 \times 36$	3.0	2.40	0.267(2)	0.088(1)	0.8056	1.00	$3.20^3 \times 3.17$
Improved	$12^3 \times 48$	4.0	2.35	0.26(1)	0.065(5)	0.7969	1.00	$3.16^3 \times 3.02$

Chapter 7

Cooling and Smearing $SU_c(3)$ Gauge Field configurations

The approximate chiral symmetry of QCD arises from the smallness of the masses of the u and d quarks. Typical strong interaction scales are around $\Lambda_{\overline{MS}} \approx 200$ MeV. The light quark masses are of order 10 MeV. In the chiral limit for the light quark sector where quark masses vanishes, $m_u = m_d = 0$ and one can have an exact global $U_V(2) = U_V(1)\otimes SU_V(2)$ flavour symmetry, where $U_V(1)$ expresses the quark (fermion) number conservation, and $SU_V(2)$ the isospin symmetry for the vector current V. At zero quark mass $U_V(2)$ can be extended to $U_V(2)\otimes U_A(2) = U_V(1)\otimes U_A(1)\otimes SU_V(2)\otimes SU_A(2)$ which is generated by the vector and axial currents.

The $SU_A(2)$ axial isospin symmetry is spontaneously broken by the vacuum expectation value of the scalar quark densities, giving rise to three massless pseudoscalar bosons, one for each of the broken generators of a spontaneously broken global symmetry. In a somewhat poorer approximation the strange quark mass ($m_s \approx 150$ MeV) can also be neglected. In this case the chiral flavour symmetry is extended from $SU_V(2)\otimes SU_A(2)$ to $SU_V(3)\otimes SU_A(3)$. In fact, the symmetry of the classical continuum QCD with three massless quarks is larger than $SU_V(3)\otimes SU_A(3)$ namely $U_V(3)\otimes U_A(3) = U_V(1)\otimes U_A(1)\otimes SU_V(3)\otimes SU_A(3)$. The additional $U_A(1)$ symmetry is explicitly broken by quantum effects. This is due to the presence of an Adler-Bell-Jackiw anomaly, which is proportional to the non–Abelian field strength tensor. This anomaly prevents the restoration of the $U_A(1)$ symmetry, even in the massless continuum limit. In reality the explicit breaking of the symmetry is due to light quarks propagating by zero modes which in turn arises from instantons which themselves describe some tunnelling effect between different physical states. Since the $U_A(1)$ anomaly is proportional to the non–Abelian field strength tensor it is therefore also proportional to the topological charge density, and hence the topological charge of a gauge field configuration in the gluonic sector. The topological charge can be related to the number of chiral zero modes via the Atiyah–Singer index theorem [34]. The Atiyah–Singer index theorem states that the difference between the number of positive chirality modes and the negatives ones of the Dirac operator in an external colour gauge field is equal to the topological charge. Firstly, as a physical consequence of this explicit symmetry breaking, the $U_A(1)$ axial symmetry is not realized at the quantum level, which is why it is an anomaly, hence implying that a more realistic continuous chiral symmetry is $U_V(1)\otimes SU_V(3)\otimes SU_A(3)$. Secondly, its spontaneous breaking to $U_V(1)\otimes SU_V(3)$ at small u and d quark masses give rises to the existence of eight low mass quasi–Goldstone bosons(one for each broken generator in the symmetry group). The ninth pseudoscalar meson, the η' does not become a Goldstone boson if the quark masses are put to zero. In nature the η' (about $m_{\eta'} = 958$MeV) is a lot more massive than the $\pi-$, $K-$ and the $\eta-$ mesons. Based on the limit of a large number of colours ($N_c \longrightarrow \infty$) argument, the Witten–Veneziano formula [35, 36] relates the η' mass to the topological susceptibility in a pure gluon sector and in the quenched approximation

$$\chi = \int \frac{d^4x}{\mathrm{Vol}} \langle Q(x)Q(0)\rangle = \frac{f_\pi^2}{2N_f}(m_\eta^2 + m_{\eta'}^2 - 2m_K^2), \tag{7.1}$$

leading to the expectation that $\chi \approx (180\,\mathrm{MeV})^4$.

The topological susceptibility has to be calculated in the pure gauge theory such as $SU_c(3)$. The difficulty in the numerical evaluation of the topological susceptibility lies in the lattice definition of the topological charge Q. A straightforward transcription of $Q = (g^2/32\pi^2)\epsilon_{\mu\nu\rho\sigma}\int d^4x F^a_{\mu\nu}(x)F^a_{\rho\sigma}(x)$ between the continuum into the lattice suffers from very large renormalizations [37, 38] which are difficult to control without the help of supplementary techniques which drive this renormalization factor down to one. There are various ways to control the large renormalization errors. One of them is the geometrical method [39] which is based on the interpolation of the colour gauge field obtained from the link variables. This method is highly non local. The dislocations are short distance lattice artifacts that after a certain amount of iterations spoil the continuum limit. The cooling method [40] overcomes the problem encountered with the geometrical method. Cooling methods smooth out the short range quantum fluctuations. The effect of cooling can be illustrated by graphing the action density. For example, in Fig. 7.1, we can see that the topological structure of the gauge field is slowly revealed as the quantum fluctuations are removed and as the cooling procedure evolves. In Fig. 7.1, a two loop improved cooling algorithm was used, separated by fifty sweeps of cooling. Once these short range quantum fluctuations are removed it is possible to measure a sensible integer like topological charge after a given number of iterations. In Fig. 7.2, the evolution of the topological charge density after 50 intermediate sweeps of improved cooling is illustrated. This method appears to be the most robust and reliable method to calculate such observables on the lattice. This method still remain non local and questions remains about how much of the physics is lost as well as the coherency of the physics observed after such iteration. This question cannot be resolved until the physical scale is well separated from the scale of the lattice spacing.

A similar method in smoothing out the short range quantum fluctuations is found using the smearing method. This method is operating directly in $SU_c(N)$ by performing a projection onto the $SU_c(N)$ via a linear combination involving the original link variable and the product of the staples dagger. This method is commonly known as APE smearing [41].

In the following sections I present a method that improves the cooling method and construct an improved topological charge operator based on the product of link variables forming rectangular Wilson loops. Using these operators we can resolve an integer like topological charge and perform a comparative study on the relative performance of the cooling and smearing techniques.

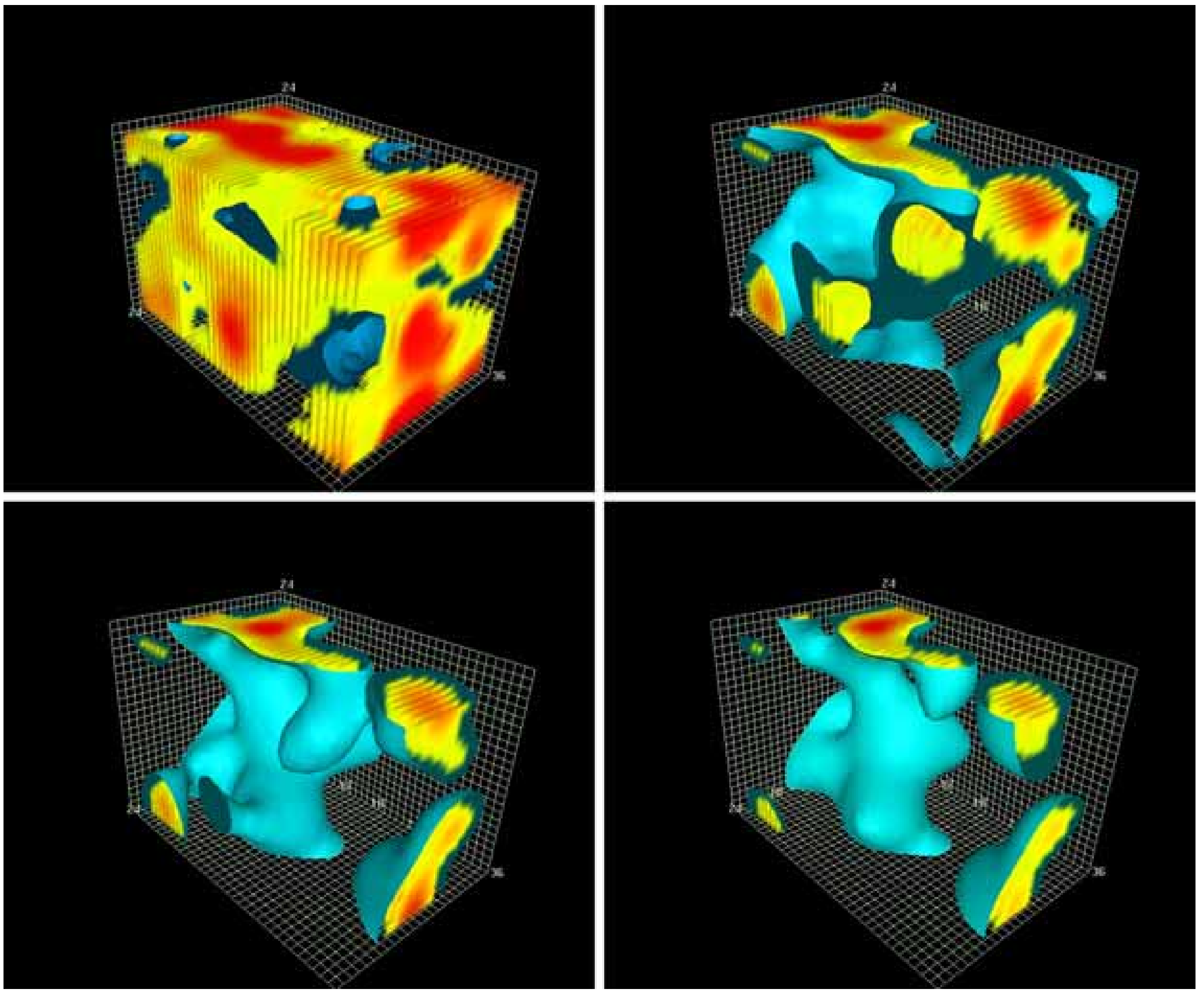


Figure 7.1: Graphical representation of the action density after 50,100,150 and 200 sweeps of improved cooling on a $24^3 \times 36$ lattice at $\beta = 5.00$.

7.1 Topological Charge Density on the Lattice and Instantons

In Appendix B.1 I review the mathematical background behind the concept of the topological charge of a gauge field and why we expect an integer value for the toplogical charge as well as showing how it relates to the gluonic action. As we saw in Eq. (2.12), the fundamental QCD action for the gluons in Euclidean space is defined as

$$S_G = \frac{1}{4} \int d^4x F_{\mu\nu}(x) F_{\mu\nu}(x). \tag{7.2}$$

This Euclidean action is finite only if $F_{\mu\nu} \longrightarrow 0$ as the Euclidean length $|x| \longrightarrow \infty$, meaning that the gauge field are vanishing at infinity. So the fields are the gauge transform of a null field, and can be considered to be pure gauge fields. In this situation they have the asymptotic $g^{-1}(x)\partial_\mu g(x)$ as $x \longrightarrow \infty$, where $g(x) \in SU_c(3)$. In $SU_c(2)$ such functions would take the form of $U(x) = a_0(x)I + \vec{a}\cdot\vec{\sigma}$.

To each field A_μ that has this asymptotic behaviour we can assign a homotopy class of maps from the sphere of very large radius into the gauge group, the class being realized by the function $g(x)$.

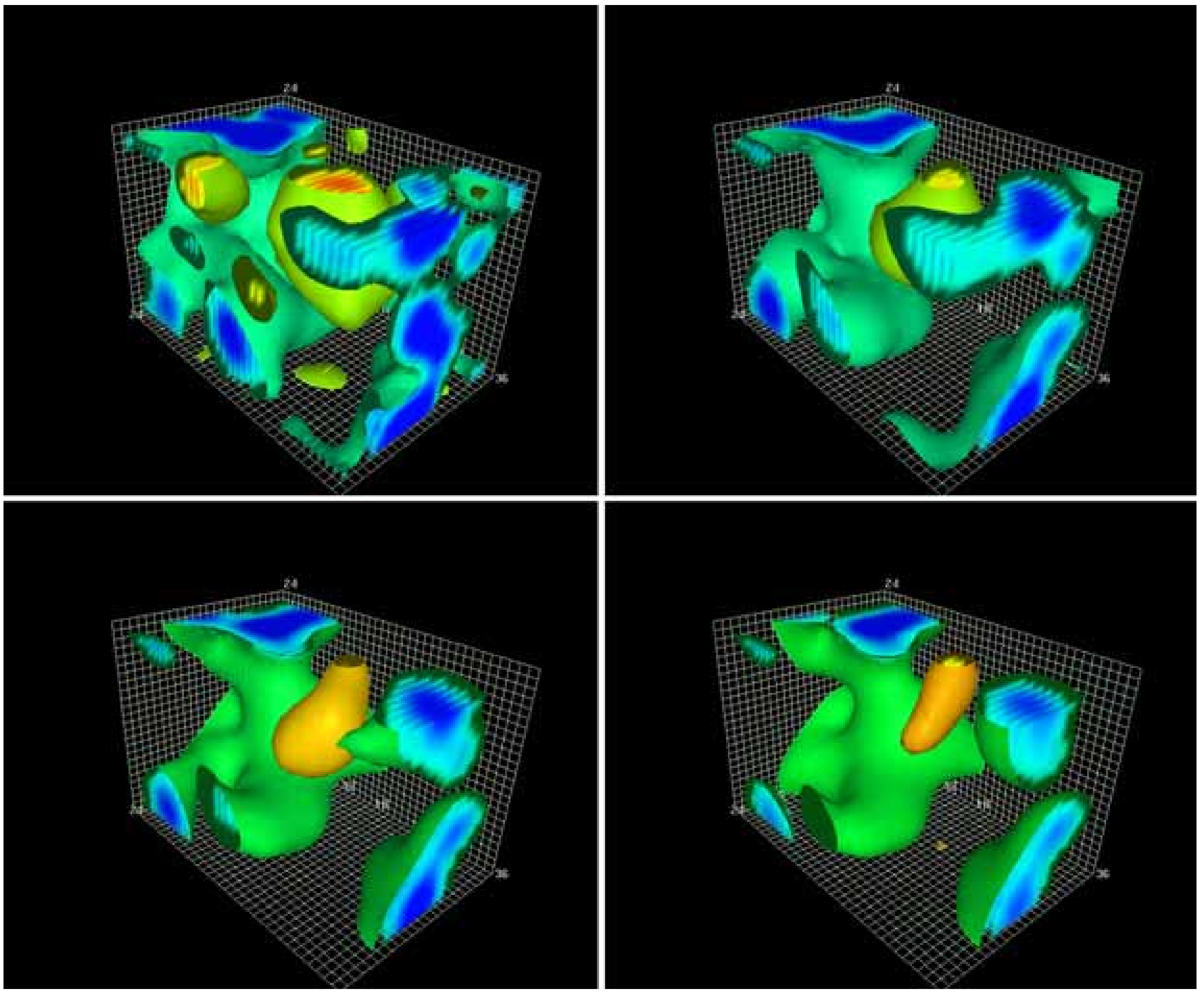


Figure 7.2: Graphical representation of the 2 loop improved topological charge density after 50,100,150 and 200 sweeps of improved cooling on a $24^3 \times 36$ lattice at $\beta = 5.00$. The configuration is the same as in Fig. 7.1. Here the topological charge is -4.

Thus a pure gauge field with a finite Euclidean action defined as in Eq.(7.2), has an associated integer, its topological charge [42]. The topological charge, q, of a pure gauge field A_μ, is usually expressed as the integral over the Euclidean space–time of a topological charge density, $Q(x)$,

$$q = \int d^4x Q(x) = \frac{g^2}{16\pi^2} \int d^4x \mathrm{Tr}(\widetilde{F}_{\mu\nu}(x) F_{\mu\nu}(x)) \quad \text{where} \quad \widetilde{F}_{\mu\nu} = \frac{1}{2}\epsilon_{\mu\nu\rho\sigma} F_{\rho\sigma}. \quad (7.3)$$

The rank two tensor $\widetilde{F}_{\mu\nu}$ is the anti–symmetric tensor dual to $F_{\rho\sigma}$. From Eq.(B.9), we can see the set as being countably infinite, in other words the QCD vacuum is not made up of only one degenerate vacuum but an infinite number of degenerate vacua each of which are characterized by the topological charge number. The local minimum of the Euclidean action, Eq.(7.2), are called instantons [45]. Instantons are what connects the vacua together separated by q units of the topological quantum number between $t = -\infty$ and $t = +\infty$. In other words instantons corresponds to tunnelling between different vacua with different topological charge q. In the semi–classical limit the topological charge can also be defined by the number of instantons n_I minus the number of anti–instantons $\overline{n}_I$, i.e. $q = n_I - \overline{n}_I$. The minimum action

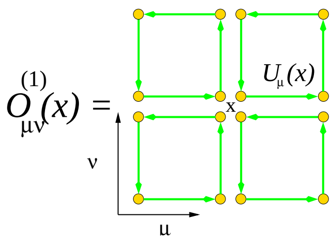


Figure 7.3: Graphical representation of the topological charge operator.

corresponding to instantons, from Eq.(7.2), is $S^0_G = 8\pi^2|q|/g^2$. This is realized when $F_{\mu\nu}$ are self/anti-self–dual depending if $q = \pm 1$ respectively, i.e. $\widetilde{F}_{\mu\nu} = \pm F_{\mu\nu}$ for $q = \pm 1$ respectively.

7.1.1 Topological Charge Operator on the lattice

We construct the lattice topological charge density operator analogous to the standard Wilson action via the clover definition of $F_{\mu\nu}$. The topological charge Q_L and the topological charge density $q_\mathrm{L}(x)$ are defined as follows

$$Q_\mathrm{L} = \sum_x q_\mathrm{L}(x) = \frac{g^2}{32\pi^2}\epsilon_{\mu\nu\sigma\rho}\sum_x \mathrm{Tr}\left(F_{\mu\nu}(x)F_{\sigma\rho}(x)\right). \tag{7.4}$$

The non–Abelian field strength tensor can be expressed as

$$a^2 g F_{\mu\nu}(x) \;=\; \frac{-i}{8}\left[\left(\mathcal{O}^{(i)}_{\mu\nu}(x) - {\mathcal{O}^{(i)}}^{\dagger}_{\mu\nu}(x)\right) - \frac{1}{3}\mathrm{Tr}\left(\mathcal{O}^{(i)}_{\mu\nu}(x) - {\mathcal{O}^{(i)}}^{\dagger}_{\mu\nu}(x)\right)\right]. \tag{7.5}$$

The operator $\mathcal{O}^{(i)}_{\mu\nu}(x)$ is constructed from the product of gauge links. The dummy index, i, is just for notation purposes.

The ability for Q_L to converge to integer value really depends on the definition of $\mathcal{O}^{(i)}_{\mu\nu}(x)$. When $i = 1$, we define $\mathcal{O}^{(1)}_{\mu\nu}(x)$ as the sum of the 1×1 plaquettes loop shown in Fig. 7.3, which can analytically be written as:

$$\begin{aligned}
\mathcal{O}^{(1)}_{\mu\nu}(x) \;=\;& U_\mu(x)\,U_\nu(x+\hat\mu)\,U^\dagger_\mu(x+\hat\nu)\,U^\dagger_\nu(x)\\
+\;& U_\nu(x)\,U^\dagger_\mu(x+\hat\nu-\hat\mu)\,U^\dagger_\nu(x-\hat\mu)\,U_\mu(x-\hat\mu)\\
+\;& U_\mu(x-\hat\mu)\,U^\dagger_\nu(x-\hat\mu-\hat\nu)\,U_\mu(x-\hat\mu-\hat\nu)\,U_\nu(x-\hat\nu)\\
+\;& U^\dagger_\nu(x-\hat\nu)\,U_\mu(x-\hat\nu)\,U_\nu(x+\hat\mu-\hat\nu)\,U^\dagger_\mu(x)\,.
\end{aligned} \tag{7.6}$$

This definition for $F_{\mu\nu}(x)$ is usually the one that is inserted in the Clover term in the Sheikoleslami–Wholert [46] improved quark action. We will define the resulting lattice topological charge as Q_L.

Lattice operators possess a multiplicative lattice renormalization factor, $Q_L = \mathcal{Z}_Q(\beta)\,Q$ which relates the lattice quantity Q_L to the continuum quantity Q. Perturbative calculations indicate $\mathcal{Z}_Q(\beta) \approx 1 - 5.451/\beta + \mathcal{O}(1/\beta^2)$ [38]. This large renormalization causes a problem when one is working at $\beta \approx 6.0$, as $\mathcal{Z}_Q(\beta) \ll 1$. This implies that the topological charge is almost impossible to calculate directly. However when one applies cooling or smearing techniques to remove the problem of the short range quantum fluctuations giving rise to $\mathcal{Z}_Q(\beta) \ll 1$, one can resolve near integer topological charge. One can apply the operator Q_L to cooled configurations or smeared configurations. The latter case may be regarded as employing an improved operator in which the smeared links are understood to give rise to additional higher dimension irrelevant operators designed to provide a smooth approach to the continuum limit. However, even after a significant amount of smoothing it is difficult to resolve perfect integer value topological charge. Alternative definition is therefore necessary.

We define our improved topological charge operator, $Q_{\rm L}^{\rm Imp}$, using

$$\mathcal{O}^{(2)}_{\mu\nu}(x) = c_1 \mathcal{O}^{(1)}_{\mu\nu}(x) + \frac{c_2}{u_0^2}\mathcal{I}^{(2)}_{\mu\nu}(x), \tag{7.7}$$

where $\mathcal{I}^{(2)}_{\mu\nu}(x)$, depicted in Fig. 7.4, is the 2×1 and 1×2 Wilson loop improvement

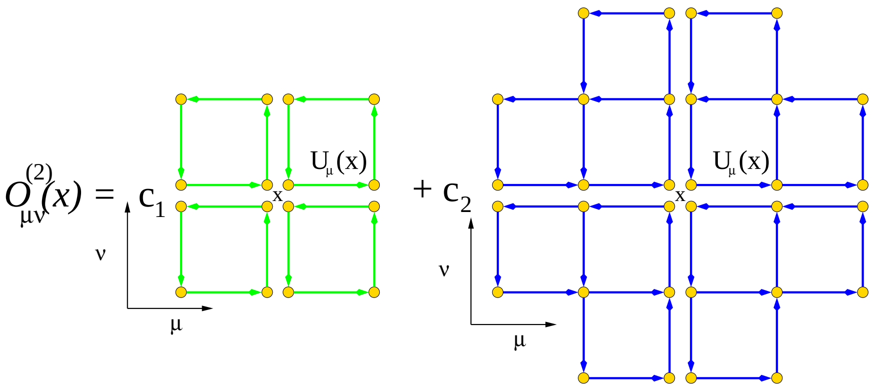


Figure 7.4: Graphical representation of the improved topological charge operator.

operator. This operator can also be expressed in terms of link variables as:

$$\begin{aligned}
\mathcal{I}^{(2)}_{\mu\nu}(x) &= U_\mu(x)U_\mu(x+\hat\mu)U_\nu(x+2\hat\mu)U^\dagger_\mu(x+\hat\mu+\hat\nu)U^\dagger_\mu(x+\hat\nu)U^\dagger_\nu(x)\\
&+ U_\mu(x)U_\nu(x+\hat\mu)U_\nu(x+\hat\mu+\hat\nu)U^\dagger_\mu(x+2\hat\nu)U^\dagger_\nu(x+\hat\nu)U^\dagger_\nu(x)\\
&+ U_\nu(x)U_\nu(x+\hat\nu)U^\dagger_\mu(x-\hat\mu+2\hat\nu)U^\dagger_\nu(x-\hat\mu+\hat\nu)U^\dagger_\nu(x-\hat\mu)U^\dagger_\mu(x-\hat\mu)\\
&+ U_\nu(x)U^\dagger_\mu(x-\hat\mu+\hat\nu)U^\dagger_\mu(x-2\hat\mu+\hat\nu)U^\dagger_\nu(x-2\hat\mu)U_\mu(x-2\hat\mu)U_\mu(x-\hat\mu)\\
&+ U^\dagger_\mu(x-\hat\mu)U^\dagger_\mu(x-2\hat\mu)U^\dagger_\nu(x-2\hat\mu-\hat\nu)U_\mu(x-2\hat\mu-\hat\nu)U_\mu(x-\hat\mu-\hat\nu)U_\nu(x-\hat\nu)\\
&+ U^\dagger_\mu(x-\hat\mu)U^\dagger_\nu(x-\hat\mu-\hat\nu)U^\dagger_\nu(x-\hat\mu-2\hat\nu)U_\mu(x-\hat\mu-2\hat\nu)U_\nu(x-2\hat\nu)U_\nu(x-\hat\nu)\\
&+ U^\dagger_\nu(x-\hat\nu)U_\mu(x-\hat\nu)U_\mu(x+\hat\mu-\hat\nu)U_\nu(x+2\hat\mu-\hat\nu)U^\dagger_\mu(x+\hat\mu)U^\dagger_\mu(x)\\
&+ U^\dagger_\nu(x-\hat\nu)U^\dagger_\nu(x-2\hat\nu)U_\mu(x-2\hat\nu)U_\nu(x+\hat\mu-2\hat\nu)U_\nu(x+\hat\mu-\hat\nu)U^\dagger_\mu(x).
\end{aligned} \tag{7.8}$$

The coefficients c_1 and c_2 are the improvement coefficients which need to be extracted from the path ordered Wilson loop expansion (this is done in Section 7.1.1). The tadepole coefficient u_0 is defined in Eq. (5.19), and was introduced for the tadepole corrections it is of crucial importance in the numerical simulation. It approaches 1 as the number of smoothing sweeps approaches a large value.

Extracting The Coefficients For The Improved Topological Charge Operator

To extract the improvement coefficients one needs to expand the Path ordered Wilson loop as in Section 2.1.3. The Wilson loop $\mathcal{C}_{\mu\nu}$ is defined by Eq. (2.15), where the contour integral is evaluated using Stoke's Theorem as in Eq. (2.16). Using the same boundaries for the contour integral as in Sec. 5.1, one obtains for the 1×1 Wilson contour and $2\times 1, 1\times 2$ improved contours we have respectively the following

$$\oint_{1\times1} A(x)\cdot dx = a^2 F_{\mu\nu}(x_0) + \frac{a^4}{24}\left(D_\mu^2 + D_\nu^2\right) F_{\mu\nu}(x_0) + \dots\,, \tag{7.9}$$

$$\oint_{2\times1} A(x)\cdot dx = 2a^2 F_{\mu\nu}(x_0) + \frac{a^4}{12}\left(4D_\mu^2 + D_\nu^2\right) F_{\mu\nu}(x_0) + \dots\,, \tag{7.10}$$

$$\oint_{1\times2} A(x)\cdot dx = 2a^2 F_{\mu\nu}(x_0) + \frac{a^4}{12}\left(D_\mu^2 + 4D_\nu^2\right) F_{\mu\nu}(x_0) + \dots\,. \tag{7.11}$$

One could easily introduce more Wilson loops to do further improvement and calculate expressions similar to Eq. (7.9,7.10,7.11) by adapting the integration bounds according to the expansion point x_0 centred inside the Wilson loop [30].

Extracting the improvement coefficients for the action, $S_{\rm G}[U]$, is done by considering the real part of $\mathcal{C}_{\mu\nu}(x)$. This implies that the first non–vanishing term in Eq. (2.15) is the one of second order in the coupling, g. Squaring Eqs.(7.9,7.10,7.11) will give three equations, Eqs. (5.10,5.11,5.12), in terms of $F_{\mu\nu}(x_0)$ from which the improvement coefficients can be calculated by taking linear combinations of these operators. These coefficients are already known in the case of a 2×1 and 1×2 improved action [11]. In the case of the topological charge operator, one needs to consider the imaginary part of the contour operator, $\mathcal{C}_{\mu\nu}(x)$. From Eqs.(7.9,7.10,7.11) it is observed that

$$a^2 F_{\mu\nu}(x_0) = \frac{5}{3}\left[\oint_{1\times1} A(x)\cdot dx\right] - \frac{1}{6}\left[\oint_{2\times1} A(x)\cdot dx + \oint_{1\times2} A(x)\cdot dx\right], \tag{7.12}$$

which implies that the improved coefficients for the 2×1 and 1×2 improved topological charge operator, Eq. (7.7), are:

$$c_1 = \frac{5}{3}, \qquad \text{and} \qquad c_2 = -\frac{1}{6}. \tag{7.13}$$

7.2 Gauge Action

In this chapter, the analysis is based on both gauge actions. The Standard Wilson action defined in Eq. (2.19) is

$$S_G = \beta \sum_x \sum_{\mu<\nu} \frac{1}{3} \Re e\, \mathrm{Tr}(1 - P_{\mu\nu}(x)), \tag{7.14}$$

where the operator $P_{\mu\nu}(x)$ is the standard plaquette operator:

$$P_{\mu\nu}(x) = U_\mu(x)\, U_\nu(x+\hat\mu)\, U_\mu^\dagger(x+\hat\nu)\, U_\nu^\dagger(x)\,. \tag{7.15}$$

The tree–level $\mathcal{O}(a^2)$–improved action was defined in Eq. (5.16) and is

$$S_G = \frac{5\beta}{3}\sum_{\rm sq} \Re e\, \mathrm{Tr}(1 - U_\mu(x)\widetilde{\Sigma}_{\rm sq}(x)) - \frac{\beta}{12u_0^2}\sum_{\rm rect} \Re e\, \mathrm{Tr}(1 - U_\mu(x)\widetilde{\Sigma}_{\rm rect}(x)). \tag{7.16}$$

The gauge field configurations are generated using the algorithm described in Chapter 4 with three diagonal $SU_c(2)$ subgroups cycled twice. Simulations are performed using a parallel algorithm on a Thinking Machines Corporations (TMC) CM-5 with appropriate link partitioning.

For standard Wilson and the Improved action the link variables are partitioned according to the algorithm described in Sec. 4.2.2 and Sec. 5.2.2 respectively.

Configurations are generated on a $16^3 \times 32$ lattice at $\beta = 4.38$ and a $24^3 \times 36$ lattice at $\beta = 5.00$, for the improved gauge glue. Similarly for the standard glue with couplings $\beta = 5.70$ and $\beta = 6.00$. Configurations are selected after 5000 thermalization sweeps from a cold start, and every 500 sweeps thereafter with a fixed mean link value. Lattice parameters are summarized in Table 7.1.

7.3 Cooling

Standard cooling minimizes the action locally at each link update. The preferred algorithm is based on the Cabbibo-Marinari [21] pseudo-heat-bath algorithm for constructing $SU_c(3)$-colour gauge configurations. A brief summary of a link update is as follows:

At the $SU_c(2)$ level the algorithm is transparent. An element of $SU_c(2)$ may be parameterized as, $U = a_0 I + i\,\vec{a}\cdot\vec{\sigma}$, where a is real and $a^2 = 1$. Let $\widetilde{U}_\mu$ be one of the six staples associated with creating the plaquette associated with a link U_μ.

$$\widetilde{U}_\mu = U_\nu(x+\hat{\mu})\, U_\mu^\dagger(x+\hat{\nu})\, U_\nu^\dagger(x)\,. \tag{7.17}$$

We define

$$\sum_{\alpha=1}^{6} \widetilde{U}_\alpha = k\overline{U}, \qquad \text{where} \qquad \overline{U} \in SU_c(2), \quad \text{and} \quad k^2 \equiv \det\left(\sum_{\alpha=1}^{6} \widetilde{U}_\alpha\right). \tag{7.18}$$

The feature of the sum of $SU_c(2)$ elements being proportional to an $SU_c(2)$ element is central to the algorithm. The local $SU_c(2)$ action is proportional to

$$\Re e\, \mathrm{Tr}(1 - U\overline{U})\,, \tag{7.19}$$

and is locally minimized when $\Re e\,\mathrm{Tr}(U\overline{U})$ is maximized, i.e. when

$$\Re e\,\mathrm{Tr}(U\overline{U}) = \Re e\,\mathrm{Tr}(I)\,. \tag{7.20}$$

which requires the link to be updated as

$$U \longrightarrow U' = \overline{U}^{-1} = \overline{U}^\dagger = \frac{\left(\sum_{\alpha=1}^{6} \widetilde{U_\alpha}\right)^\dagger}{k}\,. \tag{7.21}$$

Table 7.1: Parameters of the numerical simulations.

Action	Volume	N_{Therm}	N_{Samp}	β	a fm	u_0	Physical Volume fm
Wilson	$16^3 \times 32$	5000	1000	5.70	0.18	0.86085	$2.88^3 \times 5.76$
Wilson	$24^3 \times 36$	5000	1000	6.00	0.10	0.87777	$2.4^3 \times 3.6$
Improved	$16^3 \times 32$	5000	1000	4.38	0.165(2)	0.87614	$2.88^3 \times 5.76$
Improved	$24^3 \times 36$	5000	1000	5.00	0.077(1)	0.90286	$2.4^3 \times 3.6$

At the $SU_c(3)$ level, one successively applies this algorithm to various $SU_c(2)$ subgroups of $SU_c(3)$, with $SU_c(2)$ subgroups selected to cover the $SU_c(3)$ gauge group.

We explored cooling gauge field configurations using two diagonal $SU_c(2)$ subgroups. The cooling rate is slow and the action density is not smooth, even after 50 sweeps over the lattice. Addition of the third diagonal $SU_c(2)$ subgroup provides acceptably fast cooling and a smooth action density. We will see in Sec. 7.8.1 in more details how the number of $SU_c(2)$ subgroups influence the gauge group. These $SU_c(2)$ subgroups are acting as a covering group for the $SU_c(3)$ gauge group. One would therefore expect that repeated updates of $SU_c(2)$ subgroups will provide a better update for the ultimate $SU_c(3)$ link.

7.4 Improved Cooling

Our algorithm is based on the Cabbibo–Marinari [21] pseudo–heat–bath algorithm. This algorithm constructs the $SU_c(3)$ link variable, $U_\mu(x)$, according to the Boltzmann probability distribution, using newly generated $SU_c(2)$ subgroups, $\overline{V}_m(x)$.

The subscript, m, represents the position of the diagonal $SU_c(2)$ subgroup inside an $SU_c(3)$ matrix, $a_m(x)$. These subgroups are located at the $m+i, m+j$ matrix entries for $i, j = 0, 1$ for the first and second diagonal $SU_c(2)$ subgroup, i.e. $m = 1$ and $m = 2$ respectively. The third diagonal subgroup is located at the entries $(12, 21, 23, 32)$ of the $SU_c(3)$ matrix, $a_m(x)$.

The improved cooling algorithm, is itself based on the standard cooling algorithm. As in standard cooling, improved cooling also minimizes the action locally at each link update.

The local $SU_c(3)$ standard Wilson action is proportional to

$$S_G[U] = \beta \sum_{\text{sq}} \Re e\, \text{Tr}(1 - U_\mu(x)\widetilde{\Sigma}_{\text{sq}}(x)). \tag{7.22}$$

Where $\widetilde{\Sigma}_{\text{sq}}(x)$ is defined by Eq. (5.17), with $\sum_{\text{sq}} \equiv \sum_{x,\mu\neq\nu}$. This action is locally minimized when $\Re e\, \text{Tr}(U_\mu(x)\widetilde{\Sigma}_{\text{sq}}(x))$ is maximized, i.e. when

$$\Re e\, \text{Tr}(U_\mu(x)\widetilde{\Sigma}_{\text{sq}}(x)) = \Re e\, \text{Tr}(I), \tag{7.23}$$

which really means replacing the original link by the link $U_\mu(x)$ which optimizes

$$\max \Re e\, \text{Tr}\left(U_\mu(x) \sum_{\text{sq}} \widetilde{\Sigma}_{sq}(x)\right). \tag{7.24}$$

Using the existing $SU_c(3)$ link variable, $U_\mu(x)$, we can construct the staples,

$$\widetilde{\Sigma}(x;\mu) \;=\; \sum_{\mu\neq\nu}^{4} \widetilde{U}_{\mu\nu}(x). \tag{7.25}$$

Here $\widetilde{U}_{\mu\nu}(x)$ represents two of the 1×1 plaquette contained in a particular plane pointing in the ν direction out of the three possible planes on the hypercubic lattice,

$$\widetilde{U}_{\mu\nu}(x) = U_\nu(x+\hat{\mu})U_\mu^\dagger(x+\hat{\nu})U_\nu^\dagger(x) + U_\nu^\dagger(x+\hat{\mu}-\hat{\nu})U_\mu^\dagger(x-\hat{\nu})U_\nu(x-\hat{\nu}). \tag{7.26}$$

It can be shown [21], that the action may be written as:

$$\begin{aligned} S[a_m U] &= \Re e\,\mathrm{Tr}(a_m U_\mu(x)\widetilde{\Sigma}(x;\mu)) \\ &= \Re e\,\mathrm{Tr}(\overline{V}_m(x)V_m(x)) + \text{terms independent of } \overline{V}_m(x). \end{aligned} \tag{7.27}$$

Here $V_m(x)$ is a 2×2 sub–matrix extracted from the product $U_\mu(x)\widetilde{\Sigma}(x;\mu)$ and its entries are at the same place as those of the $SU_c(3)$ matrix, $a_m(x)$. The matrix $\overline{V}_m(x)$ is in $SU_c(2)$.

At the $SU_c(2)$ level the method becomes trivial since an element of $SU_c(2)$ may be parameterized as, $\overline{V}_m(x) = b_0(x)I + i\,\vec{b}(x)\cdot\vec{\sigma}$, where the components of the 4–vector, $b(x)$, are real numbers. Since $\overline{V}_m(x)$ is an $SU_c(2)$ element the constraint on $b^2(x)$ is as such that $b^2(x) = 1$, implying that $b(x)$ resides on the 3–sphere. Furthermore, an important property of the $SU_c(2)$ topological group, is that the sum of $SU_c(2)$ elements are proportional to another $SU_c(2)$ element. It is then possible to write $V_m(x)$ as:

$$V_m(x) = k\overline{V}_m(x), \quad \text{where} \quad \overline{V}_m(x) \in SU_c(2), \text{ and } k^2 \equiv \det\left(V_m(x)\right) \in R\,. \tag{7.28}$$

Now we see that by making the transformation in Eq. (7.27),

$$\overline{V}_m(x) \longrightarrow \overline{V}'_m(x) = \frac{V^\dagger_m(x)}{\sqrt{\det(V_m(x))}} = \frac{\overline{V}^\dagger_m(x)}{k}, \tag{7.29}$$

the action is minimized. This transformation is performed for every value of m corresponding to the various $SU_c(2)$ subgroups being selected consistently to cover the $SU_c(3)$ gauge group. It was found consideration of the three diagonal $SU_c(2)$ subgroups looped over twice were optimal. Round off errors in additional loops can actually increase the action.

The improved cooling method is defined when the operator $\widetilde{U}_{\mu\nu}(x)$, Eq. (7.26), is replaced by an operator based on 1×2 and 2×1 loop additional closed contours. This action based improved cooling scheme is also $\mathcal{O}(a^2)$–improved. I then rewrite the staples defined in Eq. (7.25) as:

$$\widetilde{\Sigma}^{(\mathrm{I})}(x;\mu) = \sum_{\mu\neq\nu}^{4} \widetilde{U}^{(\mathrm{I})}_{\mu\nu}(x) = \sum_{\mu\neq\nu}^{4} \frac{5}{3}\widetilde{U}_{\mu\nu}(x) - \frac{1}{12u_0^2}R_{\mu\nu}(x), \tag{7.30}$$

where $R_{\mu\nu}(x)$ is the 1×2 and 2×1 loop contributions,

$$\begin{aligned} R_{\mu\nu}(x) &= U_\nu(x+\hat{\mu})U_\nu(x+\hat{\nu}+\hat{\mu})U^\dagger_\mu(x+2\hat{\nu})U^\dagger_\nu(x+\hat{\nu})U^\dagger_\nu(x) \\ &+ U_\mu(x+\hat{\mu})U_\nu(x+2\hat{\mu})U^\dagger_\mu(x+\hat{\mu}+\hat{\nu})U^\dagger_\mu(x+\hat{\nu})U^\dagger_\nu(x) \\ &+ U_\nu(x+\hat{\mu})U^\dagger_\mu(x+\hat{\nu})U^\dagger_\mu(x+\hat{\nu}-\hat{\mu})U^\dagger_\nu(x-\hat{\mu})U_\mu(x-\hat{\mu}) \\ &+ U^\dagger_\nu(x+\hat{\mu}-\hat{\nu})U^\dagger_\nu(x+\hat{\mu}-2\hat{\nu})U^\dagger_\mu(x-2\hat{\nu})U_\mu(x-2\hat{\nu})U_\mu(x-\hat{\nu}) \\ &+ U_\mu(x+\hat{\mu})U^\dagger_\nu(x+2\hat{\mu}-\hat{\nu})U^\dagger_\mu(x+\hat{\mu}-\hat{\nu})U^\dagger_\mu(x-\hat{\nu})U_\nu(x-\hat{\nu}) \\ &+ U^\dagger_\nu(x+\hat{\mu}-\hat{\nu})U^\dagger_\mu(x-\hat{\nu})U^\dagger_\mu(x-\hat{\nu}-\hat{\mu})U_\nu(x-\hat{\nu}-\hat{\mu})U_\mu(x-\hat{\mu}). \end{aligned} \tag{7.31}$$

The coefficients are based on those of the action, Eq. (5.16), which have already been established in previous work by Lepage *et al.* [11].

The u_0 factor is not held fixed during the improved cooling iteration. Starting from its initial frozen value determined after the thermalization procedure, it is recalculated at every sweep of the iteration. After a few sweeps of improved cooling the value quickly converges to 1 being a good indication that short distance tadepole effects are being removed in the smoothing procedure.

Further work is currently underway in constructing a highly improved scheme [30].

7.5 Smearing

Here we consider two algorithms for smearing the gauge links. APE smearing [41] is now well established. Our alternate algorithm AUS smearing is designed to remove instabilities of the APE algorithm and provide an intermediate algorithm sharing features of both smearing and cooling.

7.5.1 APE Smearing

APE smearing [41] is a gauge equivariant [47] prescription for averaging a link $U_\mu(x)$ with its nearest neighbours $U_\mu(x+\hat{\nu}),\ \nu \neq \mu$. The linear combination takes the form:

$$U_\mu(x) \longrightarrow U'_\mu(x) = (1-\alpha)U_\mu(x) + \frac{\alpha}{6}\widetilde{\Sigma}^\dagger(x;\mu), \tag{7.32}$$

where $\widetilde{\Sigma}(x;\mu) = \left(\sum_{\nu=1}^{6} \widetilde{U_\nu}(x)\right)_\mu$, is the sum of the six staples defined in Eq. (7.17). The parameter α represents the smearing fraction. The algorithm is constructed as follows: (i) calculate the staples, $\widetilde{\Sigma}(x;\mu)$; (ii) calculate the new link variable $U'_\mu(x)$ given by Eq. (7.32) and then reunitarize $U'_\mu(x)$; (iii) once we have performed these steps for every link, the smeared $U'_\mu(x)$ are mapped into the original $U_\mu(x)$. This defines a single APE smearing sweep which can then be repeated.

The celebrated feature of APE smearing is that it can be realized as higher-dimension operators that might appear in a fermion action for example. Indeed, "fat link" actions based on the APE algorithm are excellent candidates for an efficient action with improved chiral properties [48]. The parameter space of fat link actions is described by the number of smearing sweeps n_{ape} and the smearing coefficient α. It is the purpose of this investigation to explore the possible reduction of the dimension of this parameter space from two to one.

7.5.2 AUS Smearing

The Annealed U Smearing (AUS) algorithm is similar to APE smearing in that we take a linear combination of the original link and the associated staples, $\widetilde{\Sigma}^\dagger(x;\mu)$. However in AUS smearing we take what was a single APE smearing sweep over all the links on the lattice and divide it up into four partial sweeps based on the direction of the links. One partial sweep corresponds to an update of all links oriented in one of the four Cartesian directions denoted $\mu = 1,..,4$. Hence in a partial sweep we calculate the reuniterized $U'_\mu(x)$ for all μ–oriented links and update all of these links $[U'_\mu(x) \longrightarrow U_\mu(x)]$ at the end of each partial sweep.

Thus the difference between AUS smearing and APE smearing is that in APE smearing no links are updated until all four partial sweeps are completed, while in AUS smearing, updated smeared information is cycled into the calculation of the next link direction, a process commonly referred to as annealing. In a sense AUS smearing is between cooling and APE smearing in that cooling updates one link at a time, AUS smearing updates one Cartesian direction at a time, and APE smearing updates the whole lattice at the same time.

As for APE smearing, the reunitarization of the $SU_c(3)$ matrix is done using the standard row by row orthonormalization procedure: begin by normalizing the first row; then update the second row by $row2 = row2 - (row2 \cdot row1)row1$; normalize the second row; finally set $row3$ equal to the cross product of $row1$ and $row2$.

For $SU_c(2)$ gauge theory, cooling and smearing are identical for the case of $\alpha = 1$ when the links are updated one at a time. The AUS algorithm changes the level of annealing and provides the opportunity to alter the degree of smoothing. At the same time the AUS algorithm preserves the gauge equivariance of the APE algorithm for $SU_c(3)$ gauge theory [47]. As we shall see, it also removes the instability of the APE algorithm at large smearing fractions α. This latter feature may provide a more efficient method for finding the fundamental modular region of Landau Gauge in trying to understand the effects of Gribov copies in gauge dependent quantities such as the gluon-propagator [49].

7.6 Improved Smearing

In this section we consider one smearing algorithm, APE smearing [41], to smear our gauge links. We extend this algorithm to produce an improved version. Smearing techniques are of great utility in constructing quark actions commonly known as fat–link actions [50].

Improving the smearing operator and comparing them to other smoothing method provides a step forward in enabling one to calculate some observables in lattice QCD as well as constructing an efficient action with improved chiral properties [48].

7.6.1 The Reunitarization of the Links

The reunitarization procedure is of crucial importance when smearing is applied to the gauge links because the APE smearing operation projects the transformed links away from the $SU_c(3)$ gauge group. It is therefore necessary to have a procedure that projects the smeared links back to the gauge group.

Standard APE smearing is gauge equivariant [47] prescription for smearing a link $U_\mu(x)$ with its nearest neighbours $U_\mu(x+\hat{\nu})$, given that $\hat{\nu} \neq \hat{\mu}$. By gauge equivariance we mean that if two starting gauge configurations are related by a gauge transformation then the respective smeared configurations are also related by the same gauge transformation.

The APE smearing process is really an iterative prescription for the link variable, $U_\mu(x)$. The transformation takes the form:

$$\begin{aligned} U_\mu(x) \longrightarrow U'_\mu(x) &= (1-\alpha)U_\mu(x) + \frac{\alpha}{6}\widetilde{\Sigma}^\dagger(x;\mu), && (7.33) \\ U_\mu(x) &= \mathcal{P}U'_\mu(x), && (7.34) \end{aligned}$$

where $\widetilde{\Sigma}(x;\mu)$ is the staples and $\mathcal{P}$ is a projection onto $SU_c(3)$ by Cabbibo Marinari maximization of the $Tr\left[U_\mu(x)U_\mu^{\dagger\prime}(x)\right]$ with two hits on the three diagonal $SU_c(2)$ subgroups. The parameter α represents the smearing fraction. The projection works by first calculating $U'_\mu(x)$ and then taking its hermitian conjugate,

$$U_\mu^{\dagger\prime}(x) = (1-\alpha)U_\mu^\dagger(x) + \frac{\alpha}{6}\widetilde{\Sigma}(x;\mu), \qquad (7.35)$$

which implies that when we take the real part of the trace of the quantity $U_\mu(x)U_\mu^{\dagger\prime}(x)$ we obtain:

$$\begin{aligned} \Re e\,\mathrm{Tr}(U_\mu(x)U_\mu^{\dagger\prime}) &= \frac{\alpha}{6}\Re e\,\mathrm{Tr}(U_\mu(x)\widetilde{\Sigma}(x;\mu)) + \text{constant} && (7.36) \\ &= \frac{\alpha}{6}\Re e\,\mathrm{Tr}(\overline{V}_m(x)V_m(x)) + \text{terms independent of } \overline{V}_m(x). \end{aligned}$$

$\overline{V}_m(x)$ is in $SU_c(2)$ and $V_m(x)$ is a 2×2 sub–matrix extracted from the product $U_\mu(x)\widetilde{\Sigma}_{sq}(x;\mu)$. Now projecting $U'_\mu(x)$ back onto $SU_c(3)$ amounts to finding an $SU_c(3)$ matrix which minimizes the action or equivalently maximizes the quantity $\Re e\,\mathrm{Tr}(U_\mu(x)U_\mu^{\dagger\prime})$. The solution is given by performing the following transformation:

$$\overline{V}_m(x)\longrightarrow\overline{V}'_m(x) = \frac{V_m^\dagger(x)}{\sqrt{\det(V_m(x))}}. \tag{7.37}$$

This method is advantageous because it uses $SU_c(2)$ subgroups to projects back the $U'_\mu(x)$ to $SU_c(3)$.

In this case a clear connection to cooling is established when $\alpha \to 1$ as

$$\max \Re e\,\mathrm{Tr}\left(U_\mu(x)U_\mu'^{\dagger}(x)\right) = \max \Re e\,\mathrm{Tr}\left(U_\mu(x)\sum_{\substack{\nu \\ \nu\neq\mu}}\widetilde{\Sigma}(x;\mu)\right). \tag{7.38}$$

7.6.2 Improved Smearing

The improved smearing algorithm is similar to APE smearing in that we take a linear combination of the original link $U_\mu(x)$ and the associated 1×1 staples $\widetilde{\Sigma}^\dagger(x;\mu)$ just as in Eq. (7.33). However in Improved APE smearing we replace the staples $\widetilde{\Sigma}(x;\mu)$ defined in Eq. (7.25) by the improved staples:

$$\widetilde{\Sigma}^{(\mathrm{I})}(x;\mu) = \sum_{\mu\neq\nu}^{4}\widetilde{U}^{(\mathrm{I})}_{\mu\nu}(x) = \sum_{\mu\neq\nu}^{4}\frac{5}{3}\widetilde{U}_{\mu\nu}(x) - \frac{1}{12u_0^2}R_{\mu\nu}(x). \tag{7.39}$$

The algorithm is constructed as follows: (i) calculate the staples, $\widetilde{\Sigma}^{(\mathrm{I})}(x;\mu)$; (ii) calculate the new link variable $U'_\mu(x)$ given by Eq. (7.33); (iii) once we have performed these steps for every link, the smeared links $U'_\mu(x)$ are projected back to the $SU_c(3)$ gauge group using the projection method $\mathcal{P}$ procedure, Eq. (7.34), described in the previous section (see Section 7.6.1, for more details on that reunitarization method). These three steps define a single improved smearing sweep.

I have compared the results obtained from this method with the standard row by row orthonormalization reunitarization procedure explored in [53], also explained in Sec. 7.5.2. Using the same topological charge operator and performing the same smoothing procedure, I have found a slight reduction in fluctuations of the topological charge in the evolution of gauge fields under APE smearing with projection via Eq. (7.38).

7.7 Numerical Simulations with Standard Wilson Gauge Action: Calibration work

In this first part of the simulation I analysed two sets of gauge field configurations. The two sets are composed of ten $16^3 \times 32$ configurations and five $24^3 \times 36$ configurations at $\beta = 5.70$ and $\beta = 6.00$ respectively. For each configuration I separately performed 200 sweeps of cooling, 200 sweeps of APE smearing at four values of the smearing fraction (α) and 200 sweeps of AUS smearing at six values of the smearing fraction.

For clarity, the number of times an algorithm is applied to the entire lattice is defined as $n_{\rm c}$, $n_{\rm ape}(\alpha)$ and $n_{\rm aus}(\alpha)$ for cooling, APE smearing and AUS smearing respectively. I monitor both the total action normalized to the single instanton action $S_0 = 8\pi^2/g^2$ and the topological charge $Q_{\rm L}$ of Eq. (7.4) and observe their evolution as a function of the appropriate sweep variable and smearing fraction α.

For APE smearing we consider four different values for the smearing fraction α including 0.30, 0.45, 0.55, and 0.70. Larger smearing fractions reveal an instability in the APE algorithm where the links are rendered to noise.

The origin of this instability is easily understood in fat-link perturbation theory [51] where the smeared vector potential after n APE smearing steps is given by

$$A_\mu^{(n+1)}(q) = \sum_\nu \left[f^n(q) \left(\delta_{\mu\nu} - \frac{\widehat{q}_\mu \widehat{q}_\nu}{\widehat{q}^2} \right) + \frac{\widehat{q}_\mu \widehat{q}_\nu}{\widehat{q}^2} \right] A_\nu^{(n)}(q) \, , \tag{7.40}$$

reflecting the transverse nature of APE smearing. Here

$$\widehat{q}_\mu = \frac{2}{a} \sin\left(\frac{a\, q_\mu}{2} \right) , \tag{7.41}$$

and

$$f(q) = 1 - \frac{\alpha}{6} \widehat{q}^2 \, . \tag{7.42}$$

For $f(q)$ to act as a form factor at each vertex in perturbation theory over the entire Brillouin zone

$$-\frac{\pi}{a} < q_\mu \leq \frac{\pi}{a} \, , \tag{7.43}$$

one requires $-1 \leq f(q) \leq 1$ which constrains α to the range $0 \leq \alpha \leq 3/4$.

The annealing process in AUS smearing removes this instability. Hence, the parameter set for AUS smearing consists of the APE set plus an extra two, $\alpha = 0.85$ and 1.00.

7.7.1 Action Analysis

APE and AUS Smearing Calibration

The action normalized to the single instanton action S/S_0 can provide some insight into the number of instantons left in the lattice as a function of the sweep variable for each algorithm. However, the main concern is the relative rate at which the algorithms perform. In Fig. 7.5, I show S/S_0 as a function of cooling sweep on the $24^3 \times 36$ lattice for five different configurations. The close proximity of the five curves is typical of the configuration dependence of the normalized action S/S_0.

Figures 7.6 and 7.7 report results for APE and AUS smearing respectively. Here we focus on one of the five configurations, noting that similar results are found for the other configurations. Each curve corresponds to a different value of the smearing fraction α.

A similar analysis of the $16^3 \times 32$ lattice at $\beta = 5.70$ is also performed yielding analogous results. In fact, taking the physical volumes of the two lattices into account reveals qualitatively similar action densities after 200 sweeps.

Note that in Fig. 7.7, the curve associated with the smearing parameter $\alpha = 1.00$, crosses over the one generated at $\alpha = 0.85$, when the sweep number, $n_{\rm aus}(\alpha)$, is approximately 40 sweeps. Thus $\alpha = 1$ is not the most efficient smearing fraction for

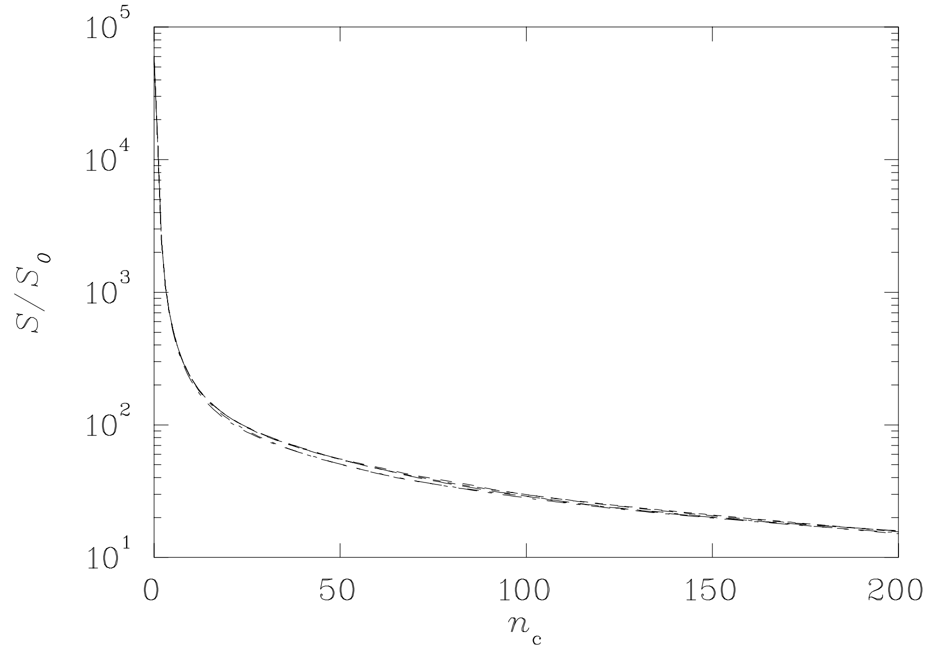


Figure 7.5: The ratio S/S_0 as a function of cooling sweeps n_c for five configurations on the $24^3 \times 36$ lattice at $\beta = 6.00$. The single instanton action is $S_0 = 8\pi^2/g^2$.

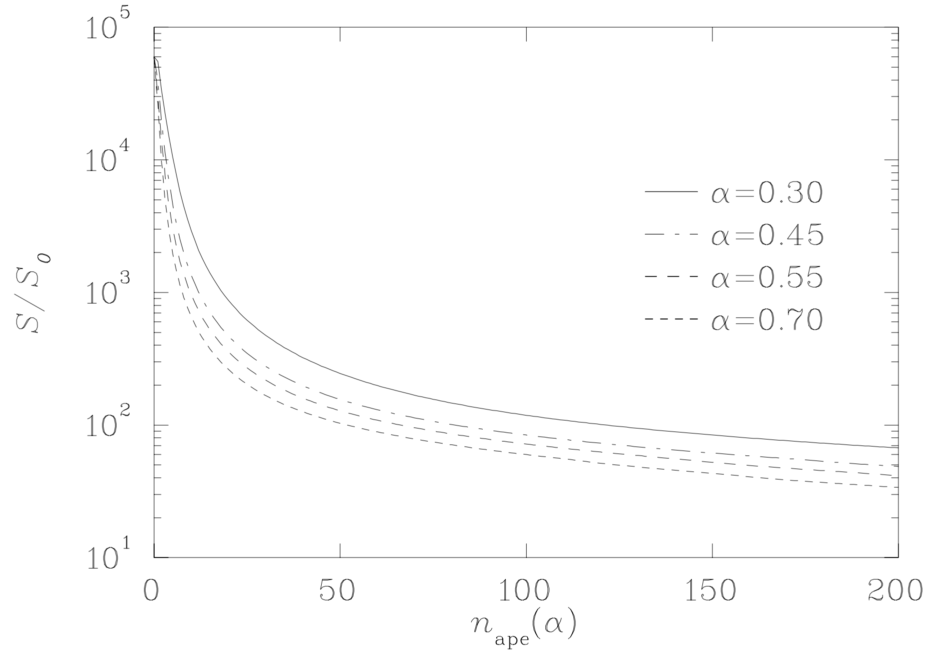


Figure 7.6: The ratio S/S_0 as a function of APE smearing sweeps $n_\mathrm{ape}(\alpha)$ for a configuration on a $24^3 \times 36$ lattice at $\beta = 6.00$. Each curve has an associated smearing fraction α.

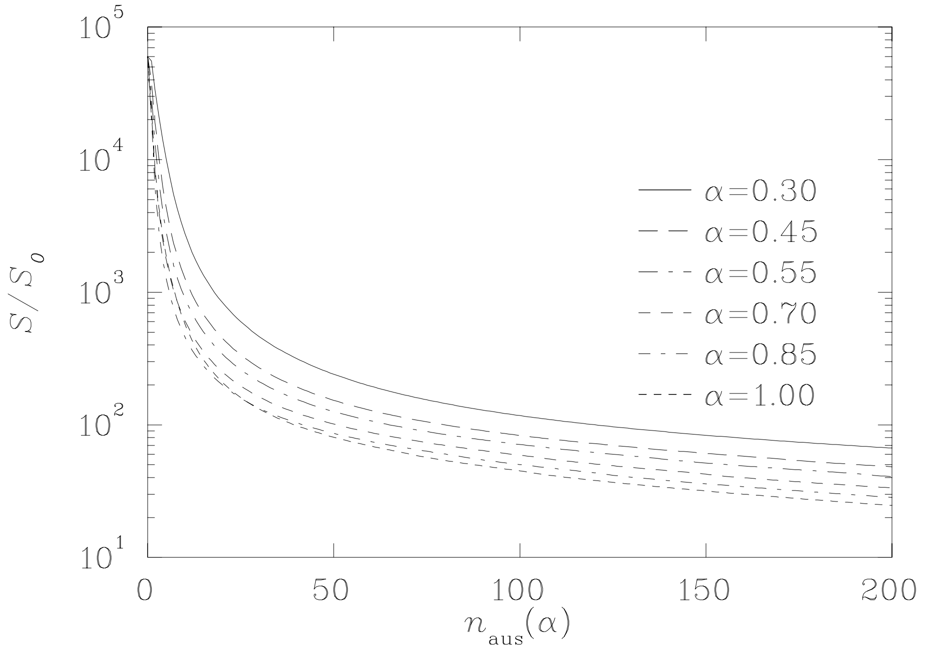


Figure 7.7: The ratio S/S_0 as a function of AUS smearing sweep $n_{\mathrm{aus}}(\alpha)$ for a configuration on a $24^3 \times 36$ lattice at $\beta = 6.00$. Each curve has an associated smearing fraction α.

unsmoothed gauge configurations. However, as the configurations become smooth, $\alpha = 1.00$ becomes the most efficient choice.

To calibrate the rate at which the algorithms reduce the action, I record the number of sweeps $n(\alpha)$ required for the smeared action to cross various thresholds S_T. This is repeated for each of the smearing fractions α under consideration. In establishing the relative α dependence for the number of sweeps $n(\alpha)$, a simple linear relation between the number of sweeps required to cross S_T at one α compared to another α' is first considered, i.e.

$$n(\alpha') = c_0 + c_1\, n(\alpha)\,. \tag{7.44}$$

Anticipating that c_0 will be small if not zero, I divide both sides of this equation by $n(\alpha)$ and plot as a function of $n(\alpha)$. Deviations from a horizontal line at large $n(\alpha)$ will indicate failings of this linear assumption.

Fig. 7.8 displays results for α' fixed to 0.55 for the APE smearing algorithm and Fig. 7.9 displays analogous results for AUS smearing. I omit thresholds that result in $n(0.55) < 10$ as these points will have integer discretization errors exceeding 10%. For $\alpha = 0.70$ discretization errors the of order of 10% are clearly visible in both figures.

For the smearing fraction $\alpha \leq 0.85$ both plots show little dependence of $n(\alpha' = 0.55)/n(\alpha)$ on $n(\alpha)$. This supports our simple ansatz of Eq. (7.44) and indicates c_0 is indeed small as one would expect.

The non-linear behaviour for $\alpha = 1.00$ in AUS smearing, in Fig. 7.9, reflects the cross over in Fig. 7.7, for $\alpha = 1.00$. For unsmeared configurations, $\alpha = 1.00$ appears to be too large. Further study in $SU_c(2)$ is required to resolve whether the origin of the smearing inefficiency lies in the sum of staples lying too far outside the $SU_c(3)$ gauge group for useful reunitarization, or whether the annealing of the links in AUS smearing is insufficient relative to cooling.

To determine the α dependence of c_1, I plot $c_1 = \langle n(\alpha' = 0.55)/n(\alpha)\rangle$ as a function of α. Here the angular brackets denote averaging over all the threshold values

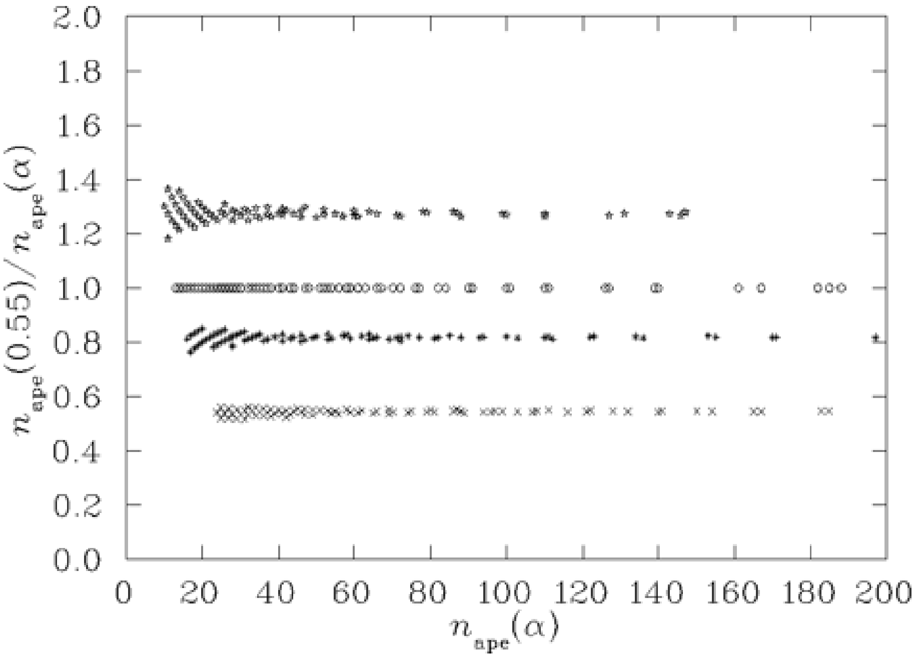


Figure 7.8: The ratio $n_{\text{ape}}(0.50)/n_{\text{ape}}(\alpha)$ versus $n_{\text{ape}}(\alpha)$ for numerous threshold actions on the $24^3 \times 36$ lattice at $\beta = 6.00$. From top down the data points correspond to $\alpha = 0.70$, 0.55, 0.45 and 0.30.

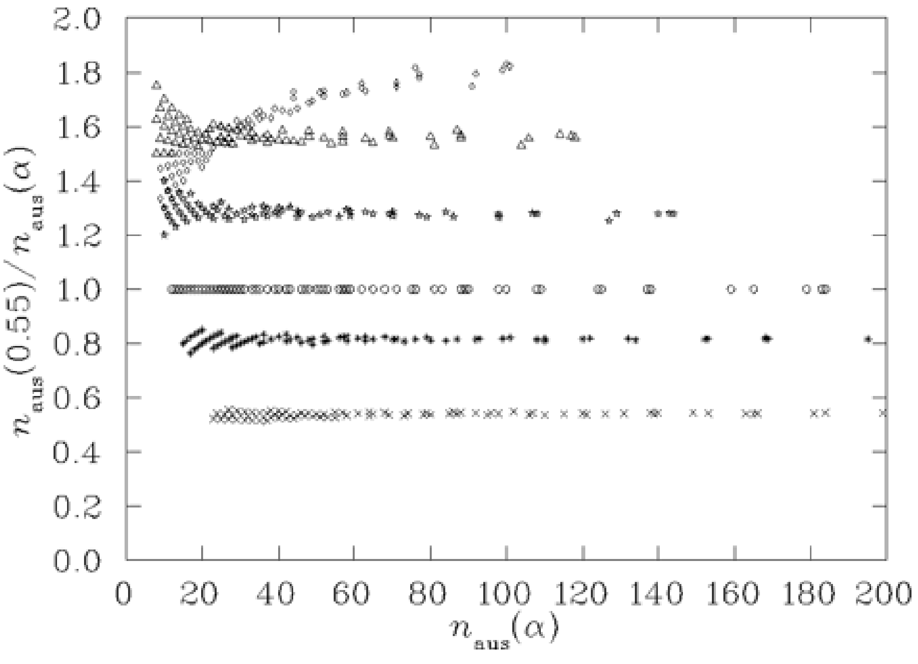


Figure 7.9: The ratio $n_{\text{aus}}(0.55)/n_{\text{aus}}(\alpha)$ versus $n_{\text{aus}}(\alpha)$ for numerous threshold actions on the $24^3 \times 36$ lattice at $\beta = 6.00$. From top down the data points correspond to $\alpha = 1.00$, 0.85, 0.70, 0.55, 0.45, 0.30.

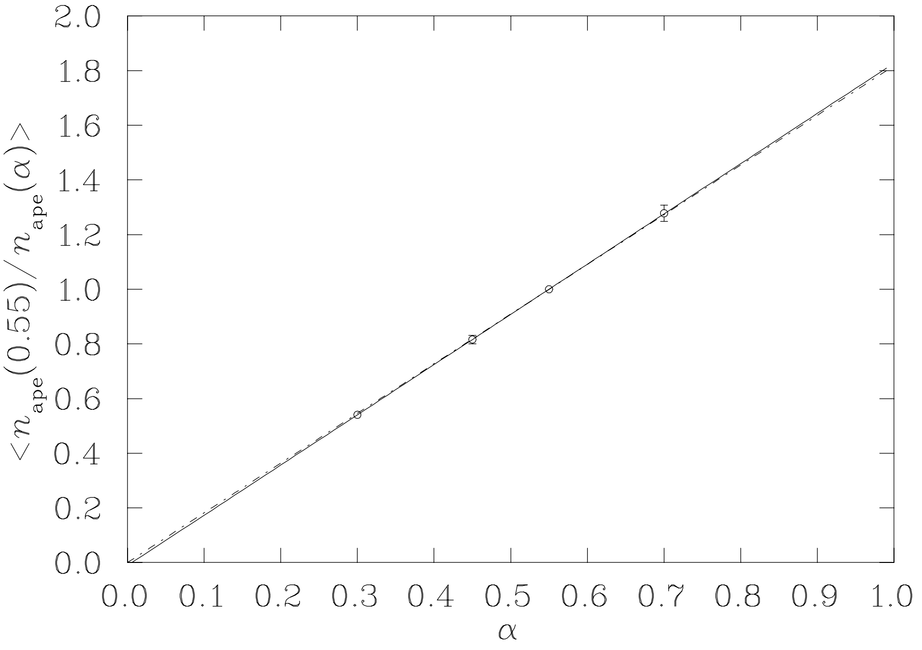


Figure 7.10: Illustration of the α dependence of $c_1 = \langle n(\alpha' = 0.55)/n(\alpha)\rangle$ for APE smearing. The solid line fit to the data indicates $c_1 = 1.838\,\alpha - 0.011$ whereas the dashed line, constrained to pass through the origin, provides a slope of 1.818.

considered; i.e. averaging over data in the horizontal lines of Figs. 7.8 and 7.9.

Fig. 7.10 for APE smearing and Fig. 7.11 for AUS smearing indicate that the relationship between c_1 and α is linear with zero intercept to an excellent approximation. Indeed, ignoring the point at $\alpha = 1$ for AUS smearing, one finds the same coefficients for the α dependence of APE and AUS smearing. When the fits are constrained to pass through the origin, one finds a slope of 1.818 which is the inverse of $\alpha' = 0.55$. Hence we reach the conclusion that

$$\frac{n_{\rm ape}(\alpha')}{n_{\rm ape}(\alpha)} \simeq \frac{\alpha}{\alpha'} \quad \text{and} \quad \frac{n_{\rm aus}(\alpha')}{n_{\rm aus}(\alpha)} \simeq \frac{\alpha}{\alpha'}\,. \tag{7.45}$$

This analysis based on the action suggests that a preferred value for α does not really exist. In fact it has been recently suggested that one should anticipate some latitude in the values for $n(\alpha)$ and α that give rise to effective fat-link actions [51]. What we have done here is established a relationship between $n(\alpha)$ and α, thus reducing what was potentially a two dimensional parameter space to a one dimensional space. This conclusion will be further supported by the topological charge analysis below.

To summarize these finding we plot the ratios

$$\frac{\alpha'\,n_{\rm ape}(\alpha')}{\alpha\,n_{\rm ape}(\alpha)} \quad \text{and} \quad \frac{\alpha'\,n_{\rm aus}(\alpha')}{\alpha\,n_{\rm aus}(\alpha)}\,, \tag{7.46}$$

designed to equal 1 in Figs. 7.12 and 7.13 for APE and AUS smearing respectively. Figs. 7.14 and 7.15 report the final results of a similar analysis for APE and AUS smearing respectively at $\beta = 5.7$. Here the $\alpha = 1.0$ results are omitted from the AUS smearing results for clarity.

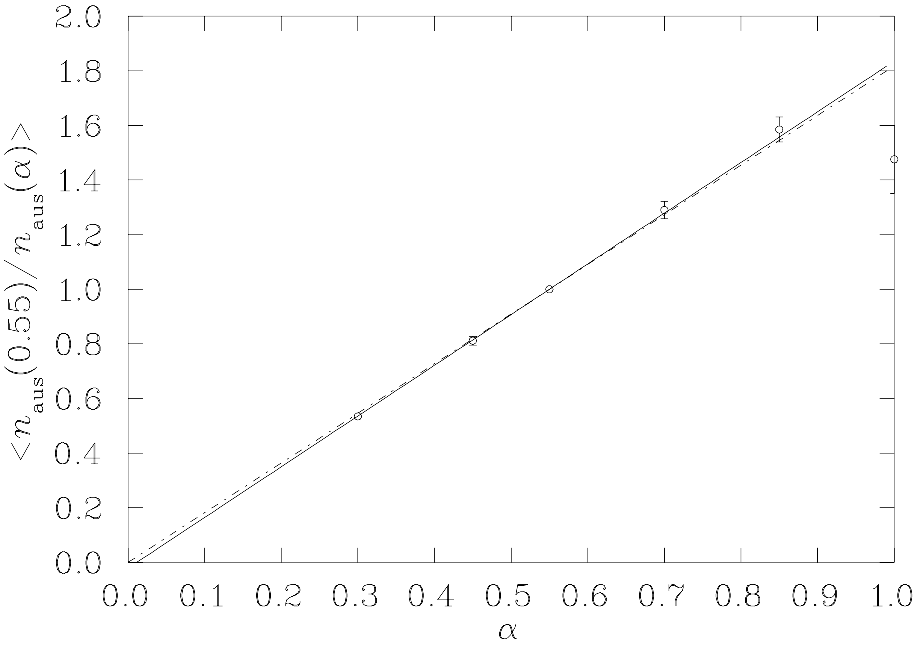


Figure 7.11: Illustration of the α dependence of $c_1 = \langle n(\alpha' = 0.55)/n(\alpha)\rangle$ for AUS smearing. Fits to the data exclude the point at $\alpha = 1$. The solid line fit to the data indicates $c_1 = 1.857\,\alpha - 0.021$ whereas the dashed line, constrained to pass through the origin, provides a slope of 1.818.

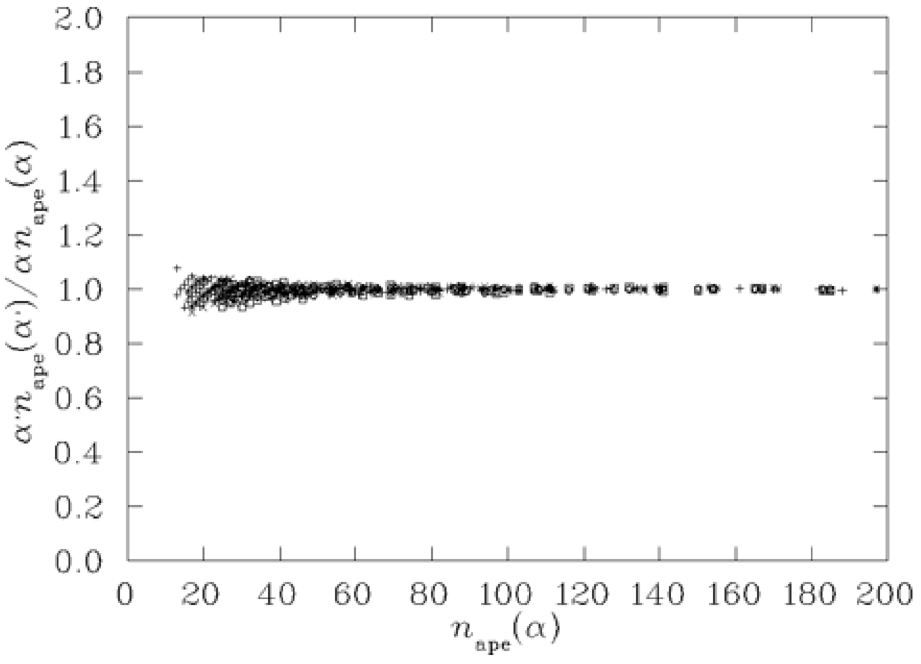


Figure 7.12: Illustration of the degree to which the relations of (7.45) are satisfied for the action under APE smearing. Here $\alpha < \alpha'$ and $\alpha' = 0.30$, 0.45, 0.55, and 0.70. Data are from the $24^3 \times 36$ lattice at $\beta = 6.0$.

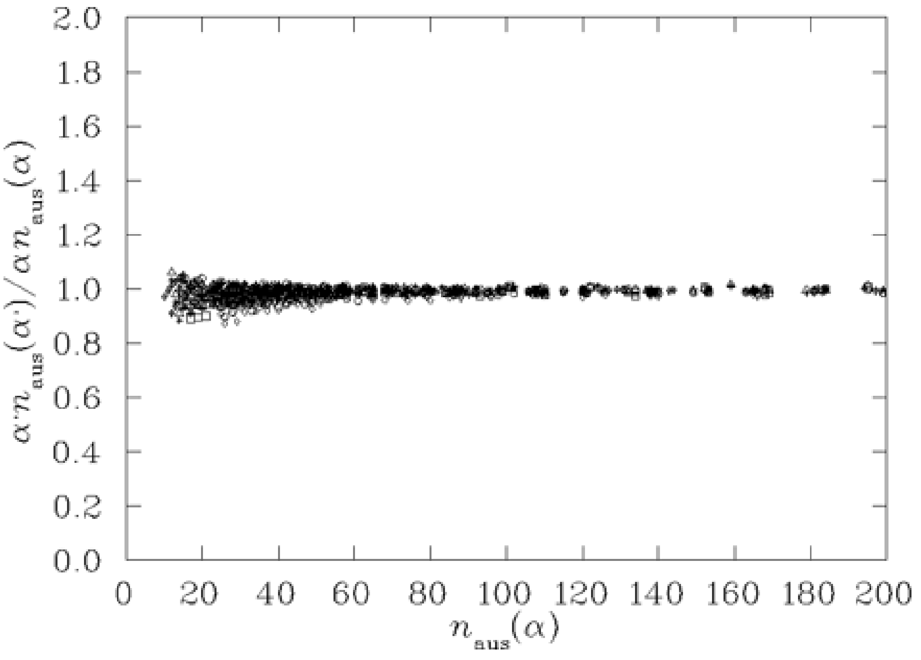


Figure 7.13: Illustration of the degree to which the relations of (7.45) are satisfied for the action under AUS smearing. Here $\alpha < \alpha'$ and $\alpha' = 0.30$, 0.45, 0.55, 0.70 and 0.85. Data are from the $24^3 \times 36$ lattice at $\beta = 6.0$.

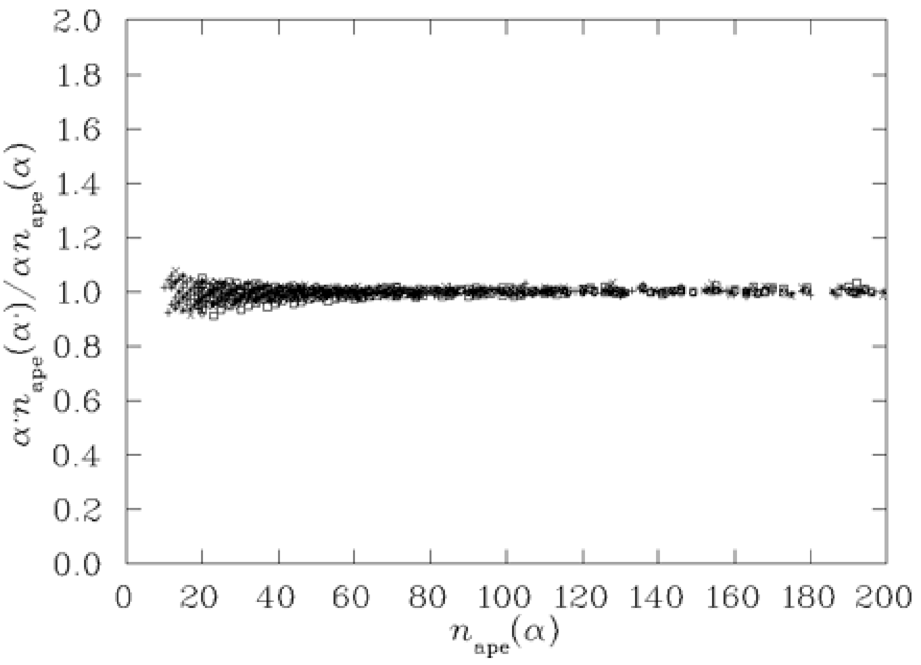


Figure 7.14: Illustration of the degree to which the relations of (7.45) are satisfied for the action under APE smearing. Here $\alpha < \alpha'$ and $\alpha' = 0.30$, 0.45, 0.55, and 0.70. Data are from the $16^3 \times 32$ lattice at $\beta = 5.7$.

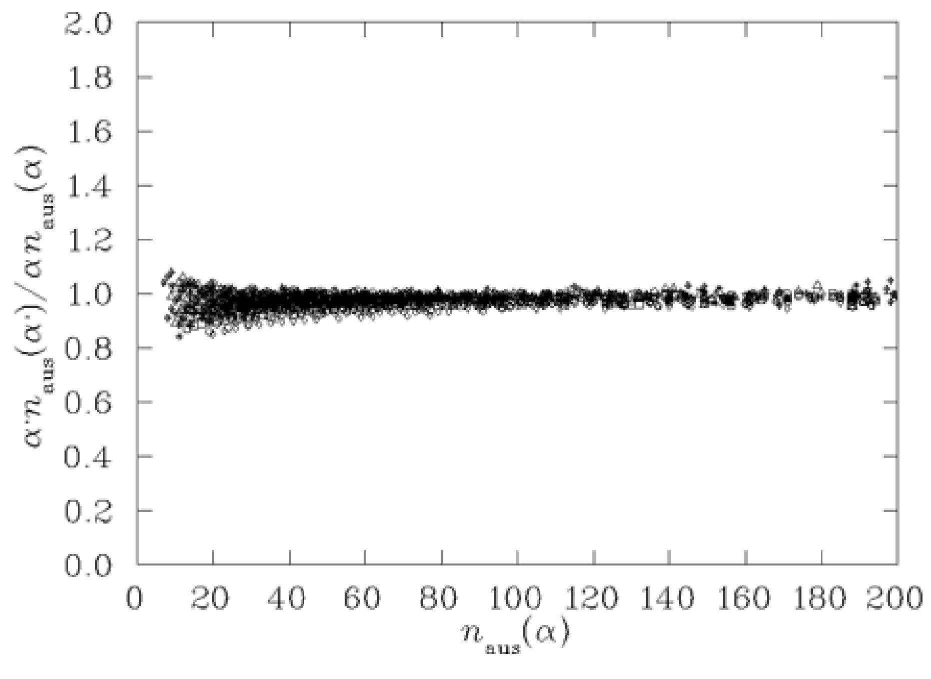


Figure 7.15: Illustration of the degree to which the relations of (7.45) are satisfied for the action under AUS smearing. Here $\alpha < \alpha'$ and $\alpha' = 0.30$, 0.45, 0.55, 0.70 and 0.85. Data are from the $16^3 \times 32$ lattice at $\beta = 5.7$.

Cooling Calibration

Here the previous analysis is repeated, this time comparing cooling with APE and AUS smearing. I consider the same linear ansatz for the relationship and plot $n_{\rm c}/n_{\rm ape}(\alpha)$ versus $n_{\rm ape}(\alpha)$ for APE smearing in Fig. 7.16. Fig. 7.17 reports the ratio $n_{\rm c}/n_{\rm aus}(\alpha)$ versus $n_{\rm aus}(\alpha)$ for AUS smearing. At small numbers of smearing sweeps, large integer discretization errors of the order of 25% are present, as n_c is as small as 4. With this in mind, we see an independence of the ratio on the amount of cooling/smearing over a wide range of smearing sweeps for both APE and AUS smearing. This supports a linear relation between the two algorithms.

Averaging the results provides

$$\frac{n_{\rm c}}{n_{\rm ape}(0.50)} = 0.330 \quad \text{and} \quad \frac{n_{\rm c}}{n_{\rm aus}(0.55)} = 0.340\,, \tag{7.47}$$

or more generally

$$n_{\rm c} \simeq 0.600\,\alpha\,n_{\rm ape}(\alpha) \quad \text{and} \quad n_{\rm c} \simeq 0.618\,\alpha\,n_{\rm aus}(\alpha)\,. \tag{7.48}$$

Hence we see that cooling is much more efficient at smoothing than the smearing algorithms requiring roughly half the number of sweeps for a given product of n and α.

Equating the equations of Eq. (7.48) provides the following relation between APE and AUS smearing

$$\alpha\,n_{\rm ape}(\alpha) \simeq 1.03\,\alpha'\,n_{\rm aus}(\alpha')\,, \tag{7.49}$$

summarizing the near equivalence of the two smearing algorithms. It is important to recall that the annealing of the AUS smearing algorithm removes the instability of APE smearing encountered at large smearing fractions, α. Thus AUS smearing offers

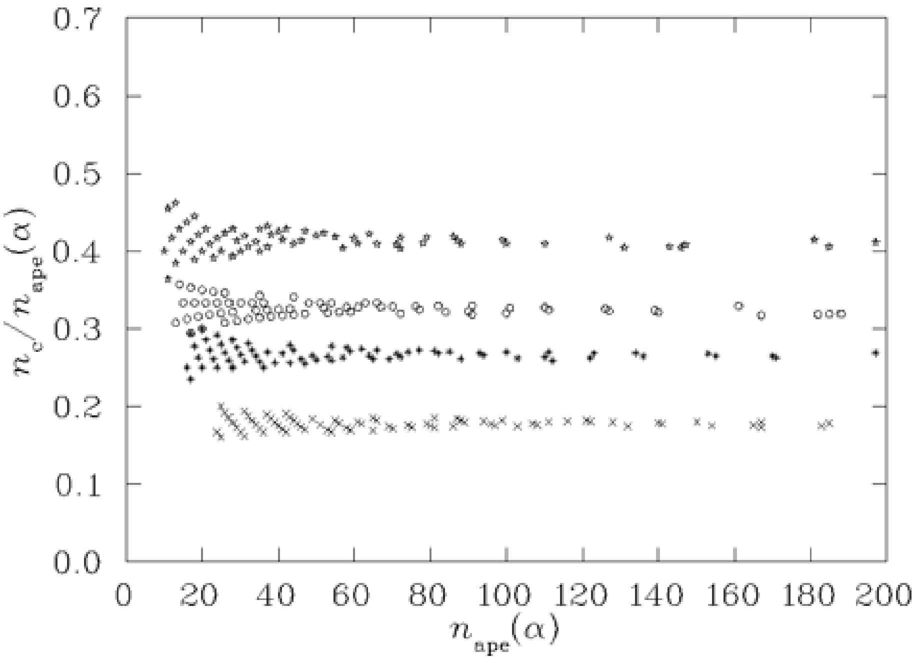


Figure 7.16: Ratio of cooling to APE smearing sweeps as a function of the number of APE smearing sweeps. From top down $\alpha = 0.70$ 0.55, 0.45 and 0.30.

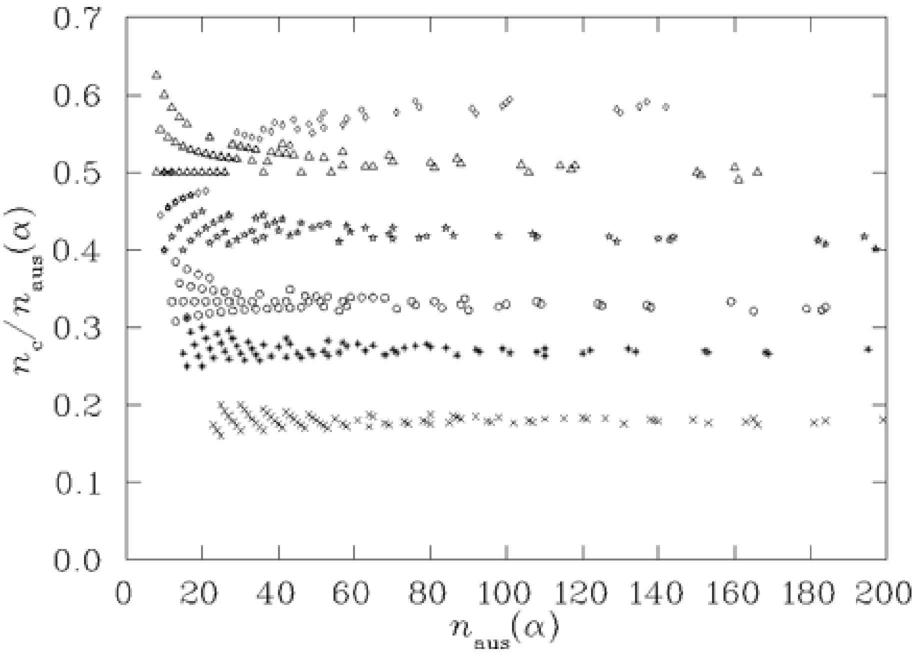


Figure 7.17: Ratio of cooling to AUS smearing sweeps as a function of the number of AUS smearing sweeps. From top down $\alpha = 1.00$, 0.85 0.70 0.55, 0.45 and 0.30.

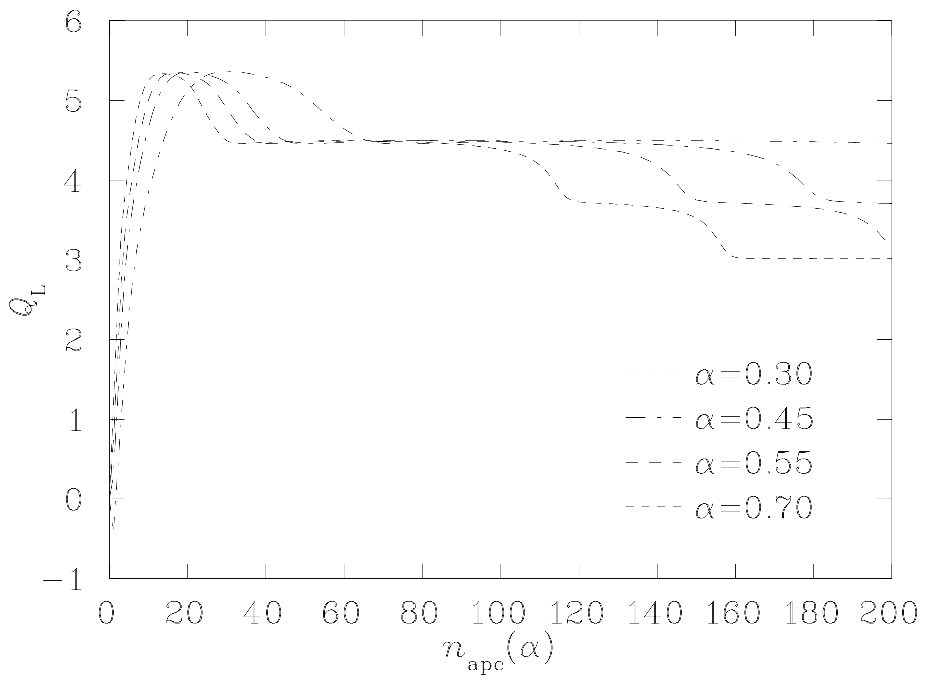


Figure 7.18: Typical evolution curve of the lattice topological charge operator as a function of the number of sweeps for APE smearing. Data are from the $24^3 \times 36$ lattice at $\beta = 6.00$. Each curve corresponds to a particular α value and indicated.

a more stable gauge equivariant smoothing of gauge fields. It offers faster smoothing due to its ability to handle larger smearing fractions. This algorithm may be of use in studying Gribov ambiguities in Landau gauge fixing [47].

A similar analysis of the $16^3 \times 32$ lattice at $\beta = 5.7$ provides

$$n_{\rm c} \simeq 0.572\,\alpha\,n_{\rm ape}(\alpha) \quad \text{and} \quad n_{\rm c} \simeq 0.604\,\alpha\,n_{\rm aus}(\alpha)\,, \tag{7.50}$$

with

$$\alpha\,n_{\rm ape}(\alpha) \simeq 1.06\,\alpha'\,n_{\rm aus}(\alpha')\,. \tag{7.51}$$

Here the change in the coefficient relating APE smearing to cooling appears to be proportional to β.

7.7.2 Topological Charge Density Analysis

Typical evolution curves for the lattice topological charge operator of Eq. (7.4) are shown in Figs. 7.18, 7.19 and 7.20 for APE smearing, AUS smearing and cooling respectively. These data are obtained from a typical gauge configuration on our $24^3 \times 36$ lattice at $\beta = 6.00$. The configuration used here is the same representative configuration illustrated in Figs. 7.6 and 7.7 of the action analysis.

The main feature of these figures is that the smearing/cooling algorithms produce a similar trajectory for the topological charge. For most cases only the rate at which the trajectory evolves changes. In these cases, one can use these trajectories as another way in which to calibrate the rates of the algorithms.

After a few sweeps, the topological charge converges to a near integer value with an error of about 10%, typical of clover definitions of Q at $\beta = 6.0$. The standard Wilson action is known to lose (anti)instantons during cooling due to $\mathcal{O}(a^2)$ errors

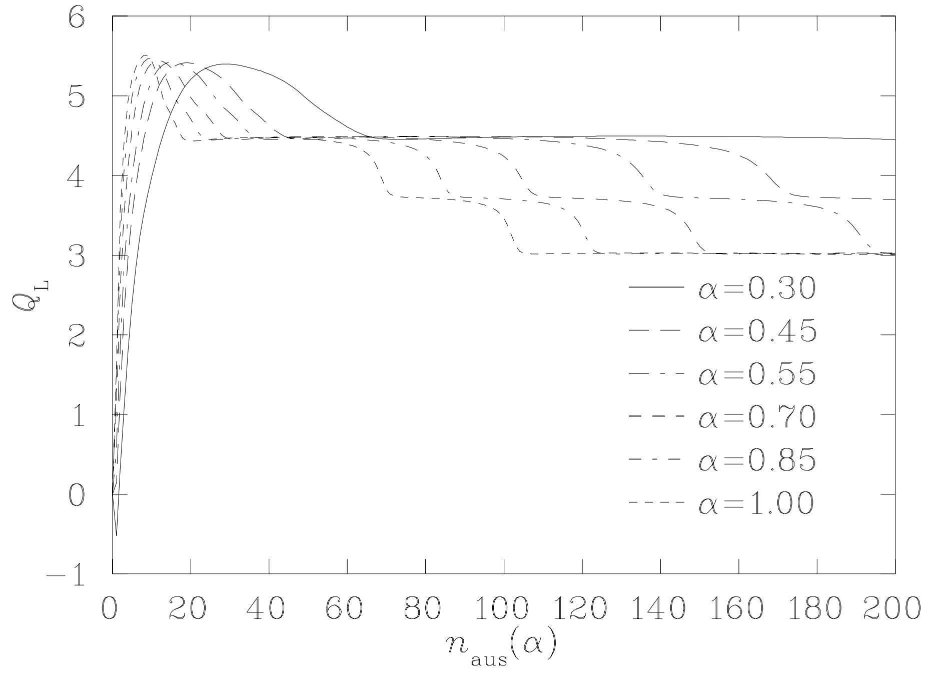


Figure 7.19: Typical evolution curve of the lattice topological charge operator under AUS smearing. Data are from the $24^3 \times 36$ lattice at $\beta = 6.00$. Each curve corresponds to a particular α value and indicated.

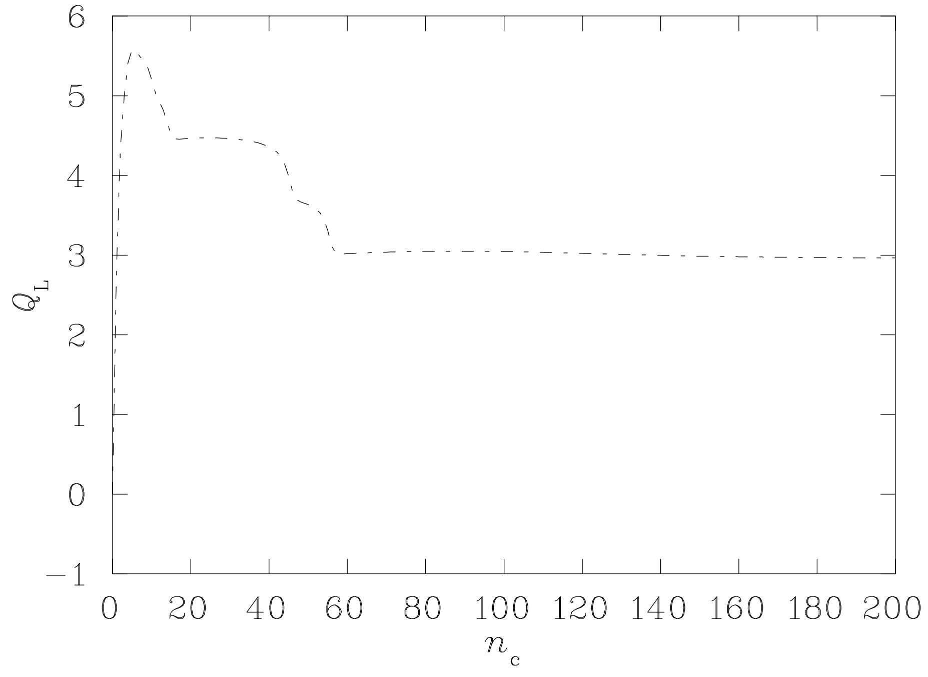


Figure 7.20: Typical evolution curve of the lattice topological charge operator as a function of the number of sweeps for cooling with three diagonal $SU_c(2)$ subgroups on a $24^3 \times 36$ lattice at $\beta = 6.00$.

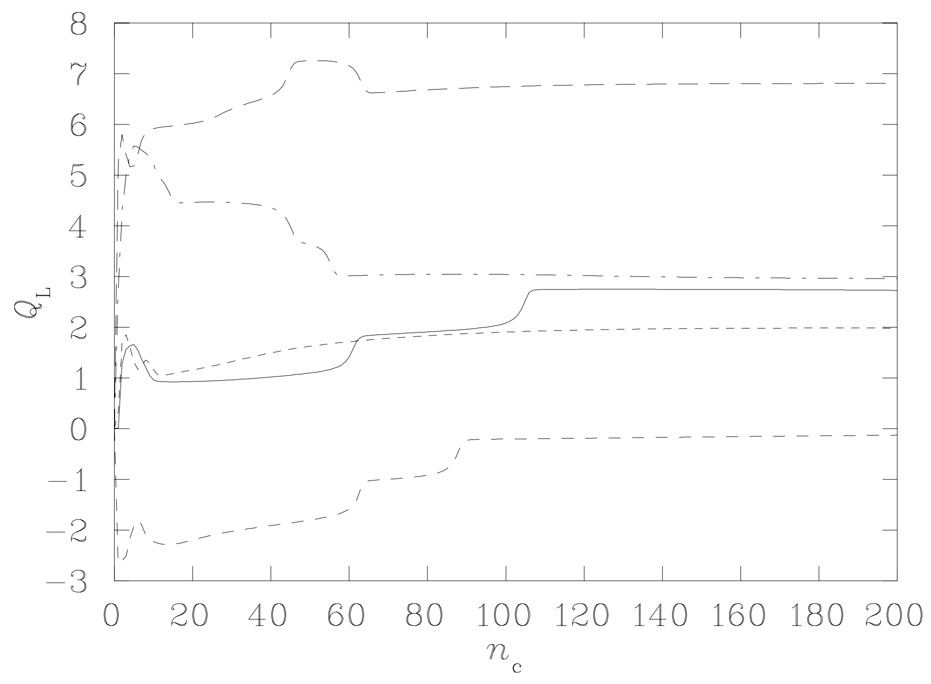


Figure 7.21: The trajectories of the lattice topological charge density operator as a function of the number of sweeps for cooling on typical $24^3 \times 36$, $\beta = 6.00$ configurations.

in the action which act to tunnel through the single instanton action bound. Given the intimate relationship between cooling and smearing discussed in Section 7 it is not surprising to see similar behaviour in the topological charge trajectories of the smearing algorithms considered here. The sharp transitions from one integer to another indicate the loss of an (anti)instanton.

These curves look quite different for other configurations. Fig. 7.21 displays trajectories for cooling on five different gauge configurations. However, the feature of similar trajectories for the various cooling/smearing algorithms remains at $\beta = 6.0$. To calibrate the cooling/smearing algorithms, I select thresholds at topological charge values where the trajectories are making sharp transitions from one near integer to another. Table 7.2 summarizes the thresholds selected and the number of sweeps required to pass though the various thresholds. Note that the second configuration, C_2, data entry in Table 7.2 corresponds to the sampling of the curves illustrated in Figs. 7.18, 7.19 and 7.20.

At $\beta = 5.7$ no analogous trajectories can be found, suggesting that the coarse lattice spacing and larger errors in the action prevent one from reproducing a similar smoothed gauge configuration using different algorithms. In this case the two-dimensional aspect of the smearing parameter space remains for studies of the topological sector. That this might be the case is hinted at in Fig. 7.15 in which the ratio of smearing results for $\beta = 5.7$ lattices is not as closely constrained to one as for the $\beta = 6.0$ results in Figs. 7.13.

As in the action analysis, the ratio of $\alpha = 0.55$ results to other α value results within APE and AUS smearing is reported. Figs. 7.22 and 7.23 summarize the results for APE and AUS smearing respectively. Again we see an enhanced spread of points at small numbers of smearing sweeps due to integer discretization errors.

Data for large numbers of sweeps at small and large α values are absent in Fig. 7.23.

Table 7.2: Summary of the number of sweeps required to pass through various topological charge thresholds. The selection of the thresholds is described in the text. Smearing fractions α are indicated in the table headings. Omissions in the table indicate either the threshold was not met within 200 sweeps or that the trajectory diverged from the most common trajectory among the algorithms.

Config.	Threshold	Cooling	APE smearing				AUS smearing					
			0.30	0.45	0.55	0.70	0.30	0.45	0.55	0.70	0.85	1.00
	1.00 fall	2	3	3	2	-	3	2	2	-	-	-
C_1	1.00 rise	6	7	5	4	3	6	4	3	2	1	1
	1.00 fall	10	41	28	24	20	35	24	19	15	25	17
	1.25 rise	2	9	6	4	3	8	4	3	2	1	1
	1.25 fall	6	25	17	14	12	24	17	13	11	-	-
	5.0 fall	11	52	35	29	23	49	33	26	20	16	13
	4.0	45	-	178	146	114	-	169	137	110	84	69
	3.5	55	-	-	195	152	-	-	189	142	120	101
C_2	5.0 rise	3	18	12	10	8	12	11	9	7	5	4
	4.0	2	11	7	6	4	10	6	5	3.5	3	2.5
	3.5	1	8	5	4	3	8	5	4	3	2	2
	3.0	1	7	4	3.5	3	6	4	3	2.5	2	2
	2.0 rise	1	4	3	2.5	2	4	3	2	1.5	1	1
	1.0 rise	2	11	7	6	4	10	6	5	3	2	2
	1.0	10	94	62	49	36	-	-	26	20	17	16
C_3	1.5 rise	3	25	17	15	11	21	13	11	8	6	6
	1.5 fall	7	42	25	20	15	39	24	19	15	12	9
	1.5	60	-	168	139	110	-	162	132	-	-	190
	6.0	20	-	135	110	85	-	136	112	88	-	-
C_4	5.0	1	6	4	3	2	-	4	3	2	2	1
	4.0	1	4	3	2	1	-	3	3	2	1	1

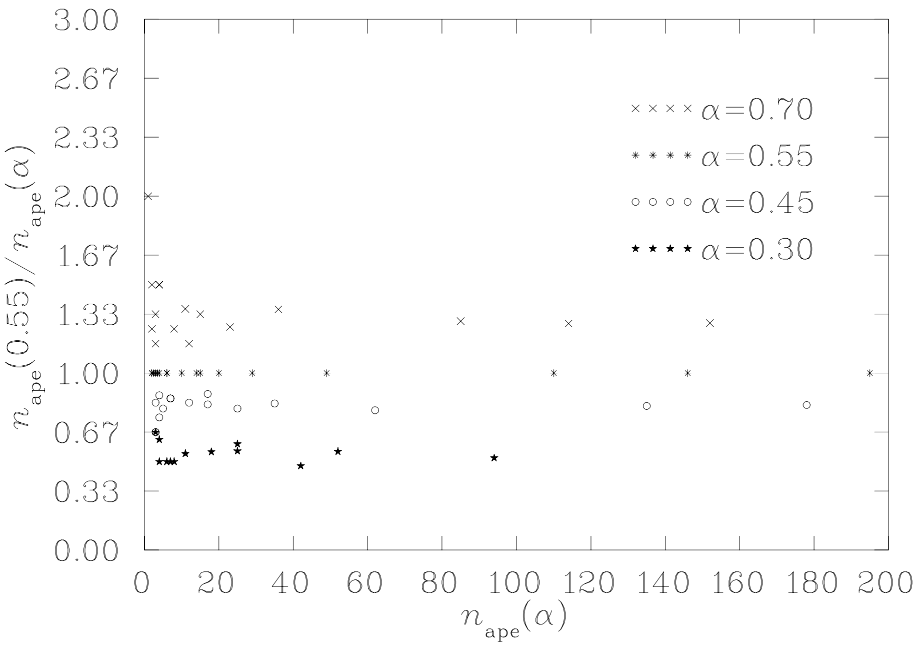


Figure 7.22: The ratio $n_{\text{ape}}(0.50)/n_{\text{ape}}(\alpha)$ versus $n_{\text{ape}}(\alpha)$ for Q_L from the data of Table 7.2 extracted from the $24^3 \times 36$ configurations at $\beta = 6.00$. From the top down, the horizontal sets of points correspond to $\alpha = 0.70$, 0.55, 0.45 and 0.30.

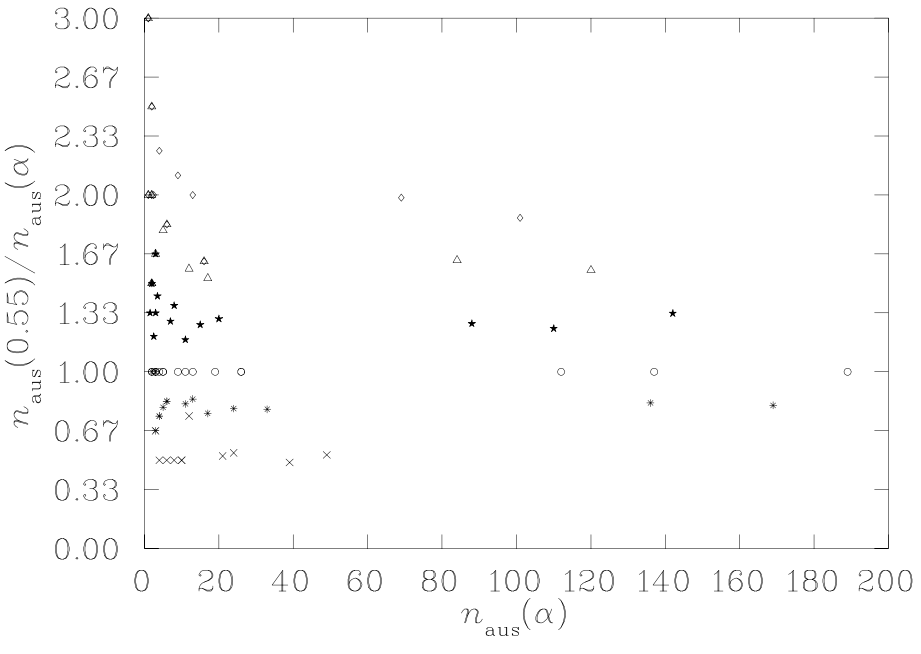


Figure 7.23: The ratio $n_{\text{aus}}(0.55)/n_{\text{aus}}(\alpha)$ versus $n_{\text{aus}}(\alpha)$ for Q_L from the data of Table 7.2 extracted from the $24^3 \times 36$ configurations at $\beta = 6.00$. From the bottom up, the symbols correspond to $\alpha = 0.30$, 0.45, 0.55, 0.70, 0.85 and 1.00.

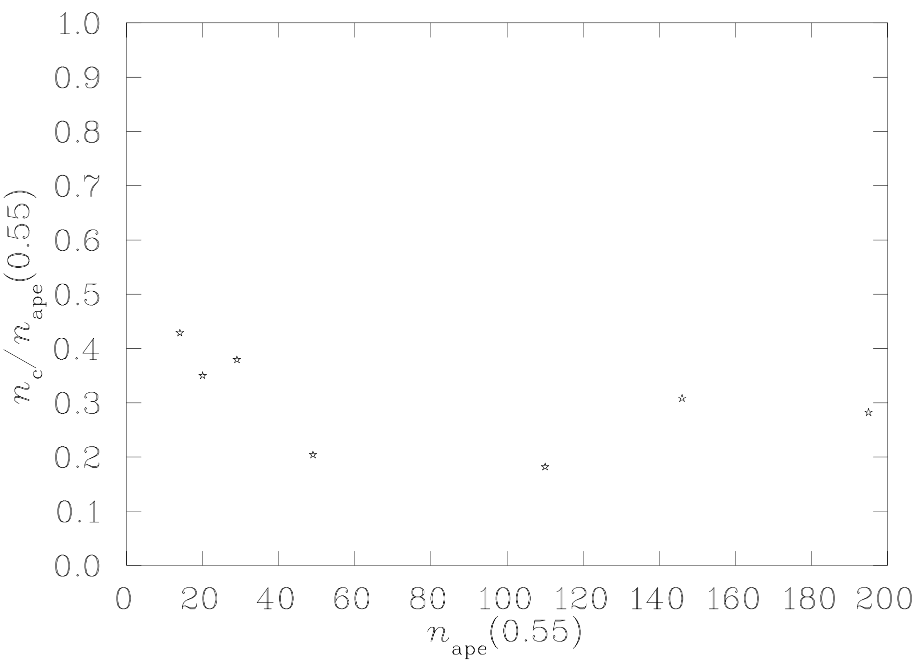


Figure 7.24: Calibration of cooling and APE smearing via the ratio $n_c/n_{ape}(0.50)$ versus $n_{ape}(0.50)$.

The absence of points for small α values is simply due to the thresholds not being crossed within 200 smearing sweeps. The absence of points for large α values reflects the divergence of the topological charge evolution from the most common trajectory among the algorithms. This divergence is also apparent in Fig. 7.9 where the $\alpha = 1.0$ AUS smearing results fail to satisfy the linear ansatz.

While the data from the topological charge evolution is much more sparse, one can see reasonable horizontal bands forming supporting a dominant linear relationship between various α values. Averages of the bands are reported in Table 7.3 along with previous results from the action analysis. With the exception of the $\alpha = 1.0$ AUS smearing results, the agreement is remarkable, leading to the same conclusions of the action analysis summarized in Eq. (7.45).

In calibrating cooling via the topological charge defects, the ratio $n_c/n_{ape}(0.50)$ is reported as a function of $n_{ape}(\alpha)$ and $n_c/n_{aus}(0.55)$ as a function of $n_{aus}(\alpha)$ for APE and AUS smearing respectively. The results are shown in Figs. 7.24 and 7.25.

The data is too poor to determine anything beyond a linear relation between n_c and $n_{ape}(\alpha)$ or $n_{aus}(\alpha)$. Averaging the results provides

$$\frac{n_c}{n_{ape}(0.50)} = 0.30(3) \quad \text{and} \quad \frac{n_c}{n_{aus}(0.55)} = 0.35(3)\,, \tag{7.52}$$

Table 7.3: The average of the ratio $< n_{ape}(0.50)/n_{ape}(\alpha) >$ or $< n_{aus}(0.55)/n_{aus}(\alpha) >$ for the action analysis, (S), and the topological charge analysis, (Q) from the $24^3 \times 36$ lattice.

	APE smearing				AUS smearing					
α	0.30	0.45	0.55	0.70	0.30	0.45	0.55	0.70	0.85	1.00
S	0.541(1)	0.816(1)	1.0	1.278(1)	0.534(1)	0.812(1)	1.0	1.290(1)	1.584(2)	1.475(7)
Q	0.54(1)	0.83(1)	1.0	1.28(2)	0.54(3)	0.812(8)	1.0	1.28(1)	1.65(4)	1.90(6)

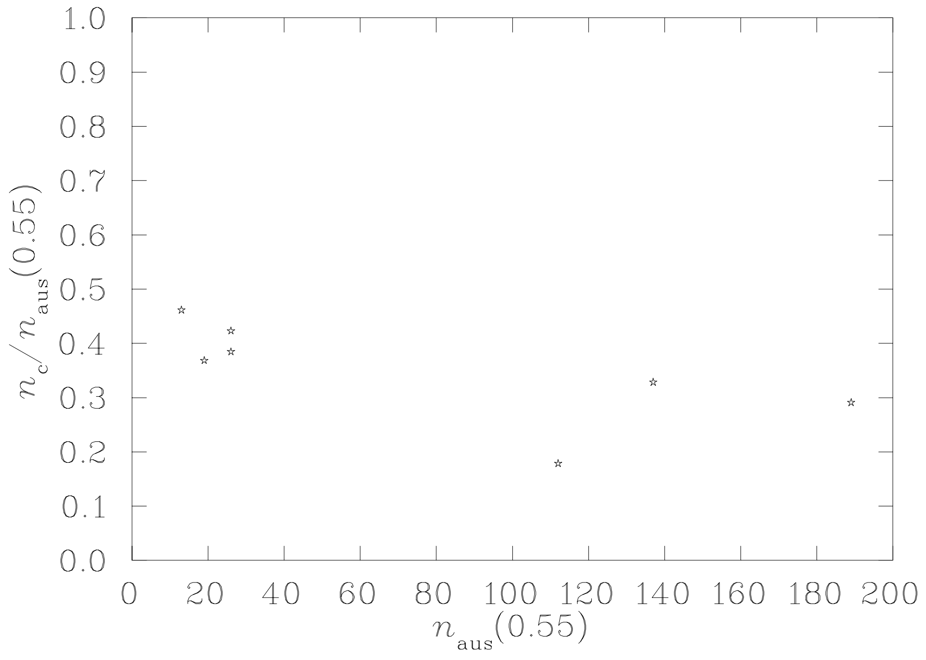


Figure 7.25: Calibration of cooling and AUS smearing via the ratio $n_c/n_{\rm aus}(0.55)$ versus $n_{\rm aus}(0.55)$.

in agreement with the earlier action based results of Eq. (7.47).

As a final examination of the relations I have established among APE smearing, AUS smearing and cooling algorithms, we illustrate the topological charge density is illustrated in Fig. 7.26. Large positive (negative) winding densities are shaded red (blue) and symmetric isosurfaces aid in rendering the shapes of the densities. In fixing the x coordinate to a constant, a three-dimensional slice of a four-dimensional $24^3 \times 36$ gauge field configuration is displayed. Fig. 7.26(a) illustrates the topological charge density after 21 APE smearing steps at $\alpha = 0.7$. The other three quadrants display APE smearing, AUS smearing and cooling designed to reproduce Fig. 7.26(a) according to the relations of Eqs. (7.45), (7.48) and (7.49). The level of detail in the agreement is remarkable.

7.7.3 Summary

The APE smearing algorithm is now widely used in a variety of ways in lattice simulations. It is used to smear the spatial gauge-field links in studies of glueballs, hybrid mesons, the static quark potential, etc. It is used in constructing fat-link actions and in constructing improved operators with smooth transitions to the continuum limit. I have shown that to a good approximation the two-dimensionful parameter space of the number of smearing sweeps $n_{\rm ape}(\alpha)$ and the smearing fraction α may be reduced to a single dimension via the constraint

$$\frac{n_{\rm ape}(\alpha')}{n_{\rm ape}(\alpha)} = \frac{\alpha}{\alpha'} \,. \tag{7.53}$$

satisfied for α and α' in the range 0.3 to 0.7. This result is in agreement with fat-link perturbation theory expectations, and survives for up to 200 sweeps over the lattice. This relation is expected to hold over the entire APE smearing range $0 < \alpha < 3/4$. I find the same relation for AUS smearing provided $\alpha \leq 0.85$. For AUS smearing,

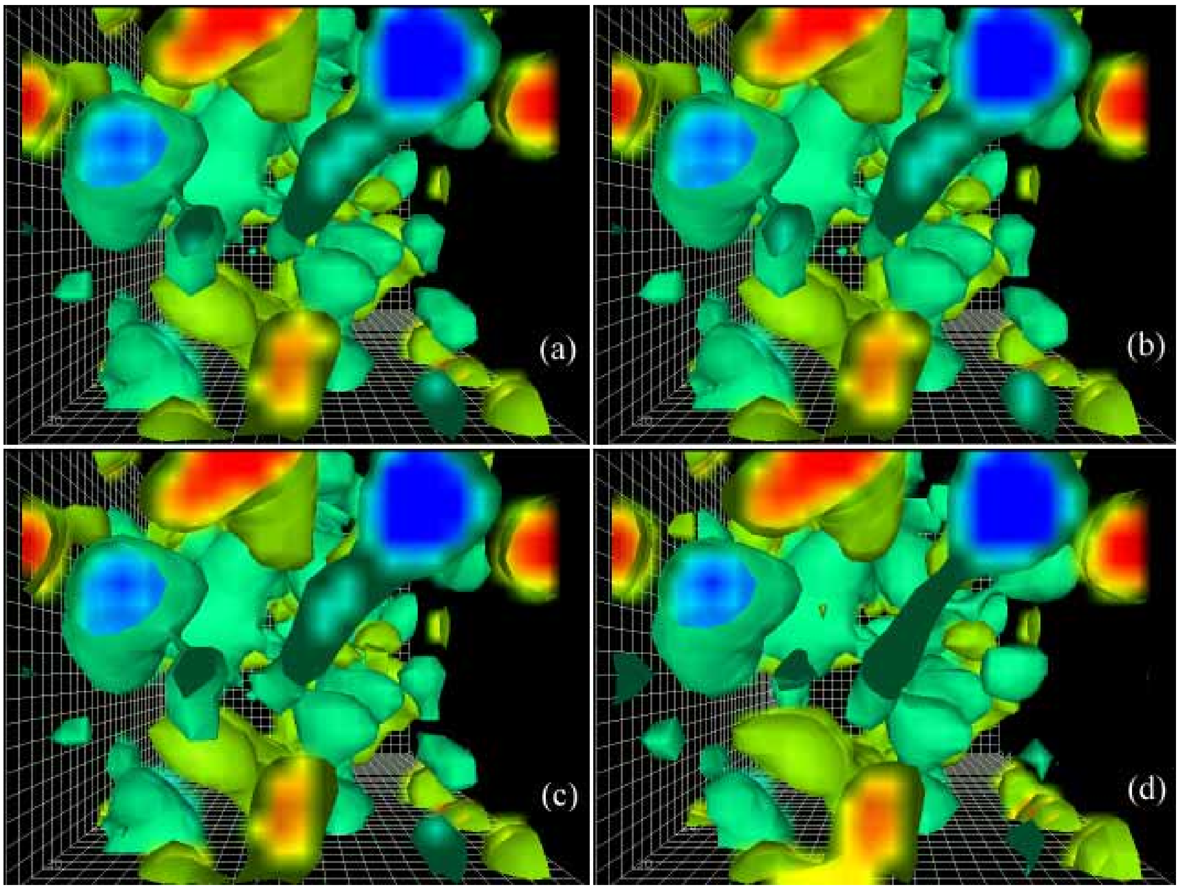


Figure 7.26: Topological charge density of a $24^3 \times 36$ lattice for fixed x coordinate. Positive (negative) windings are coloured red to yellow (blue to green). Fig. (a) illustrates the topological charge density after 21 APE smearing steps at $\alpha = 0.7$. Fig. (b) illustrates the topological charge density after 49 APE smearing steps at $\alpha = 0.3$. Relation (7.45) suggests these configurations should be similar and we see they are to a remarkable level of detail. Fig. (c) illustrates the topological charge density after 20 AUS smearing steps at $\alpha = 0.7$. Relation (7.49) suggests these configurations should be similar and again the detail of the agreement is excellent. Finally (d) illustrates the topological charge density after 9 cooling sweeps, motivated by relation (7.48). While the level of agreement is not as precise, the qualitative features of the smoothed configurations are compatible.

annealing of the links is included as one cycles through the Lorentz directions of the link variables in the smearing process.

The smoothing of gauge field configurations is often a necessary step in extracting observables in which the renormalization constants differ significantly from one, as is the case for the topological charge operator. It is also often used to gain insights into the nonperturbative features of the field theory which give rise to the observed phenomena. We have seen that cooling, APE smearing and AUS smearing produce qualitatively similar smoothed gauge field configurations at $\beta = 6.0$ provided one calibrates the algorithms as follows:

$$n_{\rm c} \simeq 0.600\,\alpha\, n_{\rm ape}(\alpha)\,, \quad n_{\rm c} \simeq 0.618\,\alpha\, n_{\rm aus}(\alpha) \quad \text{and} \quad \alpha\, n_{\rm ape}(\alpha) \simeq 1.03\,\alpha'\, n_{\rm aus}(\alpha')\,. \tag{7.54}$$

The topological charge analysis serves to confirm the action analysis results at $\beta = 6.0$ and further support these relations. However, it also reveals that at $\beta = 5.7$ different trajectories are taken. At $\beta = 5.7$ different algorithms produce smoothed gauge field configurations with similar action, but different topological properties.

As most modern simulations are performed at $\beta \geq 6.0$, the relations of Eqs. (7.53) and (7.54) will be most effective in reducing the exploration of the parameter space. It is now possible to arrive at optimal smearing by fixing the number of smearing sweeps and varying the smearing fraction, or vice versa. Finally, it is now possible to directly compare the physics of smeared and cooled gauge field configurations in a quantitative sense.

I have used the action and topological charge trajectories to calibrate the various smoothing algorithms. While the topological charge may be related to the topological susceptibility, it may be of future interest to introduce other physical quantities such as the string tension as a measure of smoothing [52].

It is well known that interpreting the physics of cooled configurations based on the standard Wilson action is somewhat of an art. The difficulty is that the $\mathcal{O}(a^2)$ errors in the action spoil an instanton under cooling by reducing the action below the one-instanton bound. A consequence of this is a lack of universality. For example, the action varies smoothly to zero while the topological sector jumps as (anti)instantons are destroyed. One may then attempt to define the concept of an optimal amount of smoothing, and perhaps also extrapolate back to zero smoothing steps. Unfortunately, all of these difficulties are apparent in the APE smearing analysis results as well.

Because of the intimate connection between smearing and cooling, as emphasized in the discussion of Section 7.5.2, it is natural to consider improved smearing, where the Hermitian conjugate of the improved staple is used in the smearing process. It will be interesting to see if the benefits of improved cooling carry over into improved smearing and this work is currently in progress.

7.8 Numerical Simulations with Improved Gauge action

In this section I extend the analysis done in Sec. 7.7 using improved gauge field configurations. Two sets of gauge field configurations are analysed. Details may be found in Table 7.1. Analysis of a few configurations proves to be sufficient to resolve the nature of the algorithms under investigation. In this simulation I consider eleven $16^3 \times 32$ configurations and six $24^3 \times 36$ configurations. The general method here is the same as in Sec. 7.7. For each configuration I separately perform 200 sweeps of cooling and 200 sweeps of improved cooling. We explore 200 sweeps of APE smearing at seven values of the smearing fraction and 200 sweeps of improved smearing at five values of the smearing fraction. For APE smearing we consider $\alpha = 0.10$, 0.20, 0.30, 0.40, 0.50, 0.60, and 0.70. Similarly for improved smearing we consider $\alpha = 0.10$ to 0.50 at intervals of 0.10. The extended nature of the staple alters the stability range of α to lie below $\alpha = 0.6$.

For clarity, the number of times an algorithm is applied to the entire lattice is defined as n_{c}, n_{Ic}, $n_{\mathrm{ape}}(\alpha)$ and $n_{\mathrm{Iape}}(\alpha)$ for cooling, improved cooling, APE smearing and improved smearing respectively. Both the total action normalized to the single instanton action $S_0 = 8\pi^2/g^2$ and the topological charge operators, Q_{L} and $Q_{\mathrm{L}}^{\mathrm{Imp}}$ are monitored, from which observe their evolution as a function of the appropriate sweep variable and smearing fraction α is observed.

7.8.1 The Influence Of The Number Of Subgroups On The Gauge Group

I now describe the influence of including additional $SU_c(2)$ subgroups in constructing the gauge group $SU_c(3)$ and explore the impact it has on the smoothing procedure.

The Cabibbo-Marinari algorithm [21] constructs the $SU_c(N)$ gauge groups using $SU_c(2)$ subgroups. It is understood that the minimal set required to construct $SU_c(3)$ matrices is two diagonal $SU_c(2)$ subgroups.

After having performed cooling on gauge field configurations I noticed that the resulting cooled gauge field configurations were not smooth even after a large number of smoothing steps. Adding a third $SU_c(2)$ subgroup made a significant difference in the resulting smoothness.

Further exploration by simply performing additional cycles, n_{cycle}, around the three diagonal $SU_c(2)$ subgroups. I monitored the smoothing rate using both the Standard Wilson and the improved action according to the cycle number being set to $n_{\mathrm{cycle}} = 1, 2$ and 3. Based on the evolution of the action, it is found that the optimum cooling rate on gauge field configurations is achieved using the three diagonal $SU_c(2)$ subgroups cycled over twice, $n_{\mathrm{cycle}} = 2$. Cycling more than twice provides very little further reduction of the action and round off errors may actually increase the action on occasion. Hence two cycles over the three diagonal $SU_c(2)$ subgroups is sufficient to precisely create the $SU_c(3)$ link which minimizes the local action. This determination is crucial to ensuring the effects of our improved action are fully reflected in the $SU_c(3)$ link.

I also monitored the evolution of the topological charge with respect to the above number of cycles. On the $16^3 \times 32$ lattice with spacing of $a = 0.165(2)$ fm, we observe a disagreement of the trajectories for the topological charge for different numbers of the $SU_c(2)$ subgroups cycles. Fig. 7.27 displays results for standard cooling and Fig. 7.28 displays similar results for improved cooling. A comparison of Figs. 7.27 and 7.28 indicates the trajectories also differ between cooling and improved cooling.

Hence we see subtle differences in the algorithms leading to dramatic differences in the topological charge. One must conclude that a lattice spacing of 0.165(2) fm is too coarse for a serious study of topology in $SU_c(3)$ gauge fields. The characteristic size of the topological fluctuations is at the scale of the lattice spacing. As such the gauge fields are simply too rough to smooth in a deterministic manner. On the other hand, a lattice spacing the order of 0.077(1) fm appears to allow a meaningful study

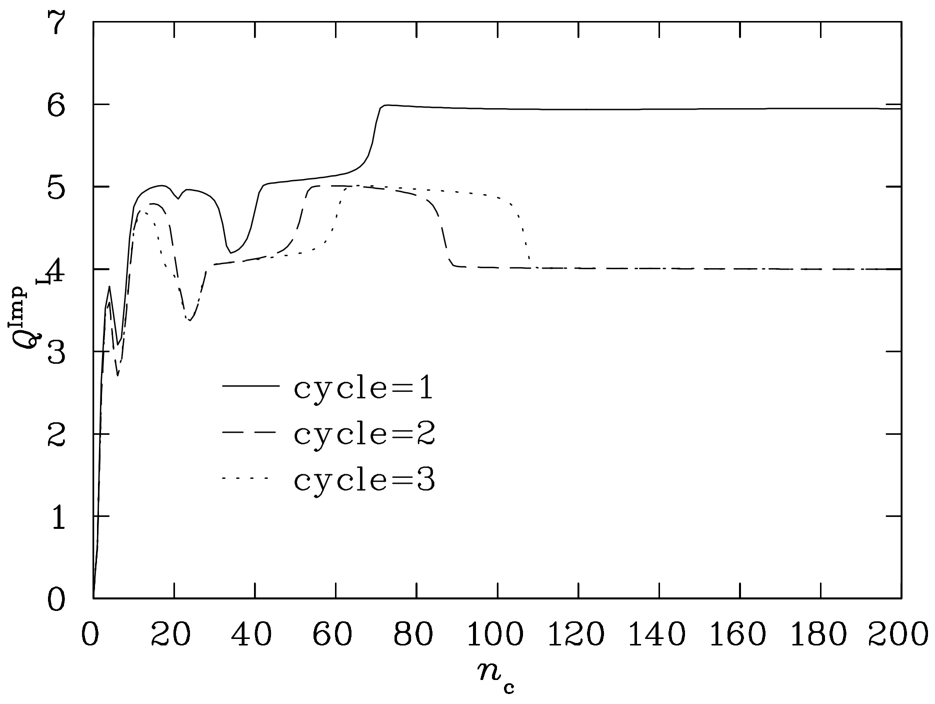


Figure 7.27: The evolution of the topological charge estimated by the improved operator as a function of standard cooling sweeps n_{c} for various numbers of $SU_c(2)$ subgroups. The curves are for a typical configuration from the $16^3 \times 32$ lattices where $a = 0.165(2)$ fm. The parameter cycle describes the number of times the three diagonal $SU_c(2)$ subgroups are cycled over.

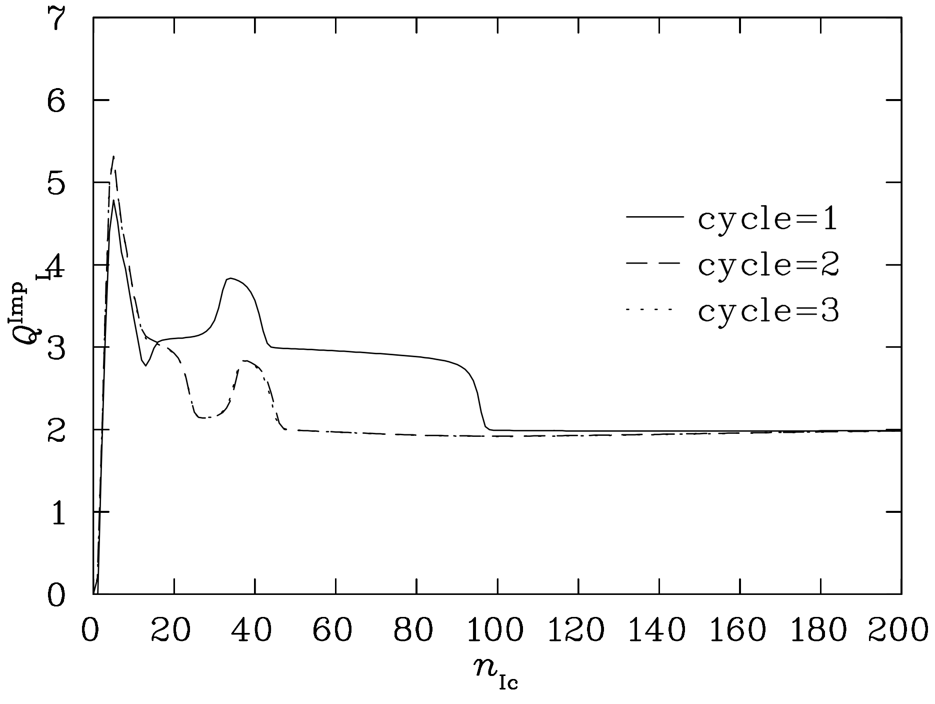


Figure 7.28: The evolution of the topological charge estimated by the improved operator as a function of improved cooling sweeps n_{Ic} for various numbers of cycles over the three diagonal $SU_c(2)$ subgroups. The curves are for the same configuration illustrated in Fig. 7.27.

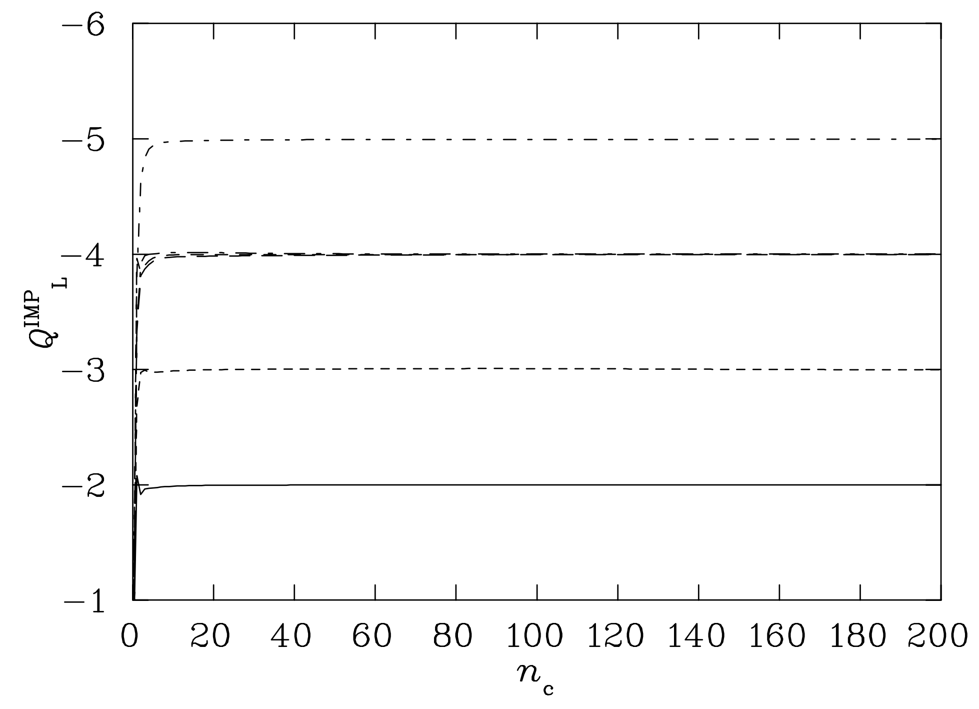


Figure 7.29: The evolution curve for the topological charge estimated via the improved operator as a function of cooling sweeps $n_{\rm c}$ for six configurations on the $24^3 \times 36$ lattices at $\beta = 5.00$ where $a = 0.077(1)$ fm.

of topology in $SU_c(3)$ gauge theory.

We also note here the accuracy with which the improved topological charge operator, Eq. (7.7), reproduces integer values. These results should be contrasted with the usual 10% errors of the unimproved operator at similar lattice spacings. Such errors on a topological charge of 5 can lead to the uncomfortable result of $Q \simeq 4.5$ when the unimproved operator is used.

With the finer $24^3 \times 36$ lattice at $\beta = 5.00$, a perfect agreement among trajectories can be observed for different numbers of cycles of the three $SU_c(2)$ subgroups. Moreover, the topological charge remains stable for hundreds of sweeps following the first three sweeps. Figures 7.29 and 7.30 compare the topological charge evolution for cooling versus improved cooling for six configurations. In every case, the two algorithms produce the same topological charge for a given configuration.

7.8.2 The Action

We begin by considering the action evolution on both lattices. Here I report the action divided by the single instanton action $S_0 = 8\pi^2/g^2$. It is important to note that although the $24^3 \times 36$ lattice has almost four times more lattice sites than the $16^3 \times 32$ lattice, the physical volume is smaller by almost a factor of three. As such the typical topological charges encountered are smaller in magnitude.

Figs. 7.31, 7.32, 7.33, and 7.34 report the typical evolution of the action under standard cooling, improved cooling, APE smearing and improved smearing respectively. Inspection of the figures reveals that improved cooling preserves the action better than standard cooling over a couple hundred sweeps. As expected, standard APE smearing remains slower than cooling or improved cooling even at our most efficient smearing

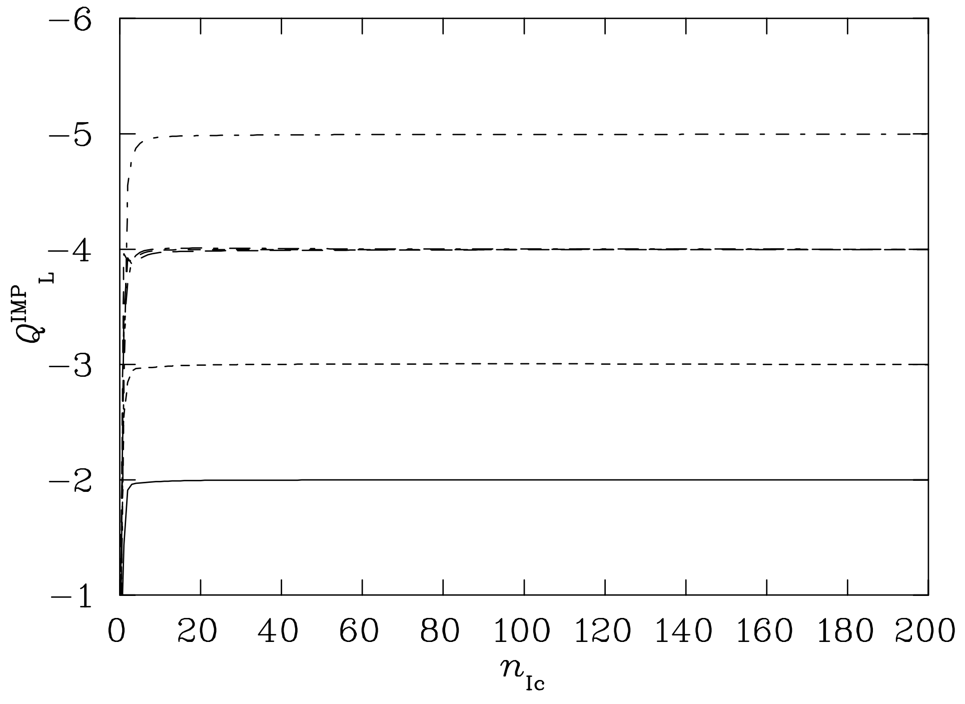


Figure 7.30: The evolution curve for the topological charge estimated via the improved operator as a function of improved cooling sweeps n_{Ic} for the same six configurations from the $24^3 \times 36$ lattices at $\beta = 5.00$ illustrated in Fig. 7.29.

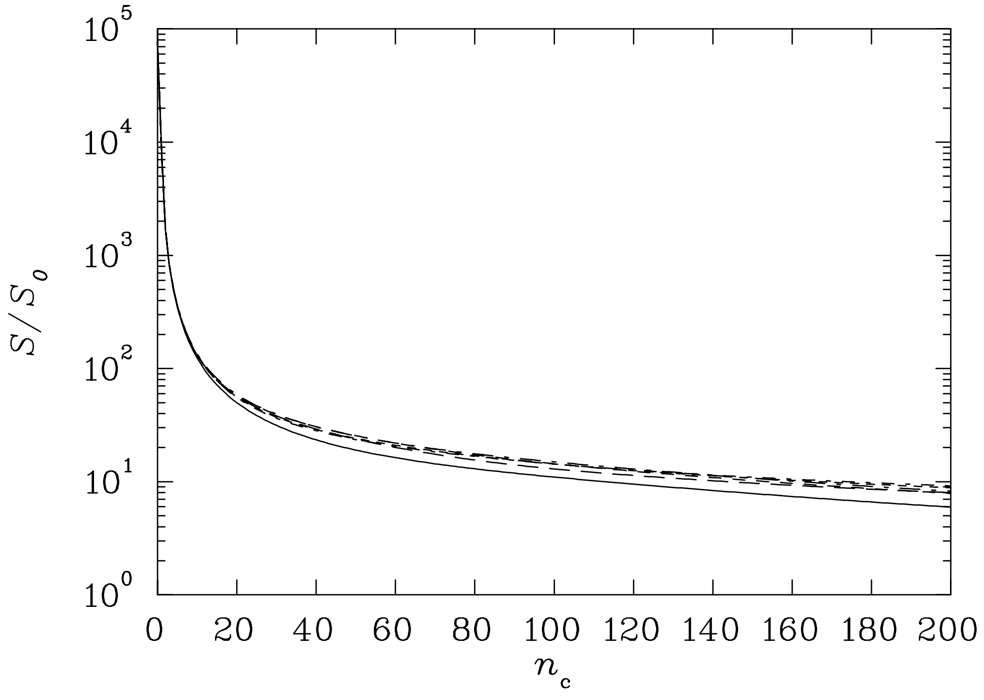


Figure 7.31: The ratio S/S_0 as a function of standard cooling sweeps n_{c} for five configurations on the $24^3 \times 36$ lattice at $\beta = 5.0$. The single instanton action is $S_0 = 8\pi^2/g^2$.

fraction ($\alpha = 0.70$). Similar results are observed for improved smearing at our most efficient smearing fraction of $\alpha = 0.50$ in Fig. 7.34.

Based on these observations, one concludes that the fastest way to remove the short

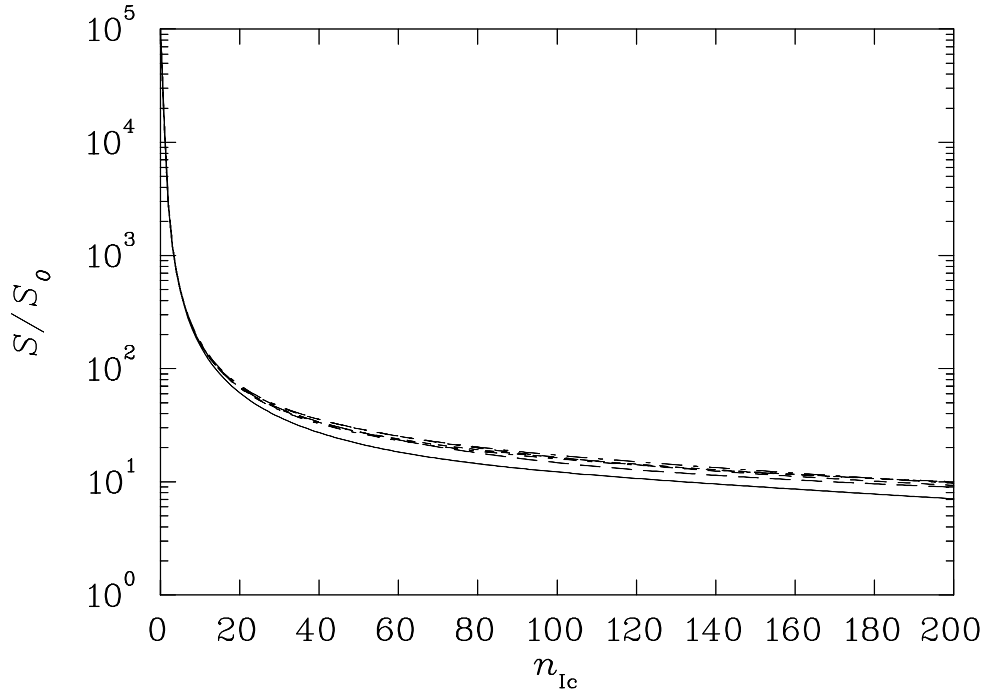


Figure 7.32: The ratio S/S_0 as a function of improved cooling sweeps $n_{\rm Ic}$ for five configurations on the $24^3 \times 36$ lattice at $\beta = 5.0$. The rate of cooling is seen to be somewhat slower than that for the standard cooling.

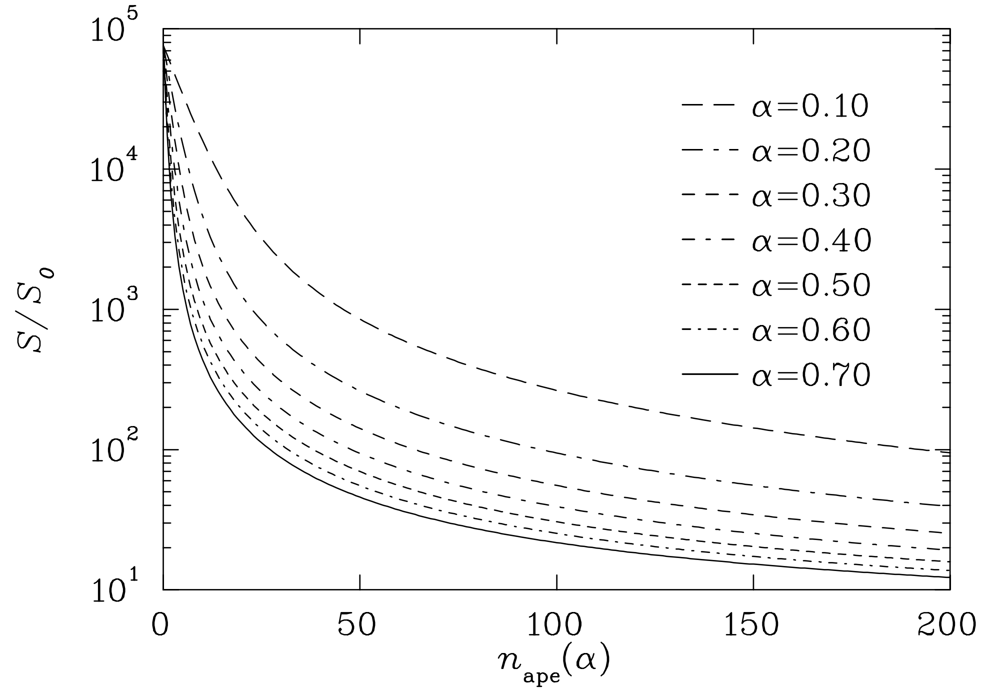


Figure 7.33: The ratio S/S_0 as a function of APE smearing sweeps $n_{\rm ape}(\alpha)$ for one configuration on the $24^3 \times 36$ lattice at $\beta = 5.0$. Each curve has an associated smearing fraction α. The rate of lowering the action for the maximum stable smearing fraction (≈ 0.75) is seen to be less than that for the other standard or improved cooling.

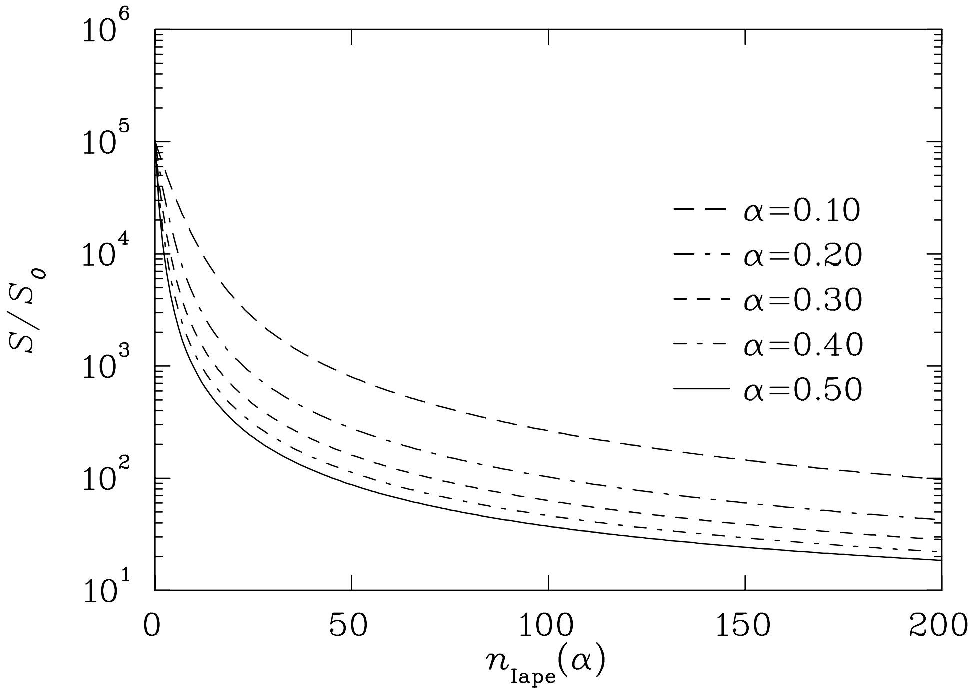


Figure 7.34: The ratio S/S_0 as a function of improved smearing sweeps $n_{\text{Iape}}(\alpha)$ for one configuration on the $24^3 \times 36$ lattice at $\beta = 5.0$. Each curve has an associated smearing fraction α. We see that this is the slowest of the four algorithms for lowering the action as a function of the sweep number.

range quantum fluctuations on an $\mathcal{O}(a^2)$ gauge field configuration, is through standard cooling, which lowers the action more rapidly than improved cooling as a function of cooling sweep. In turn we see that improved cooling is faster than the maximum stable standard APE smearing, which is faster than the maximum stable improved smearing. This is illustrated by Figs. 7.31–7.34. It is important to emphasize that the fastest way of removing these fluctuations is not necessarily the best as far as the topology is concerned. It is already established that the $\mathcal{O}(a^2)$ errors of the standard Wilson action act to underestimate the action. These errors spoil instantons which might otherwise survive under improved cooling.

7.8.3 Topological Charge from Cooling and Smearing

We begin by considering the $16^3 \times 32$ lattices having a lattice spacing of $a = 0.165(2)$ fm. In Fig. 7.35, we plot the evolution curve for the improved topological charge as a function of the cooling sweep number, n_c, for six of our configurations. Similarly in Fig. 7.36, for the same six configurations I plot the improved topological charge operator but this time as a function of improved cooling sweep n_{Ic}. The line types in Figs. 7.35 and 7.36 correspond to the same underlying configurations and are to be directly compared. For example, the solid curve corresponds to the same gauge field configuration in both figures but with a different algorithm applied to it. From these two figures we notice the two cooling methods lead to completely different values for the topological charge.

Improved cooling brings stability to the evolution of the topological charge whereas standard cooling gives rise to numerous fluctuations to the topological charge. For

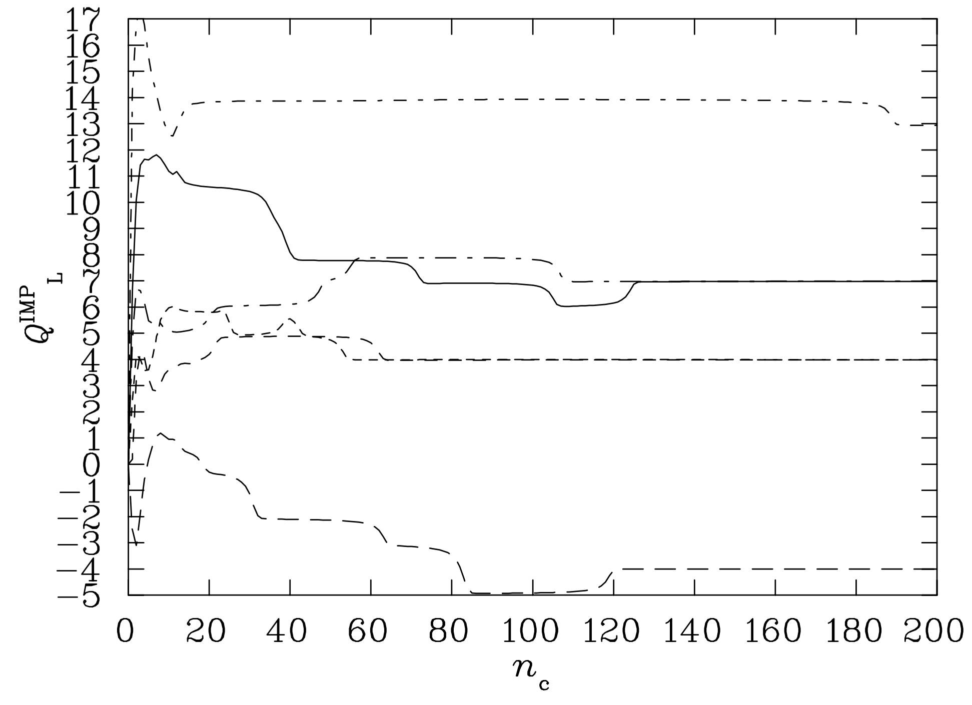


Figure 7.35: $Q_{\mathrm{L}}^{\mathrm{Imp}}$ versus n_{c} for six configurations on the $16^3 \times 32$ lattices at $\beta = 4.38$, $a = 0.165(2)$ fm. Each line corresponds to a different configuration.

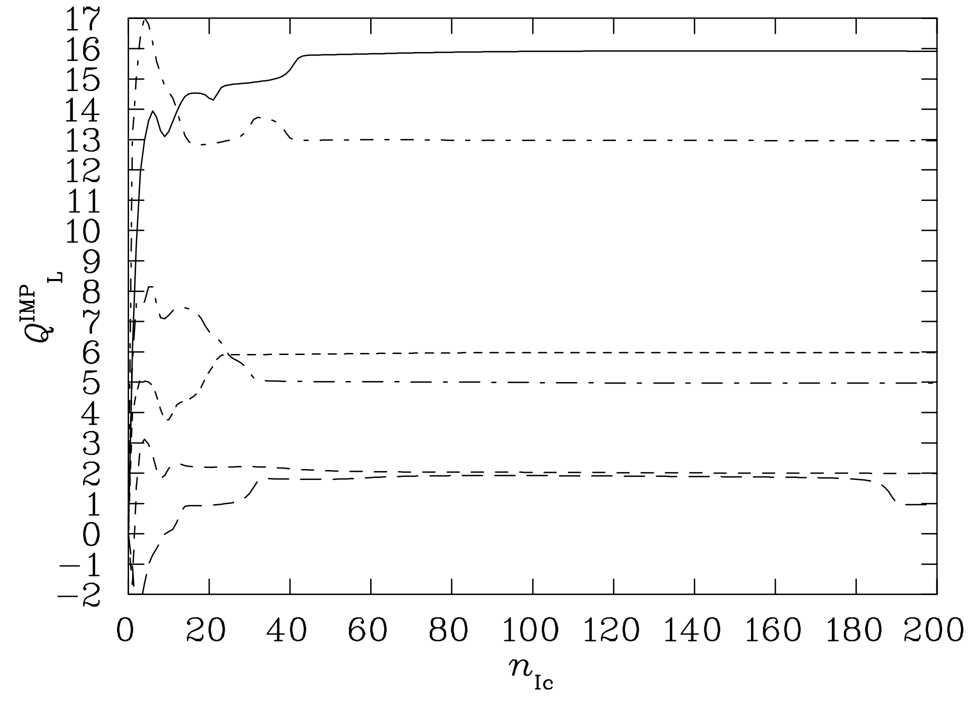


Figure 7.36: $Q_{\mathrm{L}}^{\mathrm{Imp}}$ versus n_{Ic} for six configurations on the $16^3 \times 32$ lattices. The different line types identifying different configurations match the configurations identified in Fig. 7.35.

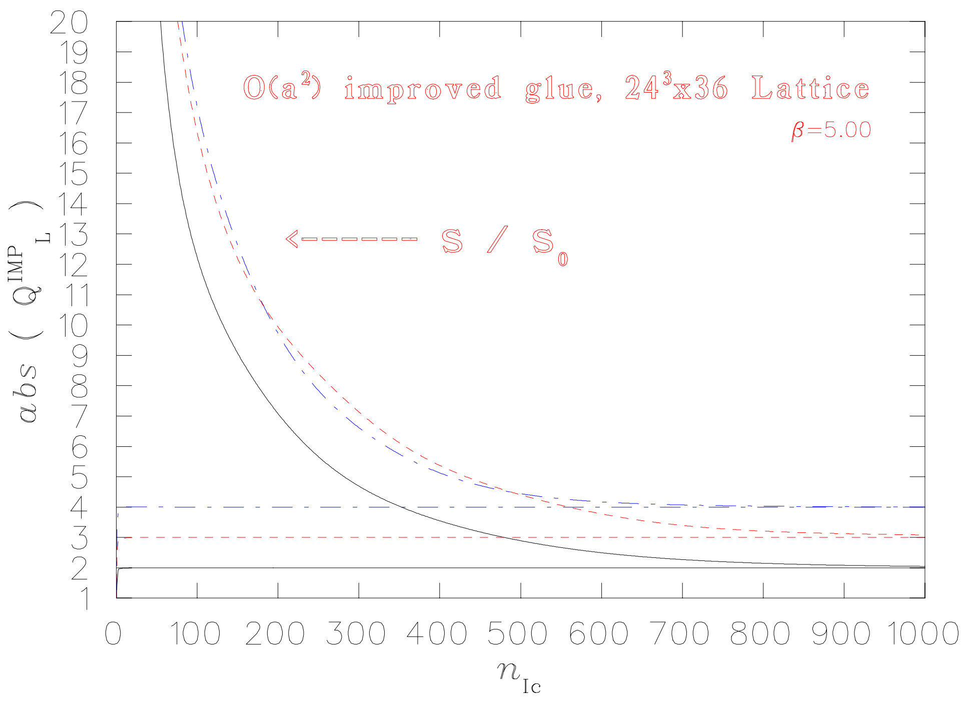


Figure 7.37: $Q_{\mathrm{L}}^{\mathrm{Imp}}$ versus n_{Ic} calculated over a thousand sweeps. Also shown is the normalized to a single instanton action, S/S_0. The action and topological curve are converging to each other for each configuration, illustrating Eq. (7.55).

improved cooling, plateaus appear after about forty sweeps and persist for hundreds of sweeps until the absolute value of $Q_{\mathrm{L}}^{\mathrm{Imp}}$ converges to the normalized to a single instanton action, S/S_0, so we have the following result:

$$\left| Q_{\mathrm{L}} \quad \text{or} \quad Q_{\mathrm{L}}^{\mathrm{Imp}} \right| \longrightarrow S/S_0, \qquad \text{as} \qquad n_{\mathrm{sweeps}} \longrightarrow \infty. \tag{7.55}$$

An illustration of this results may be seen in Fig. 7.37. This is a celebrated feature of improved cooling.

This algorithmic sensitivity of the topological charge is also seen in Figs. 7.38 and 7.39, for APE and improved smearing on a single configuration (the solid line of Figs. 7.35 and 7.36). Within APE smearing or improved smearing, the topological charge trajectories follow similar patterns but different rates for various smearing fractions. However, APE smearing leads to values for the topological charge which are different from that obtained under improved smearing. An important point is that improved smearing stabilizes the topological charge at 95 sweeps for $\alpha = 0.5$, whereas standard smearing shows no sign of stability until 140 iterations at $\alpha = 0.5$. Hence we see significant improvement in the topological aspects of the gauge field configurations under improved smearing.

On the finer lattice, we find a completely different behaviour for the topological charge evolution. The topological charge is established very quickly; after a few sweeps in the case of cooling or improved cooling as illustrated in Figs. 7.29 and 7.30. The topological charge persists without fluctuation for hundreds of sweeps, both for cooling and for smearing as illustrated in Figs. 7.40 and 7.41 for APE and improved smearing respectively. Moreover, the topological charge is independent of the smearing algorithm.

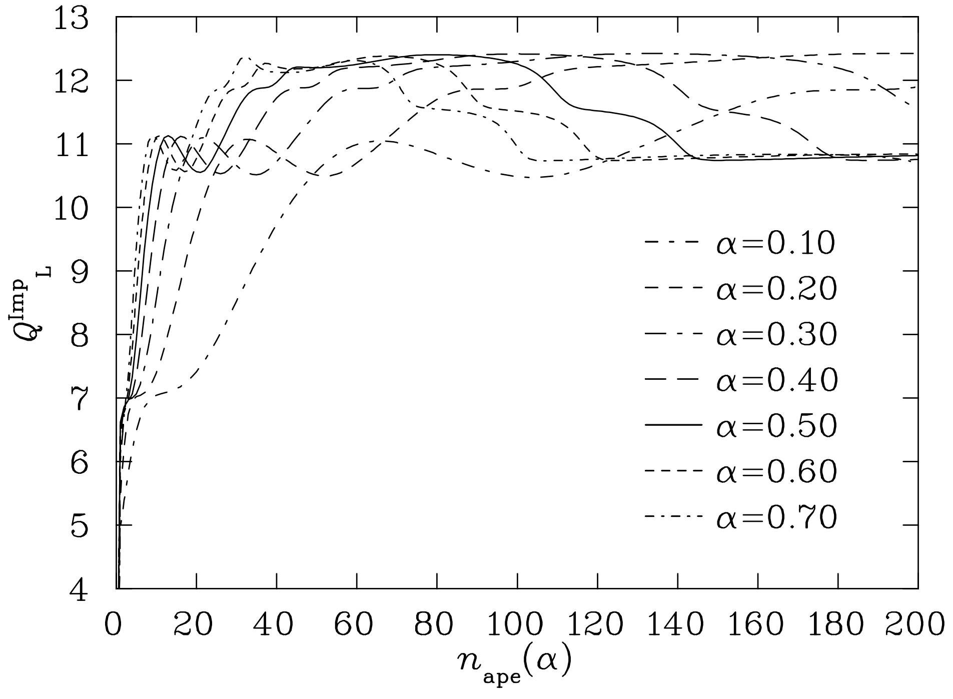


Figure 7.38: The evolution of $Q_{\mathrm{L}}^{\mathrm{Imp}}$ using APE smearing as a function of APE smearing sweep $n_{\mathrm{ape}}(\alpha)$ on the $16^3 \times 32$ lattice at $\beta = 4.38$. Here different line types correspond to different smearing fractions.

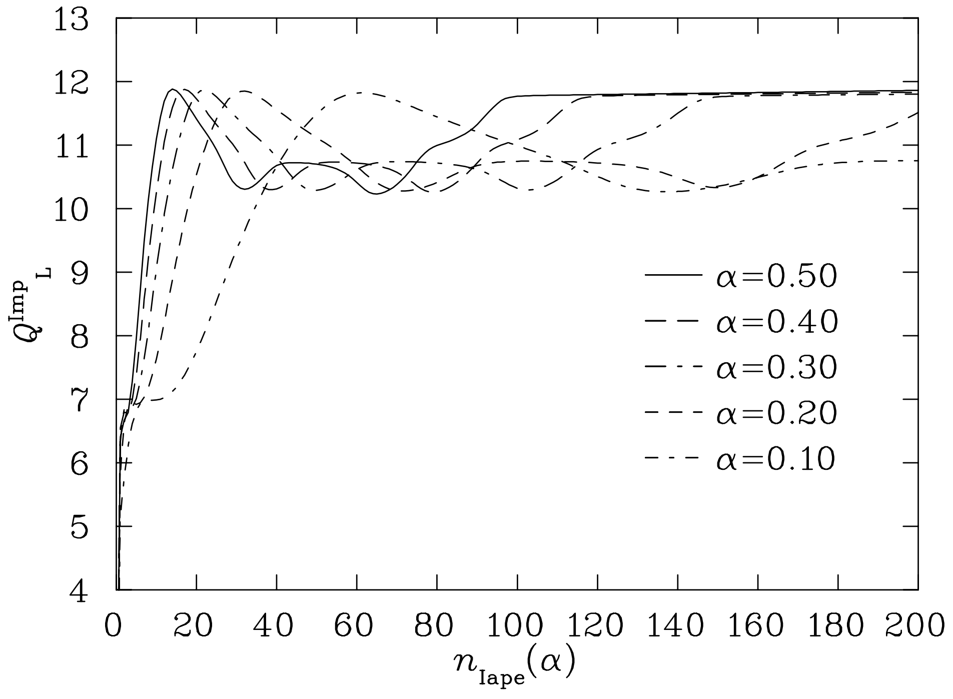


Figure 7.39: The evolution of $Q_{\mathrm{L}}^{\mathrm{Imp}}$ using improved smearing as a function of APE smearing sweep $n_{\mathrm{Iape}}(\alpha)$ on the $16^3 \times 32$ lattice at $\beta = 4.38$. Here different line types correspond to different smearing fractions.

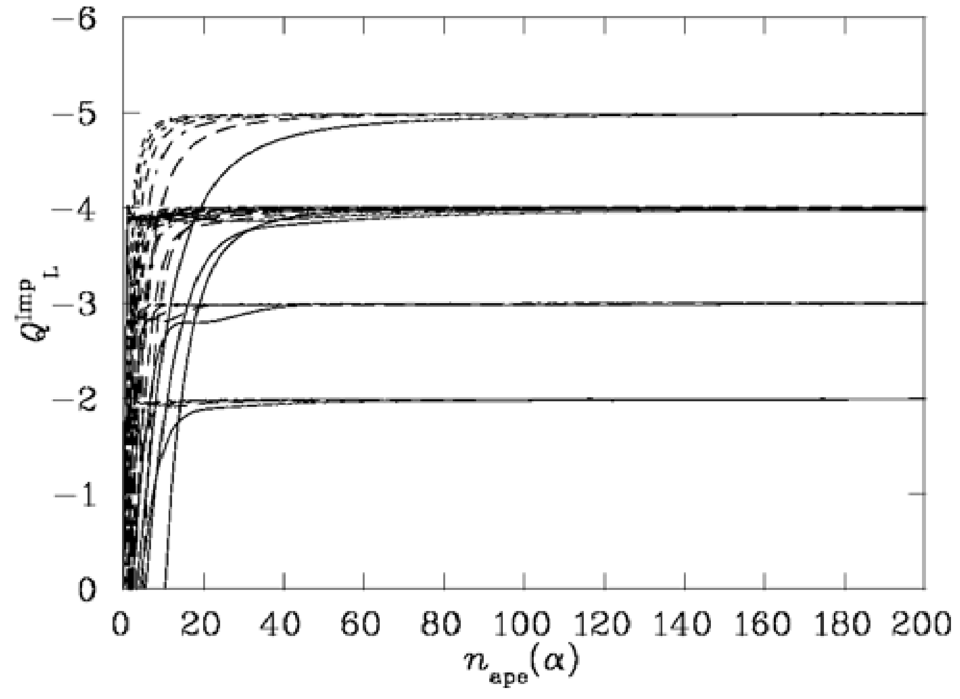


Figure 7.40: The evolution of $Q_{\text{L}}^{\text{Imp}}$ using APE smearing as a function of APE smearing sweep $n_{\text{ape}}(\alpha)$ on the $24^3 \times 36$ lattice at $\beta = 5.00$. Here different line types correspond to different smearing fractions.

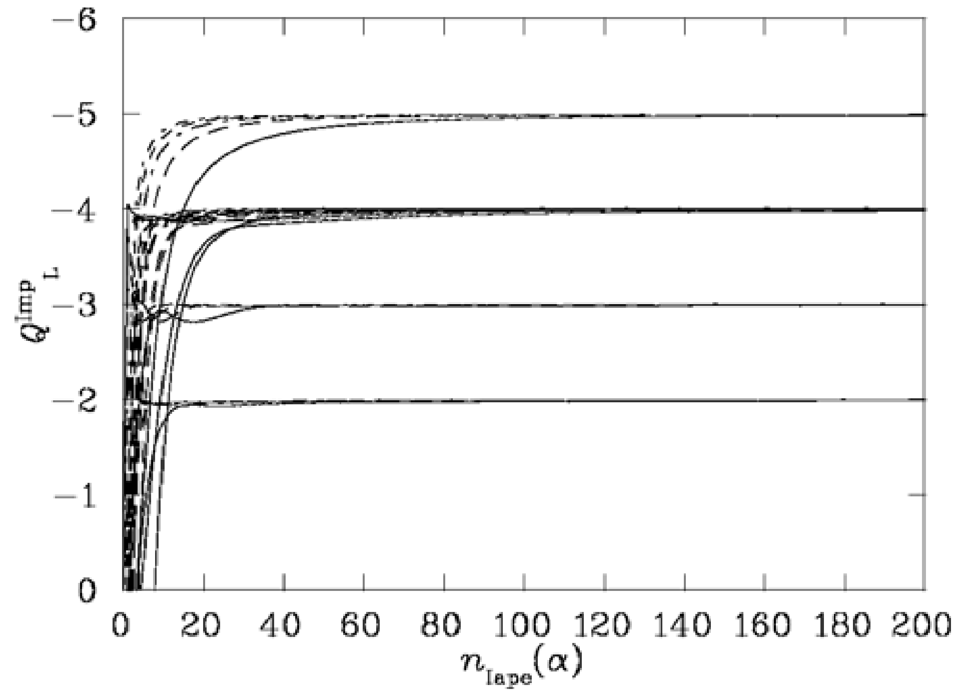


Figure 7.41: The evolution of $Q_{\text{L}}^{\text{Imp}}$ using APE smearing as a function of APE smearing sweep $n_{\text{Iape}}(\alpha)$ on the $24^3 \times 36$ lattice at $\beta = 5.00$. Here different line types correspond to different smearing fractions.

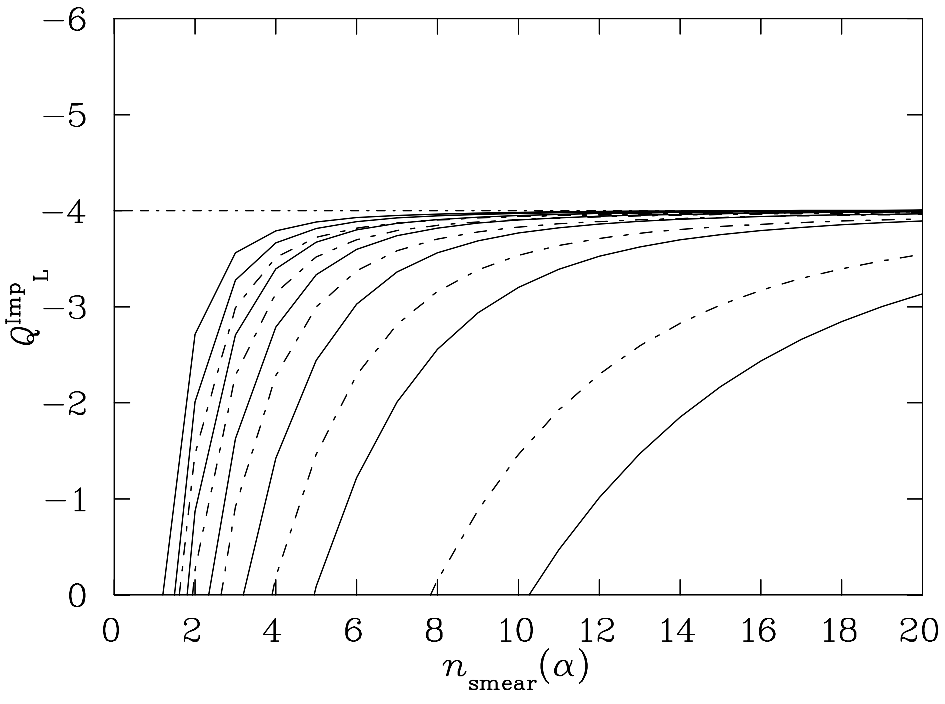


Figure 7.42: The evolution of the improved topological charge, $Q_{\rm L}^{\rm Imp}$, as a function of standard APE smearing sweeps, $n_{\rm ape}(\alpha)$, for $0.1 \le \alpha \le 0.7$ (solid lines) is compared to improved smearing sweeps, $n_{\rm Iape}(\alpha)$, (dotted-dashed lines) for the same smearing fractions $0.1 \le \alpha \le 0.5$ on the $24^3 \times 36$ lattice at $\beta = 5.00$. The horizontal dotted-dashed line is $Q_{\rm L}^{\rm Imp} = -4$.

These results are to be compared with Fig. 7.35 and Fig. 7.36, where transitions are observed even with improved cooling on the coarser lattice with $a = 0.165(2)$ fm. The results on our fine lattice suggest the characteristic size of instantons is much larger than the lattice spacing, such that the topological structure of the gauge fields is smooth at the scale of the lattice spacing.

In Fig. 7.40 for standard APE smearing we observe a slower convergence to integer topological charge than in Fig. 7.41 for improved smearing when $0.1 \le \alpha \le 0.5$. This feature of improved smearing is illustrated in detail in Fig. 7.42. However, APE smearing has the advantage to allow values for the smearing fraction up to $\alpha = 0.70$ which cannot be accessed by improved smearing.

Having demonstrated that it is possible to precisely match the behaviour of the algorithms on fine lattice spacings, we proceed to calibrate the efficiency of these algorithms in the following section.

7.8.4 Smoothing Algorithm Calibration

I now calibrate the relative rate at which quantum fluctuations are removed from typical field configurations by the various algorithms. The calibration is done using the action normalized to the single instanton action, S/S_0, on both, the $16^3 \times 32$ lattices and $24^3 \times 36$ lattices.

While there is no doubt that the algorithms may be accurately calibrated on the fine $24^3 \times 36$ lattice, the $16^3 \times 32$ lattice with $a = 0.165(2)$ fm presents more of a challenge. As a result, in most cases we will show the graphs produced from the $16^3 \times 32$ lattice

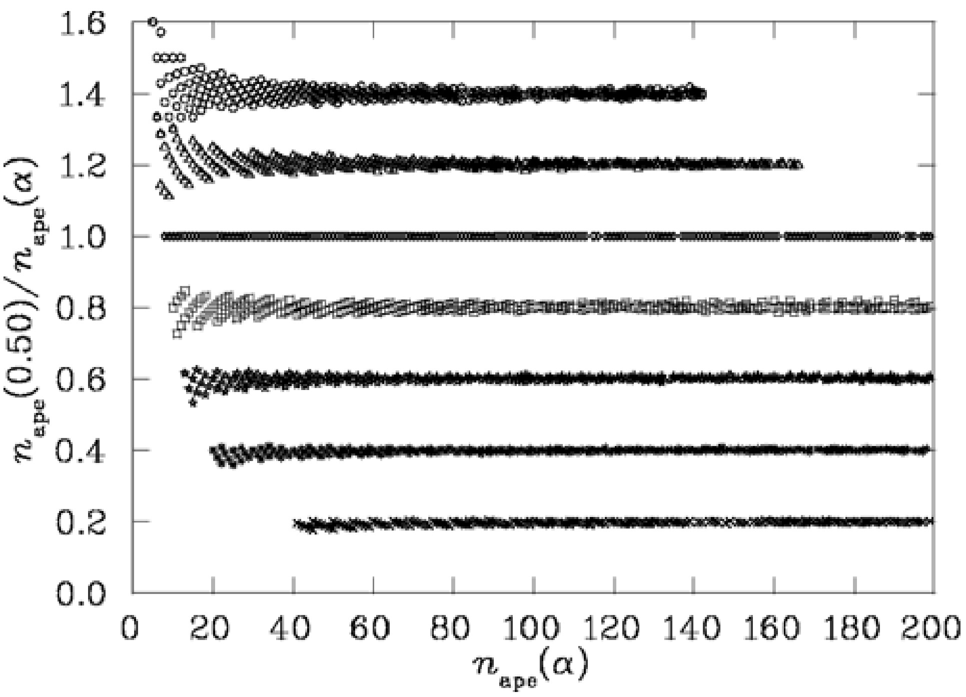


Figure 7.43: The ratio $n_{\rm ape}(0.50)/n_{\rm ape}(\alpha)$ versus $n_{\rm ape}(\alpha)$ for numerous S/S_0 thresholds on the $16^3 \times 32$ lattice at $\beta = 4.38$. From top to bottom the data point bands correspond to $\alpha = 0.7$, 0.6, 0.5, 0.4, 0.3, 0.2, and 0.1.

analysis and simply present the numerical results for both the $16^3 \times 32$ and $24^3 \times 36$ lattices.

The numerical results are summarized in Tables 7.4, 7.5, and 7.8 for the $16^3 \times 32$ lattices and in Tables 7.6, 7.7, and 7.9 for the $24^3 \times 36$ lattices.

APE Smearing and Improved Smearing Calibration

To calibrate the rate at which the algorithm reduces the action I record the nearest number of sweeps required to reach a given threshold in S/S_0. The action thresholds are spaced logarithmically to obtain a uniform distribution in the number of sweeps required to reach a threshold. The relative rates of smoothing are established by comparing the relative number of sweeps required to reach a particular threshold.

Here we calibrate the APE smearing algorithm characterized by the smearing fraction α and the number of smearing iterations $n_{\rm ape}(\alpha)$. The different threshold crossings are characterized by the number of sweeps required to reach that threshold, $n_{\rm ape}(\alpha)$. In Fig. 7.43 I show the number of sweeps required to reach a threshold when $\alpha = 0.5$, $n_{\rm ape}(0.50)$, relative to that required for other α values, $n_{\rm ape}(\alpha)$. I plot these relative smoothing rates as a function of $n_{\rm ape}(\alpha)$ such that low S/S_0 thresholds are reached after hundreds of iterations of the smoothing algorithm. Fig. 7.44 shows similar results for improved smearing. In these figures and in the following analysis, thresholds that result in fewer than five smoothing iterations are omitted as these points produce integer discretization errors of more than 20%.

Both standard and improved smearing algorithms have a relative smoothing rate which is independent of the amount of smoothing done. By calculating the average value for each of the bands in Figs. 7.43 and 7.44, we can investigate the dependence

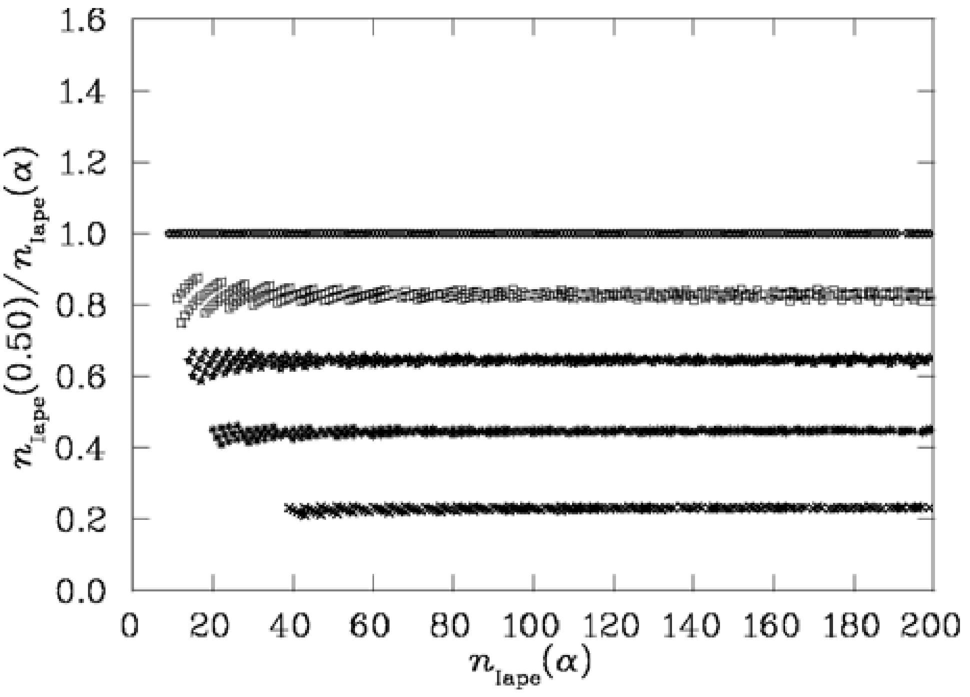


Figure 7.44: The ratio $n_{\mathrm{Iape}}(0.50)/n_{\mathrm{Iape}}(\alpha)$ versus $n_{\mathrm{Iape}}(\alpha)$ for numerous S/S_0 thresholds on the $16^3 \times 32$ lattice at $\beta = 4.38$. From top to bottom the data point bands correspond to $\alpha = 0.5$, 0.4, 0.3, 0.2, and 0.1.

of the average relative smoothing rate $\langle n_{\mathrm{ape}}(0.50)/n_{\mathrm{ape}}(\alpha)\rangle$ on α.

Fig. 7.45 illustrates a linear fit to the data constrained to pass through the origin. I find $\langle n_{\mathrm{ape}}(0.50)/n_{\mathrm{ape}}(\alpha)\rangle = 2.00\,\alpha$ such that

$$\frac{n_{\mathrm{ape}}(\alpha')}{n_{\mathrm{ape}}(\alpha)} = \frac{\alpha}{\alpha'}, \tag{7.56}$$

is in agreement with the earlier analysis, Eq. (7.46), carried out in Sec. 7.7.1 . The extent to which this relationship holds can be verified by plotting the ratio $\alpha' n_{\mathrm{ape}}(\alpha')/\alpha n_{\mathrm{ape}}(\alpha)$ and comparing the results to 1.

In plotting the band averages for the improved smearing algorithm of Fig. 7.44 in Fig. 7.46, one finds a small deviation of the points from the line $y = 2\,\alpha$. This suggests that Eq. (7.56) is not sufficiently general for the improved smearing case.

A better approximation to establish the α dependence, that is similar to Eq. (7.56) and contains Eq. (7.56) is

$$\frac{n_{\mathrm{Iape}}(\alpha')}{n_{\mathrm{Iape}}(\alpha)} = \left(\frac{\alpha}{\alpha'}\right)^{\delta}, \tag{7.57}$$

where δ is equal to one in the case of standard APE smearing and can deviate away from one for improved smearing.

In Fig. 7.47, the logarithm of Eq. (7.57) is plotted. The slope of the data provides $\delta = 0.914(1)$ for both the $16^3 \times 32$ and the $24^3 \times 36$ lattices. The value of $\delta = 1.00$ was also verified for the APE smearing data. Fig. 7.48 plots the ratio of the left- and right-hand sides of Eq. (7.57) as a function of $n_{\mathrm{Iape}}(\alpha)$ for $\alpha' > \alpha$. The ratio is one as expected with 5% for large amounts of smearing where integer discretization errors are minimized. Throughout the following analysis, δ is fixed at 0.914(1).

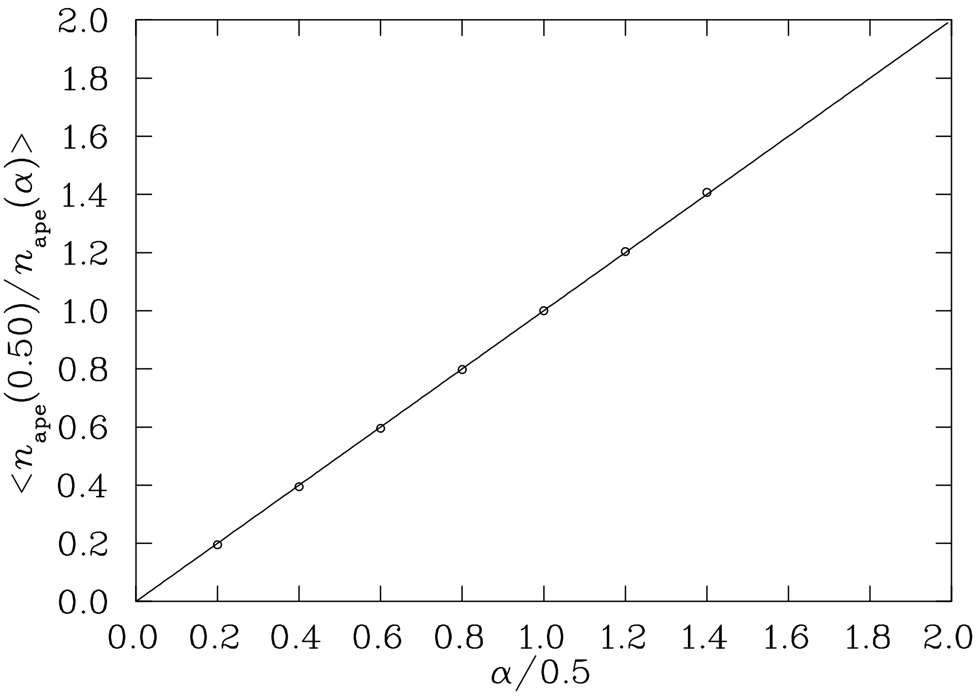


Figure 7.45: Illustration of the dependence of $\langle n_{\rm ape}(0.50)/n_{\rm ape}(\alpha)\rangle$ for APE smearing on the smearing fraction α. The solid line is a linear fit to the data constrained to pass through the origin.

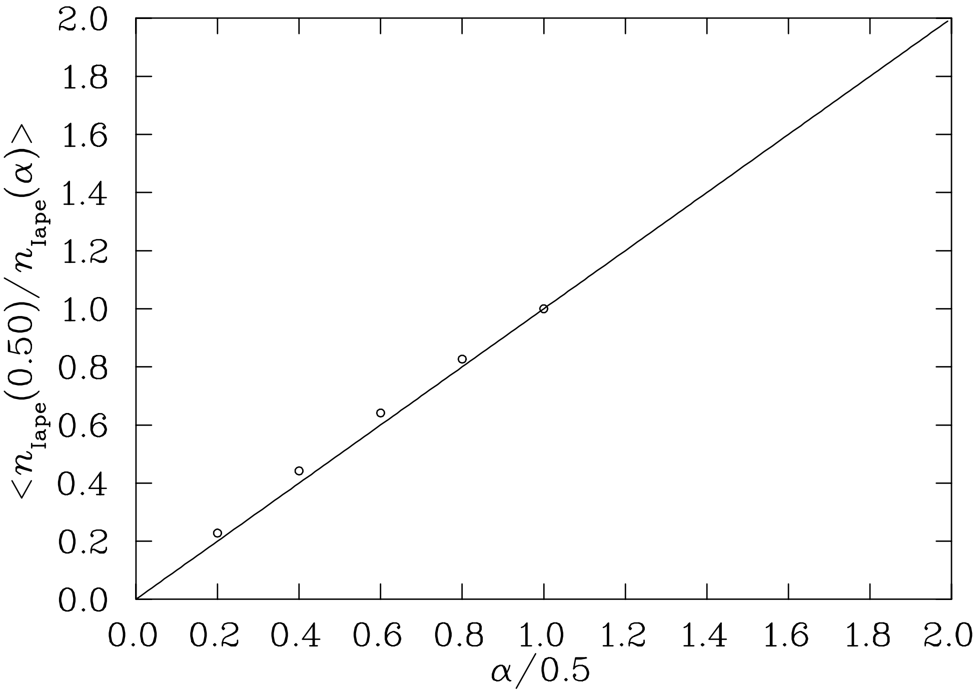


Figure 7.46: Illustration of the dependence of $< n_{\rm Iape}(0.50)/n_{\rm Iape}(\alpha) >$ for APE smearing on the improved smearing fraction α. The solid line fit is constrained to pass through the origin.

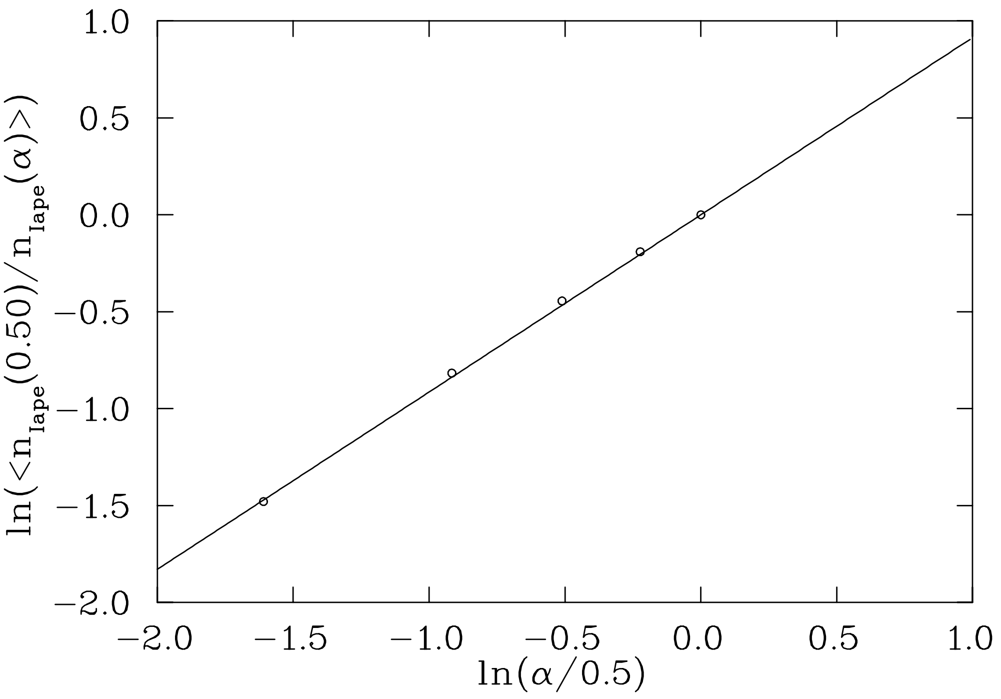


Figure 7.47: Illustration of the dependence of $\ln(< n_{\text{Iape}}(0.50)/n_{\text{Iape}}(\alpha) >)$ on the improved smearing fraction α for improved smearing. The solid line fit indicates $\delta = 0.914$.

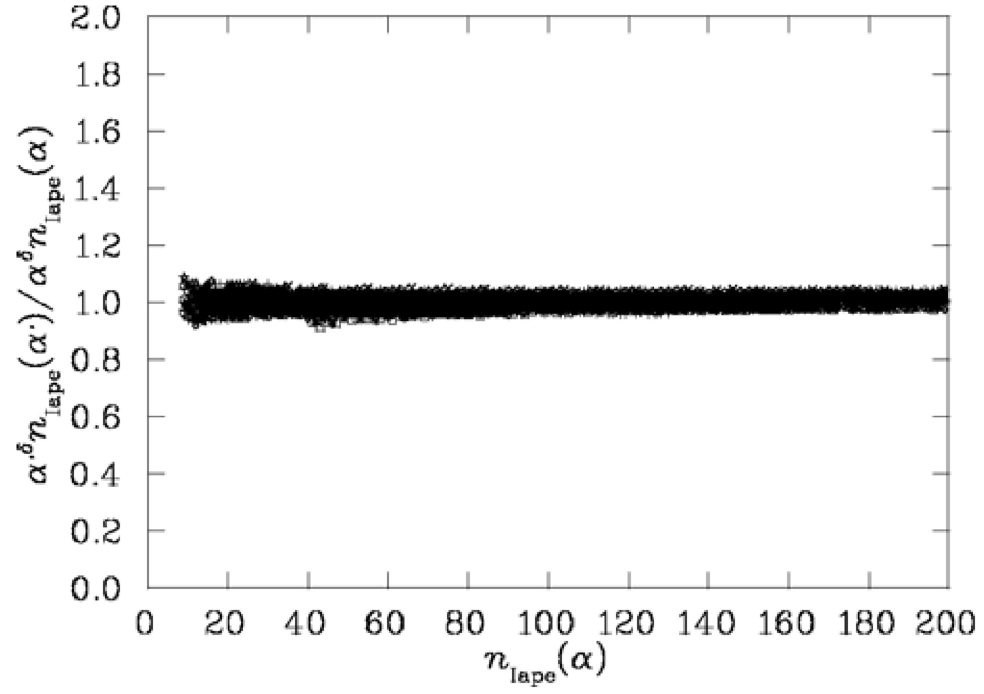


Figure 7.48: Illustration of the degree to which the relation Eq. (7.57) is satisfied for improved smearing. Here the entire data set is plotted for α and $\alpha' = 0.5$, 0.4, 0.3, 0.2 and 0.1. Data are from $16^3 \times 32$ lattice at $\beta = 4.38$.

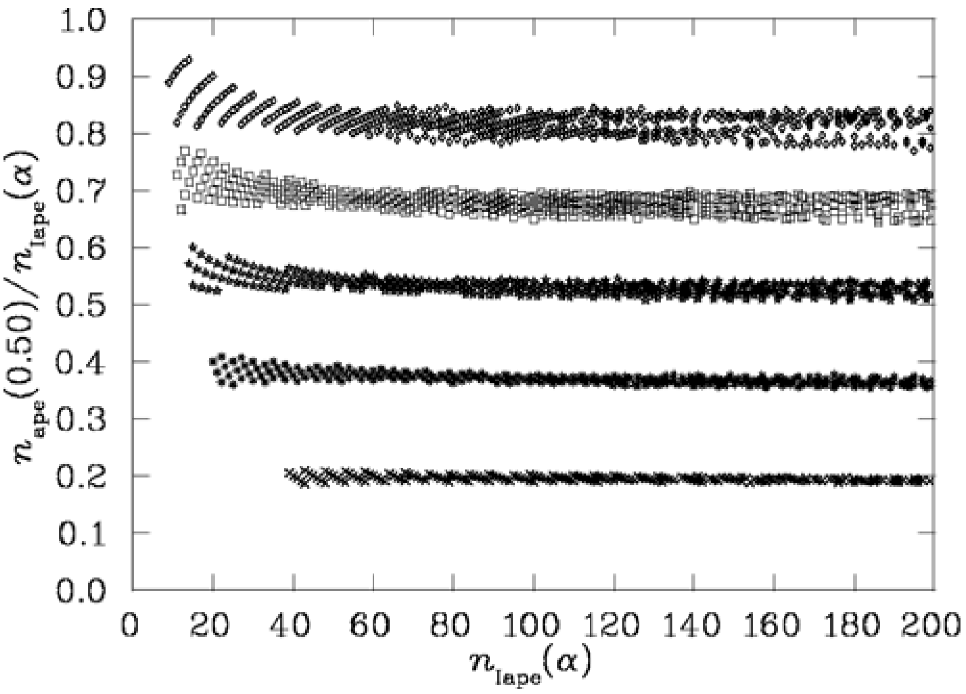


Figure 7.49: The ratio $n_{\mathrm{ape}}(0.50)/n_{\mathrm{Iape}}(\alpha)$ versus $n_{\mathrm{Iape}}(\alpha)$ for numerous threshold actions on the $16^3 \times 32$ lattice at $\beta = 4.38$. From top to bottom the data point bands correspond to improved smearing fractions $\alpha = 0.5$, 0.4, 0.3, 0.2, and 0.1.

APE and Improved smearing cross calibration

In this section I focus on the cross calibration of the smearing algorithms. In Fig. 7.49, the number of improved smearing sweeps required to reach a threshold for various improved smearing fractions relative to APE smearing at $\alpha = 0.5$ is compared. The lowest band corresponds to an improved smearing fraction of 0.10. From this, we conclude that for low α values APE smearing and improved smearing produce roughly equivalent smeared configurations. However, there are some evident differences in the rate at which both algorithms perform. For intermediate to large α there is curvature in the bands. Early in the smearing process, fewer sweeps of improved smearing are required to reach a threshold. That is, improved smearing removes action faster than APE smearing in the early stages of smearing. This behaviour is also manifest in the analogous results for the fine $24^3 \times 36$ lattice. As emphasized in the discussion surrounding Fig. 7.42, improved smearing also provides a topological charge closer to an integer than APE smearing. Together, these two properties of improved smearing identify a genuine improvement in the smearing process.

For the coarse $16^3 \times 32$ lattice data, the bands are thick for large smearing fractions indicating improved smearing does perform significantly different from standard APE smearing. Contributions from individual configurations are clearly visible as lines within the bands. This structure is due to the coarse lattice spacing of 0.165(2) fm which reveals differences between the algorithms. Such structure is not seen in the fine $24^3 \times 36$ lattice results. There a precise calibration is possible.

In Tables 7.4 and 7.5, we report the averages of each band for APE and improved smearing on the $16^3 \times 32$ lattices. In Tables 7.6 and 7.7 we report similar results for the $24^3 \times 36$ lattices.

Table 7.4: The averages of the ratios $< n_{\rm ape}(0.50)/n_{\rm ape}(\alpha) >$ and $< n_{\rm Iape}(0.50)/n_{\rm ape}(\alpha) >$ for various smearing fractions α from the $16^3 \times 32$ lattice at $\beta = 4.38$.

	α for APE smearing						
	0.10	0.20	0.30	0.40	0.50	0.60	0.70
$n_{\rm ape}(0.50)$	0.195(1)	0.394(2)	0.595(3)	0.797(4)	1.0	1.203(1)	1.407(1)
$n_{\rm Iape}(0.50)$	0.227(1)	0.465(1)	0.706(1)	0.948(1)	1.189(1)	1.431(1)	1.673(1)

Table 7.5: The averages of the ratios $< n_{\rm ape}(0.50)/n_{\rm Iape}(\alpha) >$ and $< n_{\rm Iape}(0.50)/n_{\rm Iape}(\alpha) >$ for various smearing fractions α from the $16^3 \times 32$ lattice at $\beta = 4.38$.

	α for improved smearing				
	0.10	0.20	0.30	0.40	0.50
$n_{\rm ape}(0.50)$	0.196(1)	0.376(3)	0.543(1)	0.697(1)	0.842(1)
$n_{\rm Iape}(0.50)$	0.228(1)	0.442(1)	0.641(1)	0.827(1)	1.0

Table 7.6: The averages of the ratios $< n_{\rm ape}(0.50)/n_{\rm ape}(\alpha) >$ and $< n_{\rm Iape}(0.50)/n_{\rm ape}(\alpha) >$ for various smearing fractions α from the $24^3 \times 36$ lattice at $\beta = 5.00$.

	α for APE smearing						
	0.10	0.20	0.30	0.40	0.50	0.60	0.70
$n_{\rm ape}(0.50)$	0.195(1)	0.395(1)	0.595(1)	0.797(1)	1.0	1.205(1)	1.410(1)
$n_{\rm Iape}(0.50)$	0.227(1)	0.462(1)	0.698(1)	0.937(1)	1.176(1)	1.416(1)	1.658(1)

Table 7.7: The averages of the ratios $< n_{\rm ape}(0.50)/n_{\rm Iape}(\alpha) >$ and $< n_{\rm Iape}(0.50)/n_{\rm Iape}(\alpha) >$ for various smearing fractions α from the$24^3 \times 36$ lattice at $\beta = 5.00$.

	α for improved smearing				
	0.10	0.20	0.30	0.40	0.50
$n_{\rm ape}(0.50)$	0.196(1)	0.378(1)	0.546(1)	0.704(1)	0.851(1)
$n_{\rm Iape}(0.50)$	0.228(1)	0.442(1)	0.641(1)	0.826(1)	1.0

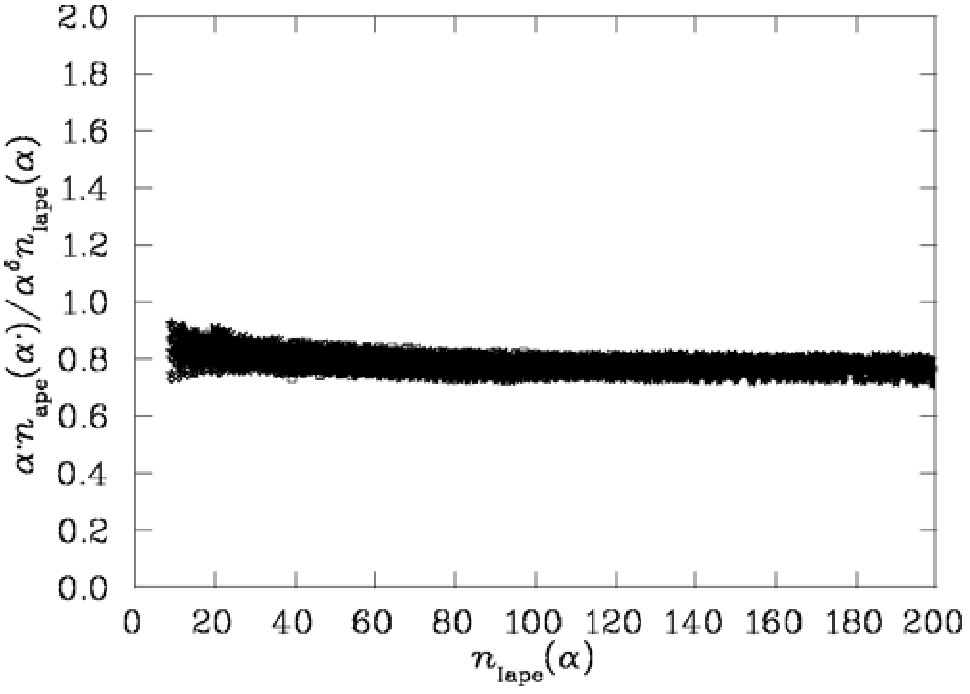


Figure 7.50: Illustration of the degree to which the relation Eq. (7.58) is satisfied for calibration of the action under APE and improved smearing. Here the entire data set is plotted.

Based on equations Eq. (7.56) and Eq. (7.57) for APE and improved smearing, we expect

$$\frac{\alpha' n_{\rm ape}(\alpha')}{\alpha^\delta n_{\rm Iape}(\alpha)} = \text{constant}\,. \tag{7.58}$$

This ratio is plotted in Fig. 7.50 where a rather mild dependence on $n_{\rm Iape}(\alpha)$ is revealed. Averaging these results provides 0.81(2) for the constant of Eq. (7.58). Similar results are seen for the finer $24^3 \times 36$ lattice, but with greater precision in the calibration reflected in a narrower band. There the constant is also 0.81(2). However, it should also be noted that for $\alpha \leq 0.5$, improved smearing achieves integer topological charge faster than standard APE smearing.

Calibration of Cooling and Smearing

In this section I apply the ansatz of equations Eq. (7.56) and Eq. (7.57) to relate the cooling and smearing algorithms. Fig. 7.51 displays results comparing cooling and standard APE smearing. For α as small as 0.1 it takes about 75 sweeps of APE smearing compared with 5 sweeps of cooling to arrive at an equivalent action. On the other end of the smearing fraction spectrum, we note the bands become very thick.

The calibration of these ratios indicates

$$\frac{n_{\rm c}}{\alpha n_{\rm ape}(\alpha)} = 0.59(1), \tag{7.59}$$

in agreement with that obtained in, Eq. (7.48), where the analysis was performed on unimproved gauge configurations. The reduction in $\mathcal{O}(a^2)$ errors in the gauge field action affect both algorithms similarly such that the calibration of the relative smoothing rates remains unaltered.

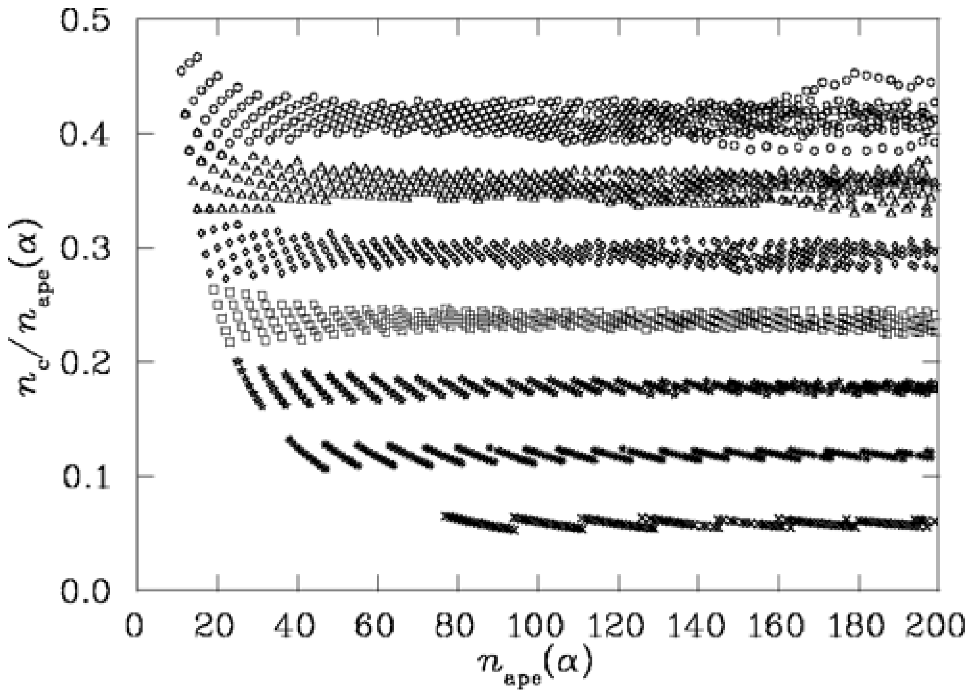


Figure 7.51: The ratio $n_{\rm c}/n_{\rm ape}(\alpha)$ versus $n_{\rm ape}(\alpha)$ for numerous action thresholds for the $16^3 \times 32$ lattice at $\beta = 4.38$. From top down the data point bands correspond to $\alpha = 0.7$, 0.6, 0.5, 0.4, 0.3, 0.2, and 0.1.

Further broadening of the bands is observed when comparing improved cooling with APE smearing as illustrated in Fig. 7.52. The precision of improved cooling relative to APE smearing leads to very different smoothed configurations at this coarse lattice spacing of 0.165(2) fm. This indicates the algorithms are sufficiently different, that an accurate and meaningful calibration is impossible.

This effect is not observed when we pass to our fine lattice spacing as displayed in Fig. 7.53. We remind the reader that the thickness of the band for small numbers of smoothing sweeps is simply due to the ratio of small integers taken in plotting the y-axis values.

The real test of improved smearing is the extent to which the algorithm can preserve action associated with topological objects and thus maintain better agreement with more precise algorithms including cooling and improved cooling. Fig. 7.54 displays results for the calibration of improved cooling with improved smearing. Comparing these results for each smearing fraction, α, with that for improved cooling and standard smearing in Fig. 7.52 reveals that the improved smearing algorithm, which was seen to be better than standard APE smearing algorithm does not perform as well as the improved cooling algorithm.

Similar results are seen in Fig. 7.55 where standard cooling is compared with improved smearing. Hence the annealing of the links in the process of cooling, where cooled links are immediately passed into the determination of the next cooled link, is key to the precision with which cooling can preserve topological structure.

Calibration of the smoothing rates as measured by the action for the algorithms under investigation are summarized in Tables 7.8 and 7.9. The entries describe the relative smoothing rate for the algorithm ratio formed by selecting an entry from the numerator column and dividing it by the heading of the denominator columns. The entry comparing APE smearing with itself reports the level to which the ansatz of

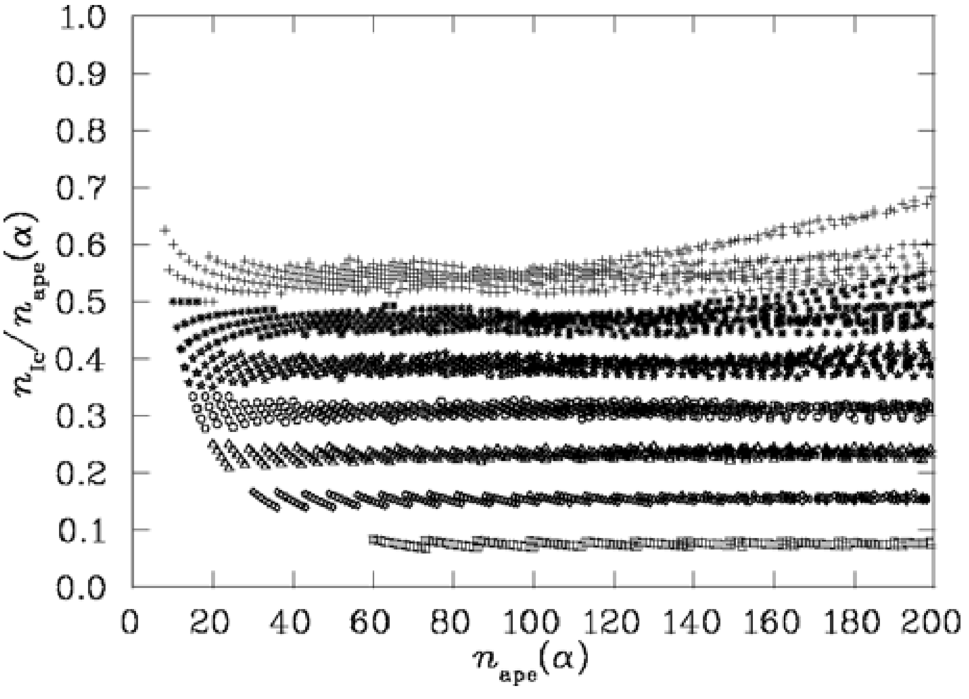


Figure 7.52: The ratio $n_{\rm Ic}/n_{\rm ape}(\alpha)$ versus $n_{\rm ape}(\alpha)$ for numerous action thresholds on the $16^3 \times 32$ lattice at $\beta = 4.38$. From top down the data point bands correspond to $\alpha = 0.7$, 0.6, 0.5, 0.4, 0.3, 0.2, and 0.1.

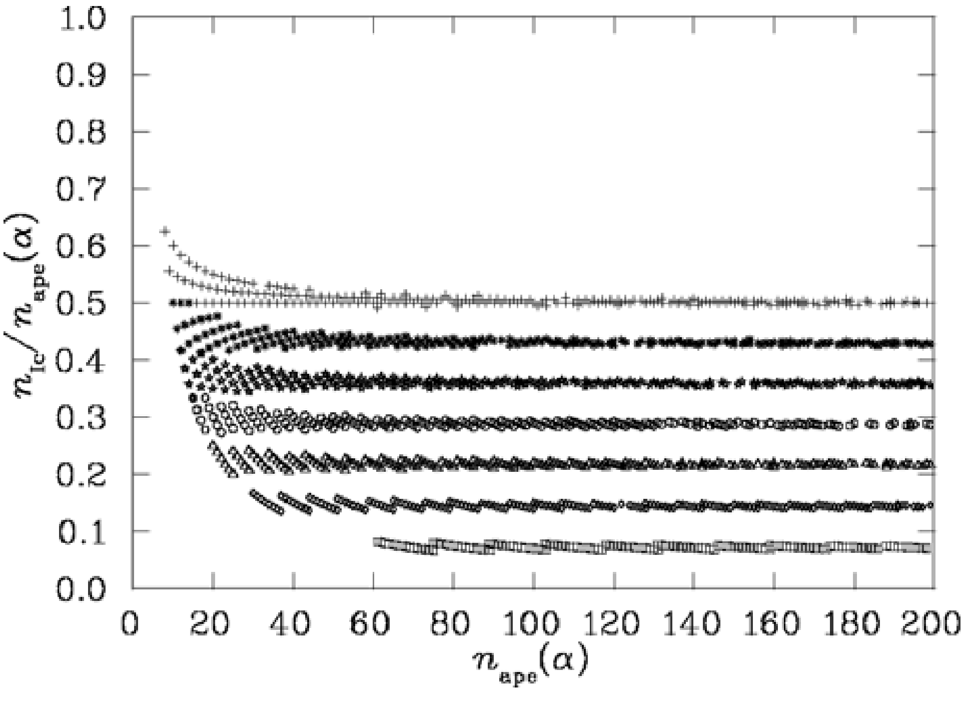


Figure 7.53: The ratio $n_{\rm Ic}/n_{\rm ape}(\alpha)$ versus $n_{\rm ape}(\alpha)$ for numerous action thresholds on the $24^3 \times 36$ lattice at $\beta = 5.00$. From top down the data point bands correspond to $\alpha = 0.7$, 0.6, 0.5, 0.4, 0.3, 0.2, and 0.1.

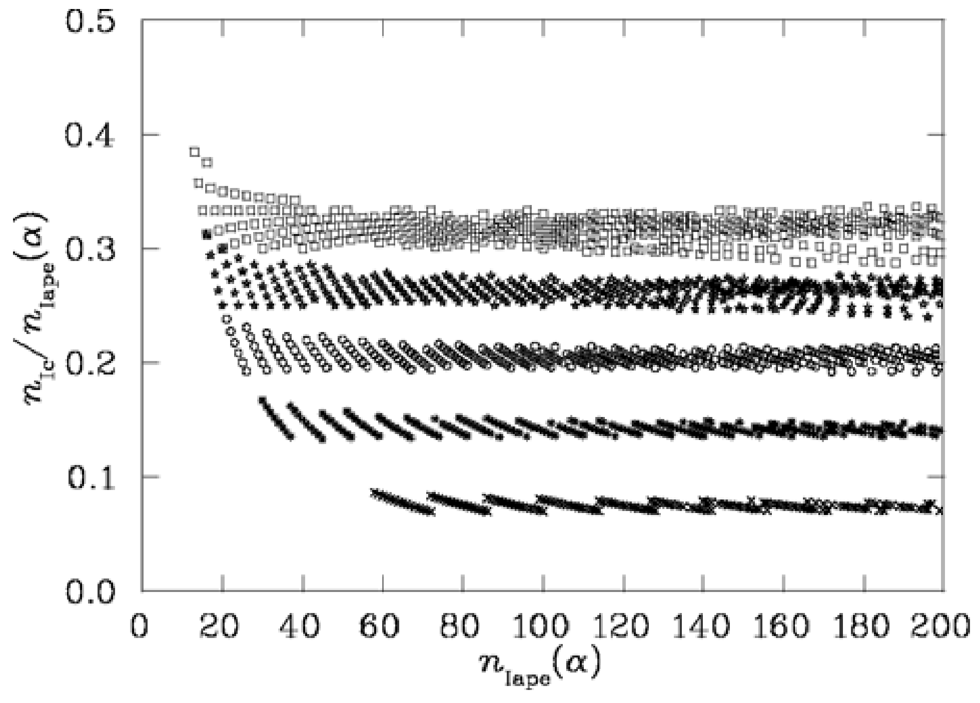


Figure 7.54: The ratio $n_{\mathrm{Ic}}/n_{\mathrm{Iape}}(\alpha)$ versus $n_{\mathrm{Iape}}(\alpha)$ for numerous action thresholds on the $16^3 \times 32$ lattice at $\beta = 4.38$. From top down the data point bands correspond to $\alpha = 0.5$, 0.4, 0.3, 0.2, and 0.1.

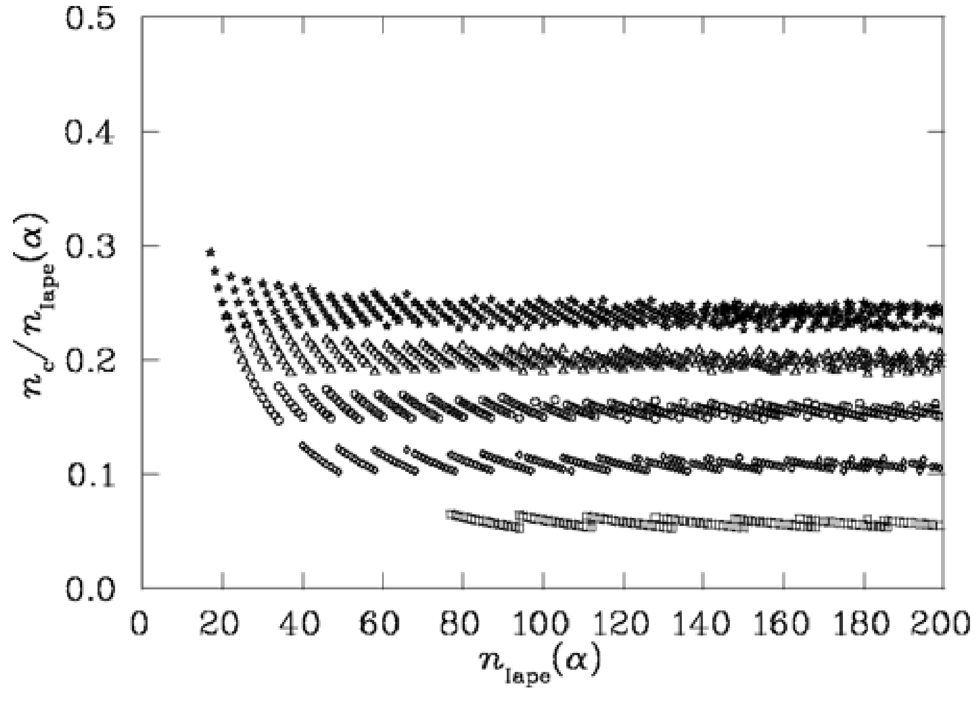


Figure 7.55: The ratio $n_{\mathrm{c}}/n_{\mathrm{Iape}}(\alpha)$ versus $n_{\mathrm{Iape}}(\alpha)$ for numerous action thresholds on the $16^3 \times 32$ lattice at $\beta = 4.38$. From top down the data point bands correspond to $\alpha = 0.5$, 0.4, 0.3, 0.2, and 0.1.

Table 7.8: Calibration coefficients for various smoothing algorithms on the $16^3 \times 32$ lattice at $\beta = 4.38$. Entries describe the relative smoothing rate for the algorithm ratio formed by selecting an entry from the numerator column and dividing it by the heading of the denominator columns. For example equation Eq. (7.59) corresponds to the first column of the third row.

	Denominator			
Numerator	$\alpha\, n_{\rm ape}(\alpha)$	$\alpha^{\delta} n_{\rm Iape}(\alpha)$	$n_{\rm c}$	$n_{\rm Ic}$
$\alpha'\, n_{\rm ape}(\alpha')$	1.00(2)	0.81(2)	1.69(3)	1.30(2)
$\alpha'^{\delta} n_{\rm Iape}(\alpha')$	1.25(3)	1.01(2)	2.13(5)	1.61(3)
$n_{\rm c}$	0.59(1)	0.47(1)	1	0.75(1)
$n_{\rm Ic}$	0.77(1)	0.62(1)	1.33(2)	1

Table 7.9: Calibration coefficients for various smoothing algorithms on the $24^3 \times 36$ lattice at $\beta = 5.00$. Entries describe the relative smoothing rate for the algorithm ratio formed by selecting an entry from the numerator column and dividing it by the heading of the denominator columns. For example equation Eq. (7.58) corresponds to the second column of the first row.

	Denominator			
Numerator	$\alpha\, n_{\rm ape}(\alpha)$	$\alpha^{\delta} n_{\rm Iape}(\alpha)$	$n_{\rm c}$	$n_{\rm Ic}$
$\alpha'\, n_{\rm ape}(\alpha')$	1.00(2)	0.81(2)	1.64(2)	1.37(1)
$\alpha'^{\delta} n_{\rm Iape}(\alpha')$	1.25(3)	1.01(2)	2.04(4)	1.67(3)
$n_{\rm c}$	0.611(9)	0.49(1)	1	0.84(1)
$n_{\rm Ic}$	0.734(8)	0.60(1)	1.19(1)	1

equation Eq. (7.56) is satisfied. Similarly the entry comparing improved smearing with itself reports the level to which the ansatz of equation Eq. (7.57) is satisfied.

Cooling versus Improved cooling

Figure 7.56 reports a comparison of standard cooling with improved cooling on eleven configurations from the coarse $16^3 \times 32$ lattice. There the ratio $n_{\rm c}/n_{\rm Ic} < 1$ confirms the expectation that standard cooling does not preserve action on the lattice as well as the $\mathcal{O}(a^2)$-improved cooling. Fewer standard cooling sweeps are required to reach the same action threshold. Calibration of the algorithms appears plausible for the first 80 sweeps of improved cooling, after which the two algorithms smooth the configurations in very different manners. Any calibration at this lattice spacing is only very approximate beyond 80 sweeps of improved cooling where distinct configuration-dependent trajectories become visible. This result is contrasted by the analogous analysis on our fine $24^3 \times 36$ lattice illustrated in Fig. 7.57. While $n_{\rm c}/n_{\rm Ic}$ remains less than one, it is closer to one here than for the coarser lattice as one might expect.

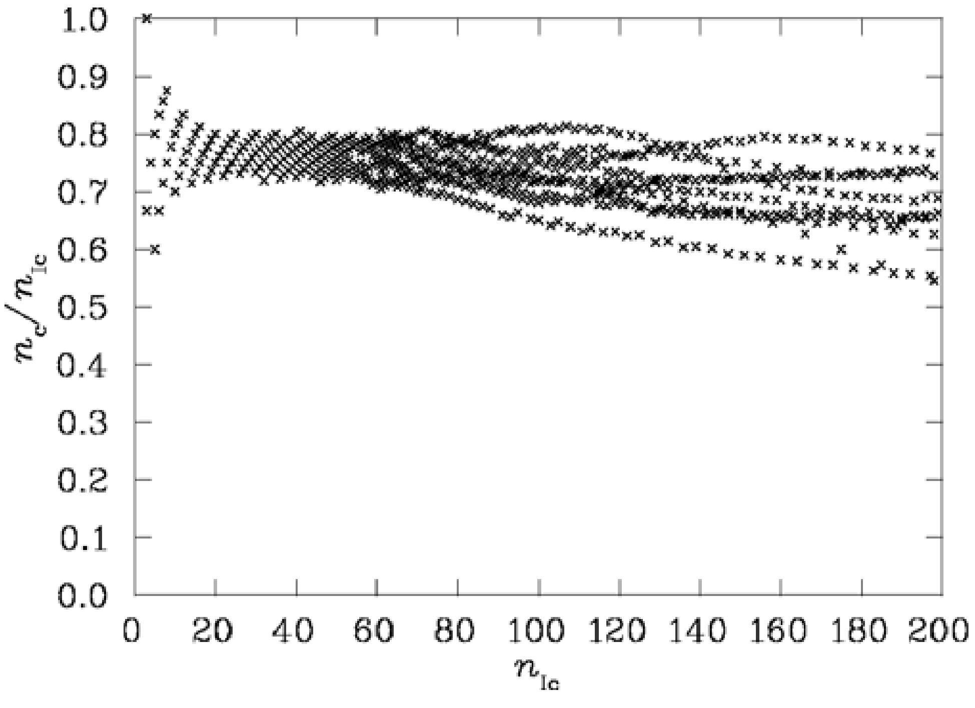


Figure 7.56: The ratio n_c/n_{Ic} versus n_{Ic} for numerous action thresholds on the $16^3 \times 32$ lattice at $\beta = 4.38$. The significant differences between the algorithms are revealed by the gauge-configuration dependence of the trajectories.

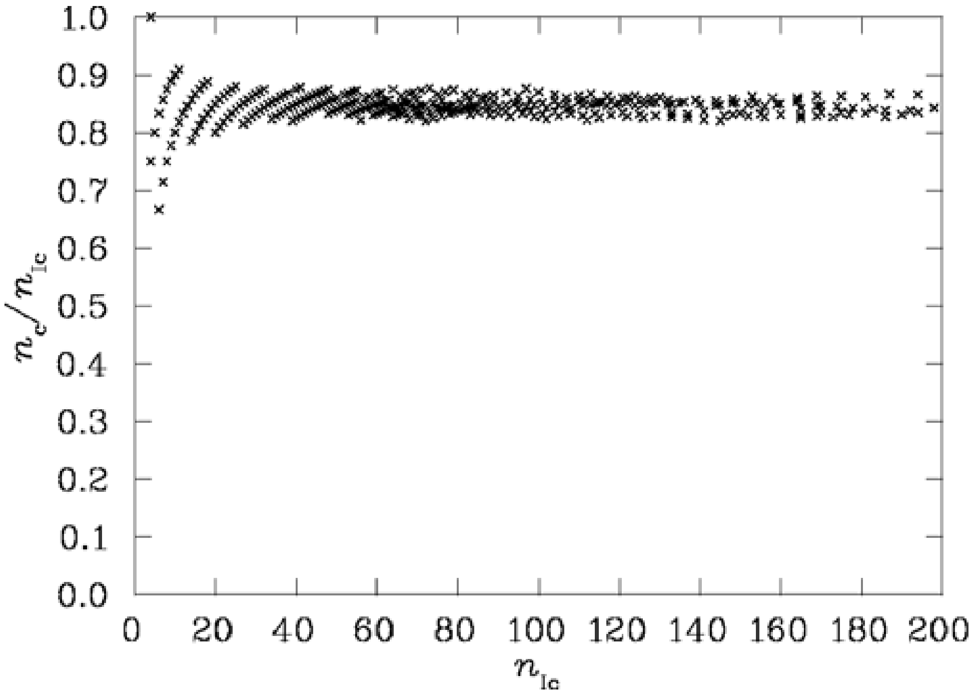


Figure 7.57: The ratio n_c/n_{Ic} versus n_{Ic} for numerous action thresholds on the $24^3 \times 36$ lattice at $\beta = 5.00$.

7.8.5 Summary

I have introduced an improved version of the APE smearing algorithm founded on the connection between cooling Eq. (7.24) and the projection of the APE smeared link back to the $SU_c(3)$ gauge group via Eq. (7.38). This connection motivates the use of additional extended paths combined with the standard "staple" as governed by the action to reduce the introduction $\mathcal{O}(a^2)$ errors in the smearing projection process.

Clear signs of improvement are observed. For a given smearing fraction α defined in equation (7.32), improved smearing preserves the action better than standard APE smearing at each smearing sweep. At the same time improved smearing brings the improved topological charge to an integer value faster than standard APE smearing.

The extended nature of the "staple" in improved smearing reduces the stability regime for the smearing fraction. I found the improved smearing algorithm to be stable for $\alpha \leq 0.5$. At $\alpha = 0.6$ the algorithm is unstable whereas standard APE smearing remains stable for $\alpha \leq 0.75$.

Given the wide variety of smoothing algorithms under investigation in the field of lattice gauge theory, I have cross calibrated the speed with which the algorithms remove action from the field configurations. In particular I have cross calibrated the smoothing rates of APE smearing at seven values of the smearing fraction; improved smearing at five values of the smearing fraction; cooling; and improved cooling. I explored smearing fractions in 0.1 intervals starting at $\alpha = 0.1$.

The calibration has been investigated over a range of 200 sweeps for each smearing algorithm on $\mathcal{O}(a^2)$-improved gauge field configurations. The results of this analysis allows one to make qualitative comparisons between cooling and smearing algorithms and in fact make quantitative comparisons of smearing algorithms with different smearing fractions on lattices as coarse as 0.165(2) fm. On our fine lattice where the lattice spacing is 0.077(1) fm, the calibration is quantitative in general.

We have found the relative smoothing rates are described via simple relationships as reported in Tables 7.8 and 7.9 for our coarse $16^3 \times 32$ and fine $24^3 \times 36$ lattices respectively. There the sensitivity of the calibration results on the lattice spacing may be reviewed.

It is worth noting, that a necessary correction to the APE smearing ratio rule, Eq. (7.46), needs to be made when improved smearing is considered. These algorithms may be calibrated via

$$\frac{n_{\rm ape}(\alpha')}{n_{\rm ape}(\alpha)} = \frac{\alpha}{\alpha'}\,, \qquad \text{and} \qquad \frac{n_{\rm Iape}(\alpha')}{n_{\rm Iape}(\alpha)} = \left(\frac{\alpha}{\alpha'}\right)^{\delta}\,, \tag{7.60}$$

for APE smearing and improved smearing respectively. I find $\delta = 0.914(1)$ without a significant dependence on the lattice spacing.

We have seen evidence that the topology of Yang-Mills gauge fields cannot be reliably studied on lattice spacings as coarse as 0.165(2) fm. Different algorithms lead to different topological charges, differing quite widely in some cases as reported in Figs. 7.35 and 7.36. Moreover, subtle differences in the cooling algorithms can lead to different topological charge determinations as illustrated in Figs. 7.27 and 7.28. These results indicate that the characteristic size of topological fluctuations in Yang-Mills gauge fields is at the scale of the coarse lattice spacing of 0.165(2) fm.

In contrast, the fine $24^3 \times 36$ lattice results where $a = 0.077(1)$ fm display excellent agreement among every smoothing algorithm considered. In this case it appears that the lattice spacing on these $\mathcal{O}(a^2)$-improved gauge field configurations is finer than the characteristic size of topological fluctuations such that the gauge fields are already sufficiently smooth to unambiguously extract the topology of the gauge fields.

As a final comparison of the smoothing algorithms, we provide a visual representation of a gauge field configuration after applying various smoothing algorithms is provided. Figure 7.58 illustrates a rendering of the topological charge density for a slice of one of the fine $24^3 \times 36$ lattice configurations. While the calibration has been carried out by considering the total action of the gauge fields, the following analysis allows one to examine the extent to which the calibration is accurate at a microscopic level.

In Fig. 7.58, red shading indicates large positive topological charge density with decreasing density becoming yellow in colour, while blue shading indicates large in magnitude, negative topological charge density decreasing in magnitude through the colour green. Here cooling (a), improved cooling (b), APE smearing at $\alpha = 0.70$ (c), APE smearing at $\alpha = 0.30$ (d), improved smearing at $\alpha = 0.50$ (e) and improved smearing at $\alpha = 0.30$ (f), are compared at the number of smoothing iterations required for each algorithm to produce an approximately equivalent smoothed gauge field configuration. While Fig. b) for improved cooling differs somewhat due to round off in the sweep number, the remaining plots compare very favourably with each other. These visualizations confirm that the different smoothing algorithms considered in this investigation can be accurately related via the calibration analysis presented here and summarized in Tables 7.8 and 7.9.

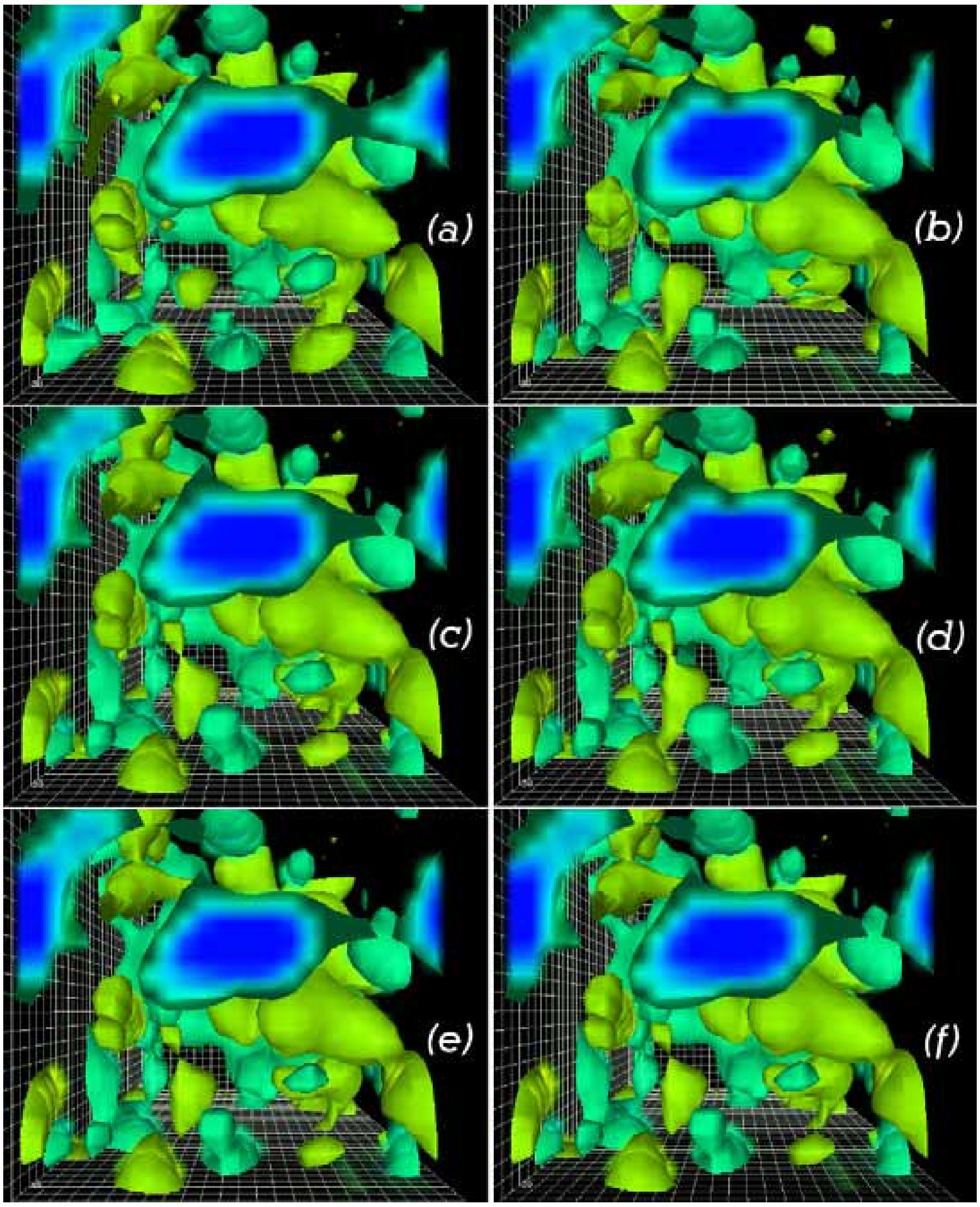


Figure 7.58: The topological charge density of a $24^3 \times 36$ lattice for fixed x coordinate. The instantons (anti-instantons) are coloured red to yellow (blue to green). Fig. a) shows the topological charge density after 9 cooling sweeps. Each of the following figures display the result of a different smoothing algorithm calibrated according to Table 7.9 to reproduce as closely as possible the results depicted in Fig. a). Fig. b) illustrates the topological charge density after 11 sweeps of improved cooling. Fig. c) shows the topological charge density after 21 APE smearing steps at $\alpha = 0.70$. Fig. d) illustrates the topological charge density after 49 APE smearing steps at $\alpha = 0.30$. In Fig. e) the topological charge density is displayed after 35 sweeps of improved smearing at $\alpha = 0.50$. Finally, Fig. f) shows the topological charge density after 55 sweeps of improved smearing at $\alpha = 0.30$. Apart from Fig. b) for improved cooling, which differs largely due to round off in the sweep number, all the plots compare very favourably with each other.

Chapter 8

Gauge Fixing in Landau Gauge

In certain applications in lattice QCD like the study of the gluon [62] or quark propagator [77, 78] or any other gauge dependent quantities, it is of crucial importance to fix the gauge prior engaging into such calculation in order to make any sense of the lattice data. There are various ways in fixing the gauge, here we are going to focus our attention on the Landau gauge and on ways to alternate this standard gauge fixing definition using a mean-field-improved perturbation theory [10] to compare different lattice definitions of the Landau gauge, and quantify the sizes of the discretisation errors. On the lattice, the standard Landau gauge condition[56] is the same as the continuum condition, $\sum_\mu \partial_\mu A_\mu = 0$, only to leading order in the lattice spacing a. In particular, I review [63] how a new $\mathcal{O}(a^2)$ improved Landau-gauge-fixing functional is derived and how it plays a central role in estimating the discretisation errors made with the standard functionals. These results primarily due to Patrick Bowman may be found in [63].

8.1 Landau Lattice Gauge Fixing

On the lattice, the idea of the gauge fixing method is based on the maximization of the usual Landau gauge fixing functional [56]

$$\mathcal{F}_1^G[\{U\}] = \sum_{\mu,x} \frac{1}{2}\mathrm{Tr}\,\left\{U_\mu^G(x) + U_\mu^G(x)^\dagger\right\}, \tag{8.1}$$

where the gauge rotated links are defined by

$$U_\mu^G(x) = G(x)U_\mu(x)G(x+\hat{\mu})^\dagger \quad \text{and} \quad \mathrm{U_\mu(x)} \equiv \mathcal{P}\exp\left\{\mathrm{ig}\int_0^{\mathrm{a}} \mathrm{dtA_\mu(x+\hat{\mu}t)}\right\}. \tag{8.2}$$

with gauge transformation proportional to the generators of the gauge group:

$$G(x) = \exp\left\{-i\sum_a \omega^a(x)T^a\right\}. \tag{8.3}$$

In the continuum the maximization condition, of Eq. (8.1), corresponds to finding the stationary points of

$$F[G] = ||A^G|| = \int d^4x\mathrm{Tr}\,\left(A_\mu^G(x)\right)^2. \tag{8.4}$$

Where $A_\mu^G(x)$ are the gauge rotated gluon fields.

If we take the functional derivative of Eq. (8.1), we obtain

$$\frac{\delta \mathcal{F}_1^G}{\delta \omega^a(x)} = \frac{1}{2} i \sum_\mu \mathrm{Tr}\left\{ \left[U_\mu^G(x-\hat{\mu}) - U_\mu^G(x) - \left(U_\mu^G(x-\hat{\mu}) - U_\mu^G(x) \right)^\dagger \right] T^a \right\}. \tag{8.5}$$

The continuum limit of this functional is obtained by performing a Taylor expansion on the gluon field $A_\mu(x+\hat{\mu}t)$ about the point x, and then integrating term-by-term followed by expanding the exponential as a power series in g up to leading order. Eq. (8.5) then become

$$\frac{\delta \mathcal{F}_1^G}{\delta \omega^a(x)} = g a^2 \sum_\mu \mathrm{Tr}\left\{ \left[\partial_\mu A_\mu(x) + \frac{1}{12} a^2 \partial_\mu^3 A_\mu(x) + \frac{a^4}{360} \partial_\mu^5 A_\mu(x) + \mathcal{O}(a^6) \right] T^a \right\} + \mathcal{O}(g^3 a^4). \tag{8.6}$$

To lowest order in a, an extremum of Eq. (8.1) satisfies $\sum_\mu \partial_\mu A_\mu(x) = 0$, which corresponds to the continuum Landau gauge condition. In praxis, this means that on the lattice that

$$\sum_\mu \partial_\mu A_\mu(x) = \sum_\mu \left\{ -\frac{a^2}{12} \partial_\mu^3 A_\mu(x) - \mathcal{H}_1 \right\}. \tag{8.7}$$

The operator $\mathcal{H}_1$ represents $\mathcal{O}(a^4)$ and higher-order terms, as it is written in Eq. (8.6). The terms on the R.H.S. of Eq. (8.7) are significantly large compared to the numerical accuracy possible in gauge fixing algorithms.

Using two-link terms, one could write down an alternative gauge fixing,

$$\mathcal{F}_2^G = \sum_{x,\mu} \frac{1}{2} \mathrm{Tr}\left\{ U_\mu^G(x) U_\mu^G(x+\hat{\mu}) + \text{h.c.} \right\}. \tag{8.8}$$

Performing the same steps as for Eq. (8.5) and expanding to $\mathcal{O}(a^4)$ we obtain

$$\frac{\delta \mathcal{F}_2^G}{\delta \omega^a(x)} = 4 g a^2 \sum_\mu \mathrm{Tr}\left\{ \left[\partial_\mu A_\mu(x) + \frac{a^2}{3} \partial_\mu^3 A_\mu(x) + \frac{16}{360} a^4 \partial_\mu^5 A_\mu(x) + \mathcal{O}(a^6) \right] T^a \right\} + \mathcal{O}(g^3 a^4), \tag{8.9}$$

which again implies the continuum Landau-gauge-fixing condition to lowest order in a.

$\mathcal{O}(a^2)$ errors can be eliminated in the same as for the gauge action, that is done by simply taking a linear combination of the two functionals $\mathcal{F}_1^G$ and $\mathcal{F}_2^G$:

$$\frac{\delta \left\{ \frac{4}{3} \mathcal{F}_1^G - \frac{1}{12 u_0} \mathcal{F}_2^G \right\}}{\delta \omega^a(x)} = g a^2 \sum_\mu \mathrm{Tr}\left\{ \left[\partial_\mu A_\mu(x) - \frac{4}{360} a^4 \partial_\mu^5 A_\mu(x) + \mathcal{O}(a^6) \right] T^a \right\} + \mathcal{O}(g^3 a^4). \tag{8.10}$$

The mean-field (tadpole) improvement parameter u_0, Eq. (5.19), was introduced to ensure that our perturbative calculation is not spoiled by large renormalisations [10].

Note that the higher order $g^3 a^4$ terms of Eqs. (8.6) (8.9) and (8.10) are to be viewed in terms of the mean-field-improved perturbation theory [10]. We shall define this improved functional as:

$$\mathcal{F}_{\text{Imp}}^G \equiv \frac{4}{3} \mathcal{F}_1^G - \frac{1}{12 u_0} \mathcal{F}_2^G. \tag{8.11}$$

The algorithm for fixing the Landau gauge are mostly local. Local algorithms generate, at each sites, a gauge matrix that maximizes the functional in question. this matrix is then applied to the lattice. This procedure is iterated over until the global maximum is found. The algorithm of preference is the "steepest descents" approach [56]. Absorbing terms of $\mathcal{O}(a^4)$ and higher into the $\mathcal{H}_i$, three difference operators can be constructed, namely:

$$\begin{aligned}\Delta_1(x) &\equiv \frac{1}{u_0}\sum_\mu [U_\mu(x-\mu) - U_\mu(x) - \text{h.c.}]_{\text{traceless}} \\ &= -2iga^2 \sum_\mu \left\{\partial_\mu A_\mu(x) + \frac{a^2}{12}\partial_\mu^3 A_\mu(x) + \mathcal{H}_1\right\} \ , \end{aligned} \tag{8.12}$$

$$\begin{aligned}\Delta_2(x) &\equiv \frac{1}{4u_0^2}\sum_\mu [U_\mu(x-2\mu)U_\mu(x-\mu) - U_\mu(x)U_\mu(x+\mu) - \text{h.c.}]_{\text{traceless}} \\ &= -2iga^2 \sum_\mu \left\{\partial_\mu A_\mu(x) + \frac{a^2}{3}\partial_\mu^3 A_\mu(x) + \mathcal{H}_2\right\} \ , \end{aligned} \tag{8.13}$$

$$\begin{aligned}\Delta_{\text{Imp}}(x) &\equiv \frac{4}{3}\Delta_1(x) - \frac{1}{3}\Delta_2(x) \\ &= -2iga^2 \sum_\mu \left\{\partial_\mu A_\mu(x) + \mathcal{H}_{\text{Imp}}\right\} . \end{aligned} \tag{8.14}$$

where the subscript, "traceless" denotes subtraction of the average of the colour-trace from each of the diagonal colour elements. The resulting gauge transformation is

$$\begin{aligned}G_i(x) &= \exp\left\{\frac{\alpha}{2}\Delta_i(x)\right\} \\ &= 1 + \frac{\alpha}{2}\Delta_i(x) + \mathcal{O}(\alpha^2). \end{aligned} \tag{8.15}$$

The parameter α can be tuned and will vary according to the lattice spacing a. The index i denotes the choice of operator taken into consideration, and will depend on order of the gauge action. During the iterative process, the gauge transformation, $G_i(x)$, is reunitarized through an orthonormalization procedure similar to the one described in Section 4.3 with the omission of the Lorentz index and with an $SU_c(3)$ matrix attached at each lattice sites instead of an $SU_c(2)$ as shown in Appendix. E.5, the routine may be found in Appendix. E.22.

The algorithm first calculates the relevant Δ_i, and then applies the associated gauge transformation, Eq. (8.15), to the gauge field.

8.2 Monitoring Discretisation Errors on the Lattice

The convergence criterion to Landau gauge is usually monitored by a quantity such as

$$\theta_i = \frac{1}{VN_c}\sum_x \text{Tr}\left\{\Delta_i(x)\Delta_i(x)^\dagger\right\} , \tag{8.16}$$

where N_c is the number of colours, and V represents the lattice volume. The quantity is designed to approach a prefixed quality factor as we sweep through the lattice, in praxis this number is of the order of 10^{-12}. When this criterion is reached the algorithm stops.

We may rewrite Eq. (8.13) by substituting in Eq. (8.7), leading to

$$\begin{aligned}\Delta_2(x) &= -2iga^2 \sum_\mu \left\{ -\frac{a^2}{12} \partial_\mu^3 A_\mu(x) + \frac{a^2}{3} \partial_\mu^3 A_\mu(x) - \mathcal{H}_1 + \mathcal{H}_2 \right\} \\ &= -2iga^2 \sum_\mu \left\{ \frac{a^2}{4} \partial_\mu^3 A_\mu(x) - \mathcal{H}_1 + \mathcal{H}_2 \right\} . \end{aligned} \tag{8.17}$$

In the same way, we may rewrite the improved term as:

$$\Delta_{\text{Imp}}(x) = -2iga^2 \sum_\mu \left\{ -\frac{a^2}{12} \partial_\mu^3 A_\mu(x) - \mathcal{H}_1 + \mathcal{H}_{\text{Imp}} \right\} . \tag{8.18}$$

Eq. (8.18) provides an estimate of the absolute size of these discretisation errors, because the improved measure has no $\mathcal{O}(a^2)$ error of its own.

The exploration of the gauge fixing errors was done on 6^4 lattices at a variety of β for both the standard Wilson gauge action, Eq. (2.19), and the $1{\times}2$, $2{\times}1$ improved gauge action, Eq. (5.16). These are defined in Sec. 2.1.3 and Sec. 5.1 respectively. Details of the simulation is put into a tabular form in Table. 8.1. The gauge field configuration were generated in the same way as in the simulations described in Chapter 7, and were gauge fixed using Conjugate Gradient Fourier Acceleration [58] with a quality stopping criterion of $\theta_1 < 10^{-12}$.

8.2.1 Numerical Simulations

Once the gauge field configuration was gauge fixed, θ_{Imp} and θ_2 were measured, giving the possibility to see the size of the residual higher order terms. In fig. 8.1, I show the evolution of the gauge fixing measures as a function of iteration. The data is for a 6^4 lattice with Wilson action at $\beta = 6.0$.

The same procedure was repeated with each of the other two functionals on all the lattices mentioned in Table. 8.1, a summary of the results may be found in Table. 8.2. The first six columns of Table. 8.2 correspond to the results obtained when the gauge field configurations are generated with the standard Wilson gauge action while the remaining six columns, on the left hand side of the table, are for the $1{\times}2$, $2{\times}1$ improved gauge action.

We can now compare the results coming from different functionals, if we compare Eq. (8.7) with Eq. (8.18), we observe that the improved measure, θ_{Imp}, consist entirely

Table 8.1: Parameters of the gauge fixing numerical simulation.

Action	Volume	N_{Therm}	N_{Samp}	β	a (fm)	Physical Volume (fm)
Wilson	$6^3 \times 6$	5000	1000	5.70	0.18	$1.08^3 \times 1.08$
Wilson	$6^3 \times 6$	5000	1000	6.00	0.10	$0.6^3 \times 0.6$
Wilson	$6^3 \times 6$	5000	1000	6.20	0.07	$0.42^3 \times 0.42$
Improved	$6^3 \times 6$	5000	1000	3.92	0.353	$2.12^3 \times 2.12$
Improved	$6^3 \times 6$	5000	1000	4.38	0.165(2)	$0.99^3 \times 0.99$
Improved	$6^3 \times 6$	5000	1000	5.00	0.077(1)	$0.46^3 \times 0.46$

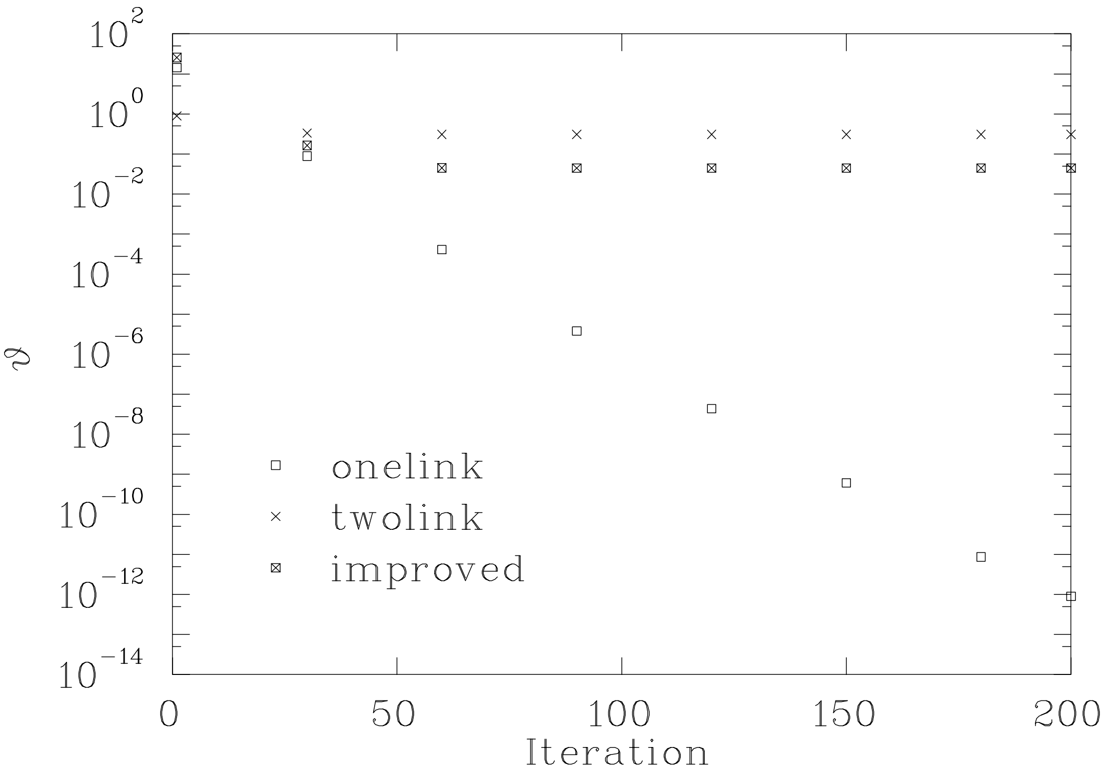


Figure 8.1: The gauge fixing measures for a 6^4 lattice with Wilson action at $\beta = 6.0$. This lattice was gauge fixed with Δ_1, so θ_1 drops steadily whilst θ_2 and $\theta_{\rm Imp}$ plateau at much higher values.

Wilson						Improved					
β	u_0	$\mathcal{F}$	$\theta_{\rm Imp}$	θ_2	$\frac{\theta_{\rm Imp}}{\theta_2}$	β	u_0	$\mathcal{F}$	$\theta_{\rm Imp}$	θ_2	$\frac{\theta_{\rm Imp}}{\theta_2}$
5.7	0.865	1	0.0679	0.611	0.111	3.92	0.837	1	0.102	0.921	0.111
6.0	0.877	1	0.0578	0.520	0.111	4.38	0.880	1	0.0585	0.526	0.111
6.2	0.886	1	0.0522	0.469	0.111	5.00	0.904	1	0.0410	0.369	0.111
β	u_0	$\mathcal{F}$	$\theta_{\rm Imp}$	θ_1	$\frac{\theta_1}{\theta_{\rm Imp}}$	β	u_0	$\mathcal{F}$	$\theta_{\rm Imp}$	θ_1	$\frac{\theta_1}{\theta_{\rm Imp}}$
5.7	0.865	2	59.0	33.2	0.563	3.92	0.837	2	57.5	32.3	0.563
6.0	0.877	2	65.1	36.6	0.563	4.38	0.880	2	53.4	30.0	0.563
6.2	0.886	2	61.7	34.7	0.563	5.00	0.904	2	52.2	29.4	0.563
β	u_0	$\mathcal{F}$	θ_1	θ_2	$\frac{\theta_1}{\theta_2}$	β	u_0	$\mathcal{F}$	θ_1	θ_2	$\frac{\theta_1}{\theta_2}$
5.7	0.865	Imp	0.0427	0.684	0.0625	3.92	0.837	Imp	0.0638	1.02	0.0625
6.0	0.877	Imp	0.0367	0.588	0.0625	4.38	0.880	Imp	0.0366	0.586	0.0625
6.2	0.886	Imp	0.0332	0.531	0.0625	5.00	0.904	Imp	0.0261	0.417	0.0625

Table 8.2: Left side of the table, the Values of the gauge-fixing measures obtained using the Wilson gluon action on 6^4 lattices at three values of the lattice spacing, fixed to Landau gauge with the one-link, two-link and improved functionals respectively. The ones obtained using the improved gluon action on 6^4 lattices at three values of the lattice spacing are shown in the right side of the table.

of discretization errors when the one–link functional is used to gauge fix a configuration to Landau gauge. Looking at table 8.2, shows that at $\beta = 6.0$,

$$\begin{aligned}\theta_{\rm Imp} &= \frac{1}{VN_c}\sum_x \mathrm{Tr}\left\{\Delta_{\rm Imp}(x)\,(\Delta_{\rm Imp}(x))^\dagger\right\} \\ &= \frac{1}{VN_c}\sum_x \mathrm{Tr}\left\{\sum_\mu \partial_\mu A_\mu(x)\left(\sum_\nu \partial_\nu A_\nu(x)\right)^\dagger\right\} = 0.058, \qquad (8.19)\end{aligned}$$

which is a significant deviation from the continuum Landau gauge if compared with the prefixed quality stopping criterion, $\theta_1 < 10^{-12}$.

As a check of the simulations, we note that the definition for $\Delta_{\rm Imp}$, Eq. (8.14), provides a constraint on the measures when gauge fixed. For example

$$\frac{\theta_{\rm Imp}}{\theta_2} = \frac{(-\frac{1}{12})^2}{(-\frac{1}{12}+\frac{1}{3})^2} = \frac{1}{9} \simeq 0.111. \qquad (8.20)$$

Similarly, by fixing with $\Delta_2(x)$ we expect

$$\frac{\theta_1}{\theta_{\rm Imp}} = \frac{(-\frac{1}{3}+\frac{1}{12})^2}{(-\frac{1}{3})^2} = \frac{9}{16} \simeq 0.563, \qquad (8.21)$$

and fixing with $\Delta_{\rm Imp}(x)$ leads to

$$\frac{\theta_1}{\theta_2} = \frac{1}{16} = 0.0625. \qquad (8.22)$$

These ratios are reproduced by the data of tables 8.2.

A gauge field configuration that is gauge fixed using $\Delta_{\rm Imp}(x)$ will satisfy

$$\sum_\mu \partial_\mu A_\mu(x) = \sum_\mu \{-\mathcal{H}_{\rm Imp}\}\,. \qquad (8.23)$$

Inserting this into Eq. (8.12) yields

$$\Delta_1(x) = -2iga^2\sum_\mu\left\{\frac{a^2}{12}\partial_\mu^3 A_\mu(x) + \mathcal{H}_1 - \mathcal{H}_{\rm Imp}\right\}, \qquad (8.24)$$

which differs from Eq. (8.18) only by an overall minus sign. This overall minus sign become irrelevant in the calculation of the corresponding θ_i.

One would expect that the three different methods to gauge fixed to Landau gauge presented above would all fix the gauge field configuration in the same way. This means, for example that, the $\theta_{\rm Imp}$ of a configuration fixed with Δ_1, would be equal to θ_1 when the gauge field configuration is fixed with $\Delta_{\rm Imp}$. Looking at Table. 8.2, we can see it is not the case, this tells us that higher-order derivative terms $\partial_\mu^n A_\mu(x)$ take different values depending on the gauge fixing functional used. Furthermore, the values in Table 8.2 reveals that in every case θ_1 is smaller when fixed with the improved functional than $\theta_{\rm Imp}$ under the one-link functional. This suggests that the additional long range information used by the improved functional is producing a gauge fixed configuration with smaller, higher-order derivatives; a secondary effect of improvement.

On the same principle, the value of θ_2 when fixed using the one-link functional, and θ_1 when fixed using the two-link functional may be compared. The differences that are produced from the two are rather large and once again attributed to differences in the size of higher-order derivatives of the gauge field. The two-link functional is coarser, it knows little about short range fluctuations, and fails to constrain higher-order derivatives. Similar conclusions are drawn from a comparison of θ_2 fixed with the improved functional and θ_{Imp} fixed with the two link functional.

It is also interesting to note that in terms of the absolute errors, the Wilson action at $\beta = 6.0$ is comparable to the improved lattice at $\beta = 4.38$, where the lattice spacing is three times larger. Therefore showing the gain in performance in passing to improved action.

To summarize: Using three different functionals to gauge fix to Landau gauge (the one-link and the two-link functionals have $\mathcal{O}(a^2)$ errors, the improved functional has $\mathcal{O}(a^4)$ errors) a method for estimating the discretisation errors has been devised. The results show that order $\mathcal{O}(a^2)$ improvement of the gauge fixing condition will improve comparison with the continuum Landau gauge in two ways: 1) through the elimination of $\mathcal{O}(a^2)$ errors and 2) through a secondary effect of reducing the size of higher-order errors.

These conclusions are robust with respect to lattice spacing and have also been verified by considering additional configurations to that presented here.

Chapter 9

Gluon Propagator with Improved action

9.1 Introduction

There has been considerable interest in the infrared behaviour of the gluon propagator, as a probe into the mechanism of confinement and as input for other calculations. Lattice gauge theory is an excellent means to study such nonperturbative behaviour. See, for example, Ref. [4, 3] for a recent survey. The analysis was primarily carried out by Dr. Patrick O. Bowman and partly inspired by the one carried out in hep-lat/0106023, see Sec. 7.8. While this gluon propagator work is not being presented here as my own Ph. D. qualifying work, I am a co–author on the subsequent papers and so I have therefore decided to include a review of this work.

Infrared quantities calculated in any lattice calculation may be affected by the finite volume of the lattice. Larger volumes mean either more lattice points (with increased computational cost) or coarser lattices (with corresponding discretization errors). The desire for larger physical volumes thus provides strong motivation for using improved actions. Improved actions have been shown to reduce discretization effects [25], although some concerns have been expressed that coarse lattices may miss important instanton physics.

In this chapter we are interested in the study of the gluon propagator in Landau gauge. The first task is to determine if some significant changes are seen in the infrared gluon propagator, even with a lattice spacing as coarse as 0.35 fm. We wish to find the strength of the gluon propagator in the infrared, see if the gluon propagator is infrared finite and see if this behaviour is similar to the one observed in three-dimensional $SU_c(2)$ studies [60]. This study is done in Landau gauge quenched QCD (pure $SU_c(3)$ Yang-Mills), using both standard Wilson gauge action and the mean-field (tadpole) improved [10] version of the tree–level $\mathcal{O}(a^2)$ Symanzik improved gauge action [69, 70, 71]. These are defined in Sec. 2.1.3 and Sec. 5.1 respectively. The gauge field configuration were generated with the Cabbibo-Marinari [21] pseudoheat-bath algorithm with three diagonal $SU(2)$ subgroups, the sampling was done in the same manner as in the simulations described in Chapter 7. The gauge field configurations were gauge fixed using the gauge fixing procedure described in Chapter 8 and in [63]. To ensure that the gauge dependent quantities remain $\mathcal{O}(a^2)$ improved, the $\mathcal{O}(a^2)$ improved gauge fixing functional, Eq. (8.11) was used. The various lattices used in this study are listed in Table 9.1, each lattice has a label for easy referencing (for example the lattice 1w would correspond to the lattice generated with the standard Wilson gauge action on a $16^3 \times 32$ lattice at a coupling of $\beta = 5.70$).

Using the various lattices, listed in Table 9.1, the gluon propagator has been calculated, see Table 9.1. Only two gluon propagators were generated using the standard Wilson gauge action, lattice 1w and 6. Lattice 1w is comparable to lattice 1i since they have the same lattice spacing hence roughly the same physical volume. Lattice 6 was generated and used in the study of the gluon propagator in Ref. [62]. The value for the tadpole factor was reported to be $u_0 = 0.878$, it has been used to normalize the propagator with respect to the other lattices. This lattice will be used here for comparison purposes only as it is the finest lattice out of the available set. All the other lattices were generated with the $\mathcal{O}(a^2)$ improved action.

The legend for denoting the data points is as follows. The data points coming from the momenta lying along a spatial Cartesian direction are displayed using a square, ($\Box$), while those lying in the temporal direction are denoted by triangles, ($\triangle$). Since the size of the temporal direction is larger than the spatial direction, the spread between the two symbols may indicate that the propagator is affected by the finite volume of the lattice. Points coming from just the four diagonal are marked by diamonds, ($\diamond$). Rotational symmetry breaking is manifested by the spread between the points coming from the diagonal and those coming from other direction.

In the continuum, the scalar function is rotationally invariant. Although the hypercubic lattice breaks O(4) invariance, it does preserve the subgroup of discrete rotations Z(4) up to finite volume effects. In our case, this symmetry is reduced to Z(3) as one dimension will be twice as long as the other three in each of the cases studied. We exploit this discrete rotational symmetry to improve statistics through Z(3) averaging. This is best explained through a simple example. Consider the propagator at momentum $q = (3, 2, 1, 4)$ (say). Z(3) symmetry means that

$$D(3,2,1,4) = D(2,3,1,4) = D(2,1,3,4) = D(1,2,3,4) = D(1,3,2,4) = D(3,1,2,4)$$

so we calculate the propagator for each of these values of momentum, and then average the results.

Table 9.1: Parameters of the gluon propagator numerical simulation. Lattices 1w and 1i have the same dimensions and approximately the same lattice spacing, but were generated with the Wilson and improved actions respectively. Lattice 6 was generated with the Wilson action. Hundred configurations were used for the lattice 1w,1i,2,3,4,5 and seventy-five for lattice 6.

Action	lattice	Volume	N_{Therm}	N_{Samp}	β	a fm	Physical Volume fm
Wilson	6	$32^3 \times 64$	-	1000	6.00	0.099	$3.18^3 \times 6.34$
Wilson	1w	$16^3 \times 32$	5000	1000	5.70	0.180(1)	$2.88^3 \times 5.76$
Improved	4	$16^3 \times 32$	5000	1000	3.92	0.352(3)	$5.63^3 \times 11.26$
Improved	1i	$16^3 \times 32$	5000	1000	4.38	0.165(2)	$2.64^3 \times 5.28$
Improved	5	$12^3 \times 24$	5000	1000	4.10	0.272(2)	$3.26^3 \times 6.53$
Improved	2	$10^3 \times 20$	5000	1000	3.92	0.352(3)	$3.52^3 \times 7.04$
Improved	3	$8^3 \times 16$	5000	1000	3.75	0.41(1)	$3.28^3 \times 6.56$

9.2 Landau Gauge Gluon Propagator

The dimensionless lattice gluon field $A_\mu(x)$ may be defined via

$$A_\mu(x+\hat{\mu}/2) = \frac{1}{2ig_0}\left(U_\mu(x) - U_\mu^\dagger(x)\right) - \frac{1}{6ig_0}\mathrm{Tr}\left(U_\mu(x) - U_\mu^\dagger(x)\right) , \tag{9.1}$$

accurate to $\mathcal{O}(a^2)$. The gauge links are defined in the usual manner, Eq. (2.1). While the infrared behaviour is unaffected by the improvement, the ultraviolet behaviour suffers due to the extended nature of an improved operator.

The gluon propagator in coordinate space

$$D_{\mu\nu}^{ab}(x,y) \equiv \langle\, A_\mu^a(x)\, A_\nu^b(y)\,\rangle , \tag{9.2}$$

is calculated using (9.1). To improve statistics, we use translational invariance and calculate

$$D_{\mu\nu}^{ab}(y) = \frac{1}{V}\langle \sum_x A_\mu^a(x) A_\nu^b(x+y)\,\rangle . \tag{9.3}$$

In this report we focus on the scalar part of the propagator,

$$D(y) = \frac{1}{N_d-1}\sum_\mu \frac{1}{N_c^2-1}\sum_a D_{\mu\mu}^{aa}(y), \tag{9.4}$$

where $N_d = 4$ and $N_c = 3$ are the number of dimensions and colours. This is then Fourier transformed followed by summing over the Lorentz components[1] of the Fourier transform

$$D(\hat{q}) = \frac{1}{N_d-1}\sum_\mu\sum_y e^{i\hat{q}\cdot y} D_{\mu\mu}(y) , \tag{9.5}$$

where the available momentum values, $\hat{q}$, are given by

$$\hat{q}_\mu = \frac{2\pi n_\mu}{aL_\mu}, \qquad n_\mu \in \left(-\frac{L_\mu}{2}, \frac{L_\mu}{2}\right] , \tag{9.6}$$

and L_μ is the length of the box in the μ direction. In the continuum, the scalar function $D(q^2)$ is related to the Landau gauge gluon propagator via

$$D_{\mu\nu}^{ab}(q) = (\delta_{\mu\nu} - \frac{q_\mu q_\nu}{q^2})\delta^{ab} D(q^2) . \tag{9.7}$$

The bare, dimensionless lattice gluon propagator $D(qa)$ is related to the renormalized continuum propagator $D_R(q;\mu)$ by

$$a^2 D(qa) = Z_3(\mu, a) D_R(q;\mu), \tag{9.8}$$

for momenta, q, sufficiently small compared to the cutoff, π/a. $D_R(q;\mu)$ is independent of a for sufficiently fine lattices; i.e., in the scaling regime. The renormalization constant $Z_3(\mu, a)$ is determined by imposing a renormalization condition at some chosen renormalization scale μ, e.g.,

$$D_R(q)|_{q^2=\mu^2} = \frac{1}{\mu^2}. \tag{9.9}$$

[1]The Landau gauge condition in momentum space, $q_\mu D_{\mu\nu}(q) = 0$ places a constraint on the Lorentz components of the propagator so that, for non-zero momentum, there are $N_d - 1$ degrees of freedom [60].

The renormalized gluon propagator can be computed both nonperturbatively on the lattice and perturbatively in the continuum for choices of the renormalization point in the ultraviolet. It can then be related to the propagator in other continuum renormalization schemes such as $\overline{\text{MS}}$. The first task is to compare our two improved, coarse lattices with the unimproved, finer one, it is sufficient to consider only their relative renormalizations. We have slightly rescaled the improved propagators so as to provide a reasonable match with old one. The relevant quantity is

$$Z_3(0.10)/Z_3(0.35) = 1.02. \tag{9.10}$$

All the propagators are plotted in physical units, which was obtain from the string tension [28] with $\sqrt{\sigma} = 440$ MeV.

9.3 Uncorrected Gluon Propagator

I first would like to show the gluon propagator as it comes out from the lattice simulation. The untouched gluon propagator calculated from the standard Wilson gauge action, lattice 1w, and from an improved gauge action 1i is shown in Figures 9.1 and 9.2 respectively. Both of these have been plotted as functions of $\hat{q}$, Eq. (9.6), for all available momenta. Both of these propagators show large divergences in the ultraviolet region, most of which appear more severe in the case of the standard Wilson action than in the case of the improved gauge action where the finite spacing errors do not exceed the infrared peak and the UV tail is generally flatter. In both of these figures neither tree–level correction or data cuts have been performed.

One way to deal with these divergences is to throw out some of the points in the ultraviolet by considering only momenta out to half of the Brillouin zone. For each of the four Cartesian directions,

$$\hat{q} \leq \frac{\pi}{2a}. \tag{9.11}$$

This momentum cut is referred to as the "half-cut" and in Fig. 9.3 and Fig. 9.4 we see that this removes the worst of the artifacts. This cut sacrifices data in the ultraviolet. Nevertheless, the two propagators, show plausible asymptotic behaviours, but there are still clear signs of lattice artifacts. As far as the infrared is concerned these two problems are of no significant weight, but as we will see, something as crude as the half-cut is not necessary and we can do much better at minimizing lattice artifacts.

Comparing Figs. 9.1 and 9.2, we see that the ultraviolet points are not so spread out with the improved gauge action, this shows an excellent restoration of rotational symmetry all the way to the edge of the Brillouin zone with an improved action.

9.4 Tree–level Correction on the Gluon Propagators

Perturbation theory tells us that the asymptotic behaviour of the gluon propagator, in the continuum, as $p^2 \to \infty$, has the form:

$$D(p) = \frac{1}{p^2}, \tag{9.12}$$

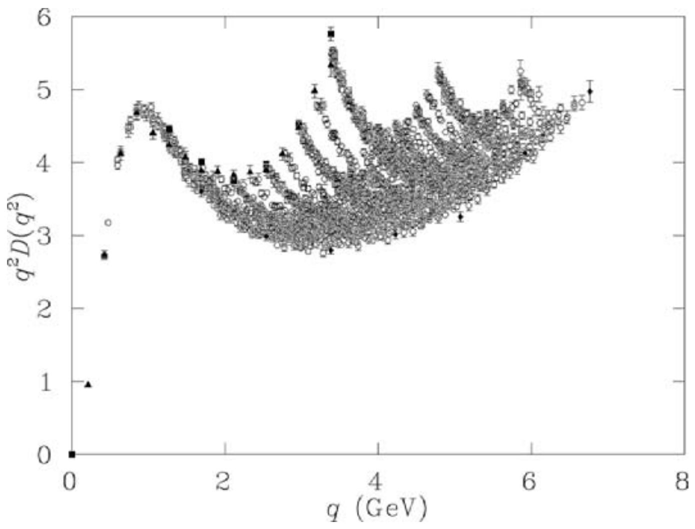


Figure 9.1: Uncorrected gluon propagator from lattice 1w ($\beta = 5.70, 16^3 \times 32$, Wilson action), plotted as a function of $\hat{q}$. The dramatic “fanning” is caused by finite spacing errors which quickly destroy the signal at large momenta.

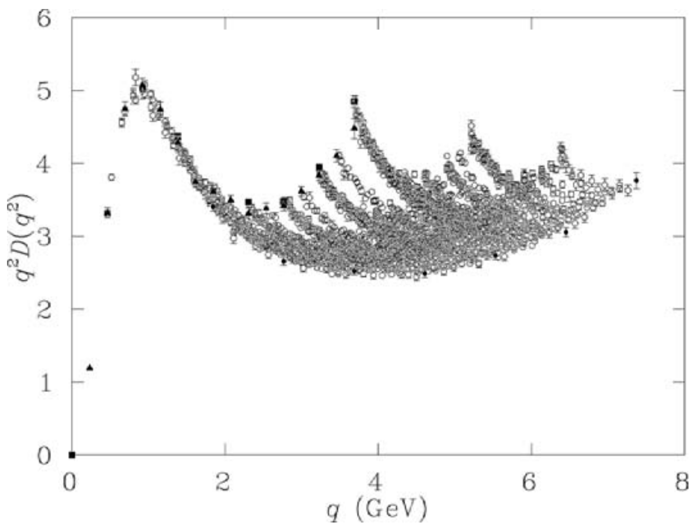


Figure 9.2: Uncorrected gluon propagator from lattice 1i ($\beta = 4.38, 16^3 \times 32$, improved action), plotted as a function of $\hat{q}$. Lattice artifacts are reduced by the improved action, but are still large.

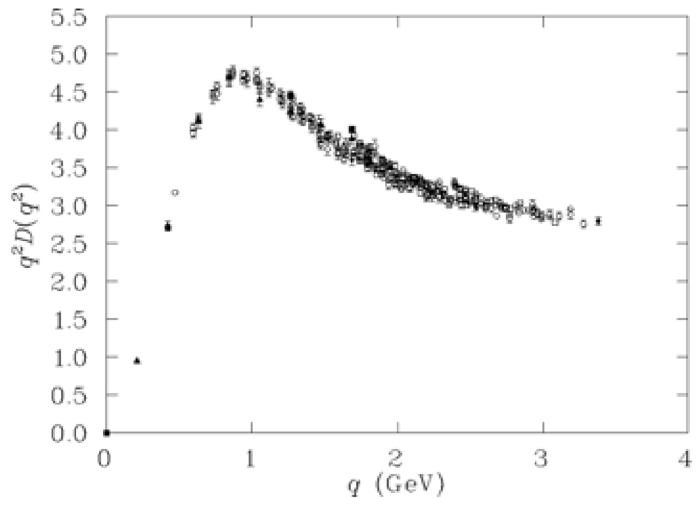


Figure 9.3: Uncorrected gluon propagator from lattice 1w ($\beta = 5.70, 16^3 \times 32$, Wilson action), plotted as a function of $\hat{q}$ with the momentum "half-cut" applied.

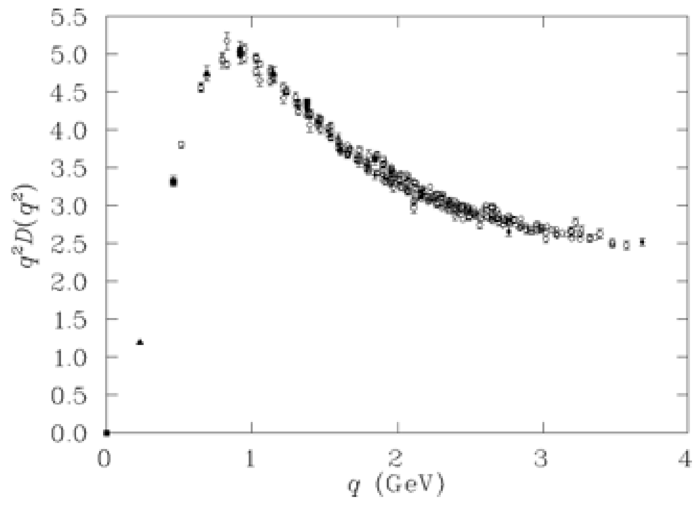


Figure 9.4: Uncorrected gluon propagator from lattice 1i ($\beta = 4.38, 16^3 \times 32$, improved action), plotted as a function of $\hat{q}$ with the momentum "half-cut" applied. The improved propagator has different normalization to the Wilson case due to a difference in the Z_3 renormalization constant.

up to logarithmic corrections. We can then use this knowledge to ameliorate our lattice results and make better contact with the continuum. It is also well known that the lattice propagator for a free massless boson with an unimproved action is inversely proportional to a trigonometric function of the discrete momentum,

$$D(\hat{q}) = \frac{1}{\frac{4}{a^2}\sum_\mu \sin^2(\frac{\hat{q}_\mu a}{2})}. \tag{9.13}$$

It was suggested, in Ref. [62] and elsewhere, that the correct momentum variable to use when examining the gluon propagator on the lattice, with the Wilson action, is not Eq. (9.6), but[2]

$$q_\mu \equiv \frac{2}{a}\sin\frac{\hat{q}_\mu a}{2}. \tag{9.14}$$

It has been observed that this choice ensures that the propagator takes its asymptotic form at large lattice momenta [62, 73].

The improved action Eq. (5.16) together with the gluon field defined in Eq. (9.21) has the $\mathcal{O}(a^2)$ improved tree-level behaviour [69, 70]

$$D^{-1}(\hat{q}) = \frac{4}{a^2}\sum_\mu \Big\{\sin^2\Big(\frac{\hat{q}_\mu a}{2}\Big) + \frac{1}{3}\sin^4\Big(\frac{\hat{q}_\mu a}{2}\Big)\Big\}. \tag{9.15}$$

Equations (9.13) and (9.15) will be used to obtain the correct momentum variable for each action.

The idea is to eliminate the irrelevant lattice operators and in doing so emphasise the nonperturbative aspects of the propagator. The procedure consist in dividing the propagator by its perturbative form. In this case we expect the propagator to approach as $q^2 \to \infty$, up to logarithmic corrections, a constant. We will see that performing such a step on the gluon propagator significantly improves its ultraviolet behaviour. Hence, all figures are plotted against $q^2 D(q^2)$, unless specified otherwise.

We will define our lattice momentum variables as

$$q_\mu^W \equiv \frac{2}{a}\sin\frac{\hat{q}_\mu a}{2}, \tag{9.16}$$

for the standard Wilson gauge action, and

$$q_\mu^I \equiv \frac{2}{a}\sqrt{\sin^2\Big(\frac{\hat{q}_\mu a}{2}\Big) + \frac{1}{3}\sin^4\Big(\frac{\hat{q}_\mu a}{2}\Big)}, \tag{9.17}$$

for our improved action. A similar approach was used in Ref. [59] for their study of the gluon propagator.

In the continuum the scalar part of the gluon propagator in Landau gauge can be written as:

$$p^2 D(p^2) = \frac{1}{1+\Pi(p^2)} = \frac{D(p^2)}{D^{\text{tree}}(p^2)}, \tag{9.18}$$

where $\Pi(p^2)$ is the scalar vacuum polarization. Asymptotic freedom predicts that $1/[1+\Pi(p^2)] \to 1$ up to logarithmic corrections.

It is argued here that first the lattice version of $D(p^2)/D^{\text{tree}}(p^2)$ experiences asymptotic freedom just as in the continuum and secondly that this quantity will approach

[2]Many authors have q and $\hat{q}$ defined the other way around, but in this context our terminology is more natural.

its continuum form according to the lattice spacing. So on the lattice we expect $D(p^2)/D^{\text{tree}}(p^2) \to 1$ as $p^2 \to \infty$ just as in the continuum, even though the lattice artifacts in both $D(p^2)$ and $D^{\text{tree}}(p^2)$ may themselves be large.

This procedure is called *tree-level correction.* This philosophy motivated the tree–level correction applied in recent studies of the quark propagator [74], and in the work presented in this thesis, especially in Chapter 10.

In figures where $q^2D(q^2)$ is plotted vs. q, the "q" in $q^2D(q^2)$ (plotted on the y-axis) is always the same as the q that is used on the x-axis, where $q = \hat{q}$, q^W or q^I as described in the text.

Let us now see what happens when we tree-level correct to the gluon propagator through the alternative momentum variables derived from the tree–level behaviour of the actions. The effects of tree-level correction may be seen in Fig. 9.5 and Fig. 9.6, where the Wilson propagator has been plotted as a function of q^W ($q^W D(q^W)$ vs. q^W) and the improved propagator as a function of q^I ($q^I D(q^I)$ vs. q^I) for all momenta of the Brillouin zone.

Both Figs. 9.5 and 9.6 are consistent with the study of Ref. [62]. In Figures 9.1 and 9.2 we saw an excellent restoration of rotational symmetry right through the momentum spectrum with improved action. Comparing Figs. 9.5 and 9.6 we clearly see that it is also the case once the gluon propagator has been tree–level corrected. Furthermore, there is a clear discrepancy between the diagonal and Cartesian points in Fig. 9.5 showing a clear sign of rotational symmetry breaking for the standard Wilson action. This compares with Fig. 9.6 where the spread of points is substantially smaller with the improved action. The gluon propagator from lattice 1i was examined versus the lattice momentum q^W, Eq. (9.16), this lead to a propagator which was strongly affected by lattice artifacts especially in the ultraviolet region where a sharp drop was observed. This is clearly a poor choice of momentum variable for this action as expected on the basis of our tree-level correction. The best prescription for optimum results at finite lattice spacing is to use the lattice momentum that is determined from the tree level behaviour of the propagator for the choice of action and gluon field.

9.5 Analysis of Lattice Artifacts on Coarse Lattices

To study the non–perturbative behaviour of the gluon propagator, one divides by its tree–level result known from lattice perturbation theory. Hence, plotted in Figs. 9.7, 9.8 and 9.11 is $q^2D(q^2)$ versus the lattice momentum defined in Eq. (9.14). In the deep ultraviolet region we expect the gluon propagator to approach, up to logarithmic corrections, a constant.

The noisy behaviour of the propagator and lattice artifacts may be better managed by selecting the momentum points that only lie within a cylinder of radius of two spatial momentum units centred about the lattice diagonal. This will enable us to eliminate these large momenta values that are most susceptible to finite lattice artifacts. To select the points appropriately, we calculate the distance $\Delta\hat{q}$ of a momentum vector $\hat{q}$ from the diagonal using

$$\Delta\hat{q} = |\hat{q}| \sin\theta, \tag{9.19}$$

where the angle θ is given by

$$\cos\theta = \frac{\hat{q}\cdot\hat{n}}{|\hat{q}|}, \tag{9.20}$$

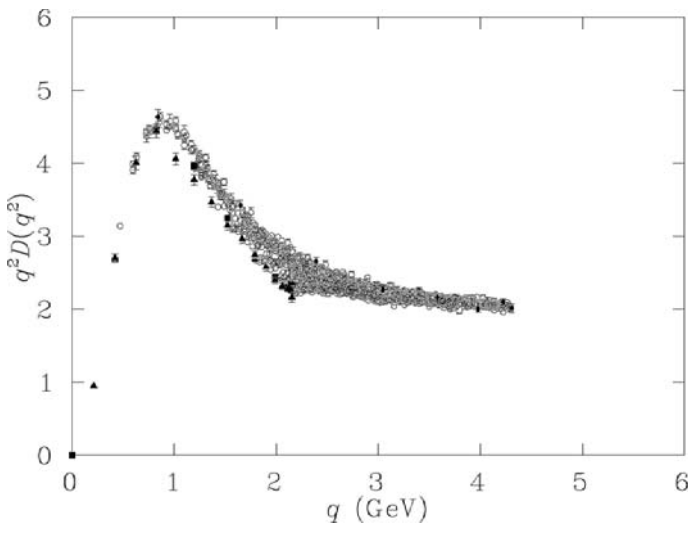


Figure 9.5: Uncut gluon propagator from lattice 1w ($\beta = 5.70, 16^3 \times 32$, Wilson action), plotted as a function of q^W for all momenta. The tree-level correction has greatly reduced discretization errors from those seen in Fig. 9.1.

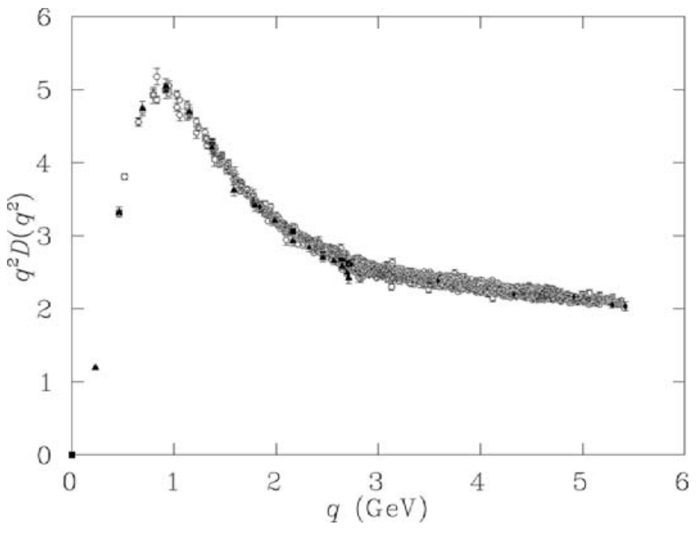


Figure 9.6: Uncut gluon propagator from lattice 1i ($\beta = 4.38, 16^3 \times 32$, improved action), plotted as a function of q^I for all momenta. The combination of improved action and tree-level correction has produced a remarkably clean signal over the entire range of accessible momenta. This figure should be compared with Fig. 9.2, and with Fig. 9.5 for the Wilson action at a similar lattice spacing.

and $\hat{n} = \frac{1}{2}(1,1,1,1)$ is the unit vector along the diagonal. The method for selecting the momentum in a such a way is known as *cylinder cut* [62].

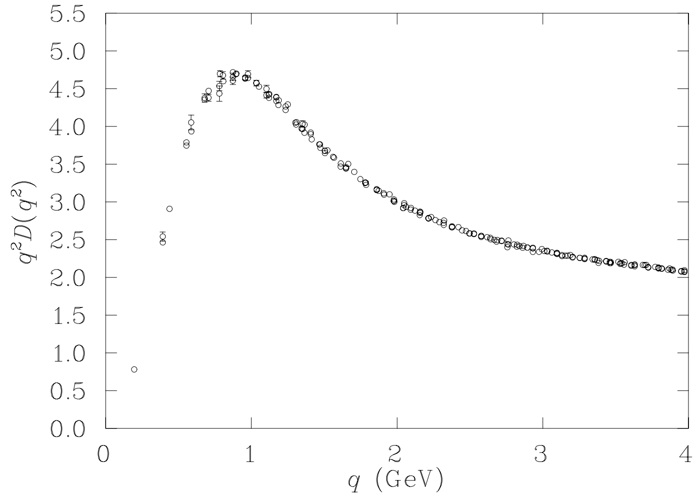


Figure 9.7: Gluon propagator from 75 standard, Wilson configurations, on a $32^3 \times 64$ lattice with spacing $a = 0.10$fm.

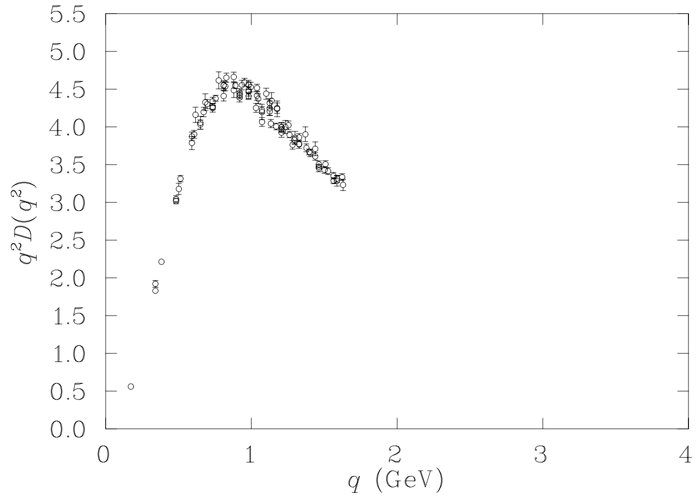


Figure 9.8: Gluon propagator from 75 tree-level improved configurations on a $10^3 \times 20$ lattice with spacing $a = 0.35$fm, and a physical volume of $3.5^3 \times 7$fm^4.

In Ref. [62], the gluon propagator was calculated using the Wilson gauge action, Eq. (2.19) on a $32^3 \times 64$ lattice at $\beta = 6.0$, corresponding to a lattice spacing of 0.1 fm. The resulting gluon propagator is reproduced in Fig. 9.7. From this figure, we can clearly see the asymptotic behaviour of the gluon propagator as well as its infrared behaviour (the main results reported in [62]). A detailed analysis shows the anisotropic finite volume errors are small, however, it is not possible to rule out isotropic finite volume artifacts.

Using the improved action described in Sec. 5.1, the gluon propagator was calculated on a small $10^3 \times 20$ lattice at a coarse spacing of $a = 0.35$ fm. The resulting graph is shown in Fig. 9.8. Comparing Fig. 9.8 with Fig. 9.7, and despite the coarseness of the lattice, we clearly see that the improved coarse lattice reproduces the infrared behaviour of Fig. 9.8. Furthermore, if the volume is increased to $5.6^3 \times 11.2$ fm generated from a $16^3 \times 32$ lattice at the same lattice spacing (lattice 4) we also see a perfect agreement in the infrared region. The improved lattice is illustrated in Fig. 9.11. We could make further comparison between the two actions after tree–level correction has been applied and once the data sets have undergone cylinder cutting. As in Sec. 9.2 and Eq. (9.10), the improved propagator need to be rescaled to make the comparison between the two actions possible. The rescaling is dependent on the lattice spacing, so for each lattice it will be slightly different. So the unimproved propagator has been multiplied by a relative renormalization of 1.09. This factor is deduced by adjusting the vertical scales of the two data sets until they agreed. In Fig. 9.9, we are comparing the gluon propagator coming from the Wilson gauge action at $\beta = 5.70$ (lattice 1w) and lattice 1i at $\beta = 4.38$. Both lattices have roughly the same physical volumes. Apart from the superior performance of the improved propagator, the two actions produces the same results.

We may also compare the results coming from the fine lattice (lattice 6) to those coming from a coarser lattice (lattice 1i) for completeness. Lattice 1i is 60% coarser than the fine lattice. Both sets of data are cylinder cut and each is tree–level corrected according to its action. The relative renormalization has been determined to be $Z_3(\text{improved})/\mathrm{Z}_3(\text{Wilson}) = 1.08$. Once again, it can be seen from Fig. 9.10 that both propagators remain consistent, as we saw above for lattice 4, moreover that the ultraviolet performance of lattice 1i is remarkable. The propagator from Ref. [62] had the momentum half-cut applied, whereas the improved propagator with lattice spacing $a = 0.165(2)$fm is shown for the entire Brillouin zone. The propagator was calculated over the same range of momenta as Ref. [62], despite using a much coarser lattice.

We can therefore conclude that these results largely agree with each other and that the behaviour of $q^2\, D(q^2)$ is not significantly affected by volume changes for the volumes considered.

Juxtaposing the three lattices and plotting $D(q^2)$ instead of $q^2 D(q^2)$ versus the lattice momentum, to allow cross examination of the deep infrared, Fig. 9.12. The points beyond 0.35 GeV are all in very strong agreement with each other giving a very robust results with respect to the volume. Fig. 9.12 also shows that only the very lowest momenta points contain some signs of finite volume effects.

At this point, it is good to compare the effects of making the lattice spacing coarser when using an improved action. Let us consider three different lattices which are getting progressively coarser, Figures 9.13, 9.14 and 9.15. These figures show the uncut gluon propagator after being tree-level corrected, the lattice spacing are $a = 0.27$, 0.35 and 0.41 fm for lattices 5, 2 and 3 respectively. A visual comparison of the three, clearly shows that as the UV cut off has been lowered, the perturbative tail of the propagator, slowly fades off while the infrared remains significantly present. There are no signal of loss of information in the infrared as the lattice become coarser.

Using these three lattices in conjunction to lattice 1i and 1w, we can examine the effect on the propagator as the lattice is getting coarser. The first four lattices are shown in Fig. 9.16. Comparing Figures 9.9 and 9.16 we see that the Wilson $\beta = 5.7$ and improved $\beta = 4.10$ and $\beta = 4.38$ results all agree well, which suggests that these are "fine enough" lattices for the study of the gluon propagator on the lattice. Examining Fig. 9.16, shows that the $\beta = 3.75$ and $\beta = 3.92$ propagators do not quite line up with the others and the UV tail of the 3^{rd} lattice rises slightly as the lattice becomes coarser. It is important to note that the propagator on lattice 3 was calculated with only 100

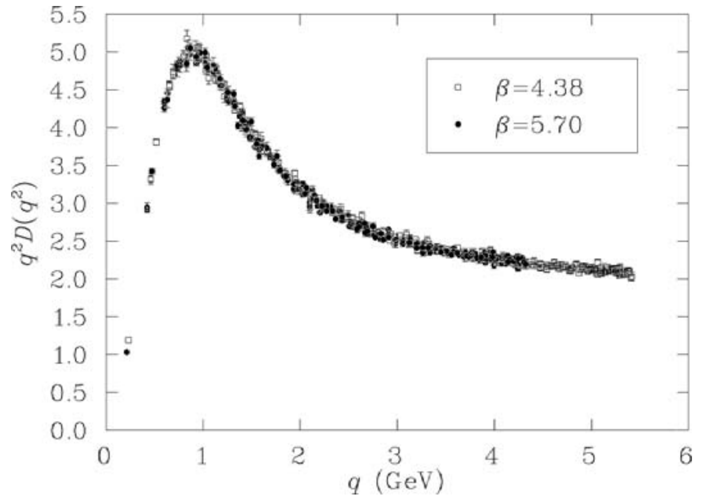


Figure 9.9: Comparison of the gluon propagator from lattices 1w at $\beta = 5.70$ and 1i at $\beta = 4.38$. Data has been cylinder cut and tree-level correction has been applied. We have determined Z_3(improved)/Z$_3$(Wilson) = 1.09 by matching the vertical scales of the data.

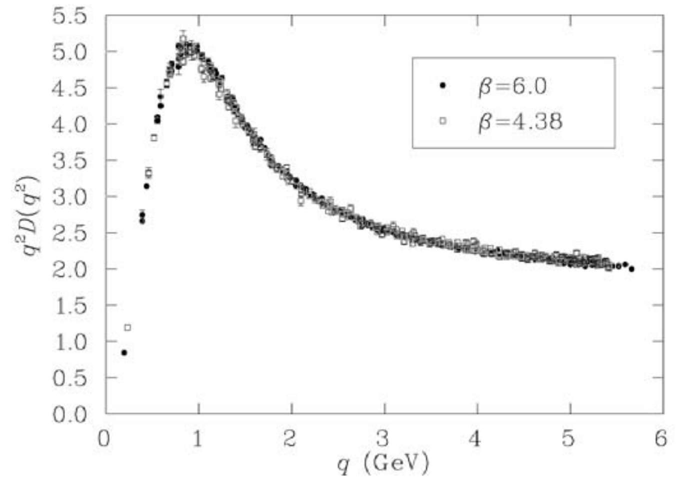


Figure 9.10: Comparison of the gluon propagator from the finest improved lattice (lattice 1i, $\beta = 4.38$) and the finest Wilson lattice (lattice 6, $\beta = 6.0$). Data has been cylinder cut and the appropriate tree-level corrections have been applied. The data from lattice 6 is half-cut whereas lattice 1i displays the full Brillouin zone. We have determined Z_3(improved)/Z$_3$(Wilson) = 1.08 by matching the vertical scales of the data.

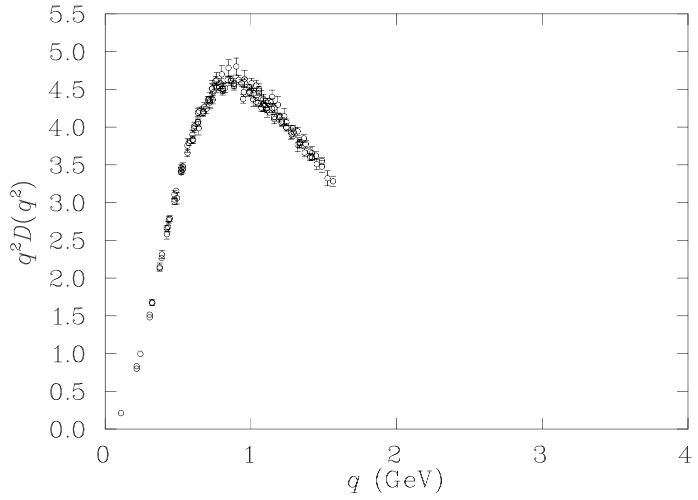


Figure 9.11: Gluon propagator from 75 tree-level improved configurations, on a $16^3 \times 32$ lattice with spacing $a = 0.35$ fm and a physical volume of $5.6^3 \times 11.2 \mathrm{fm}^4$.

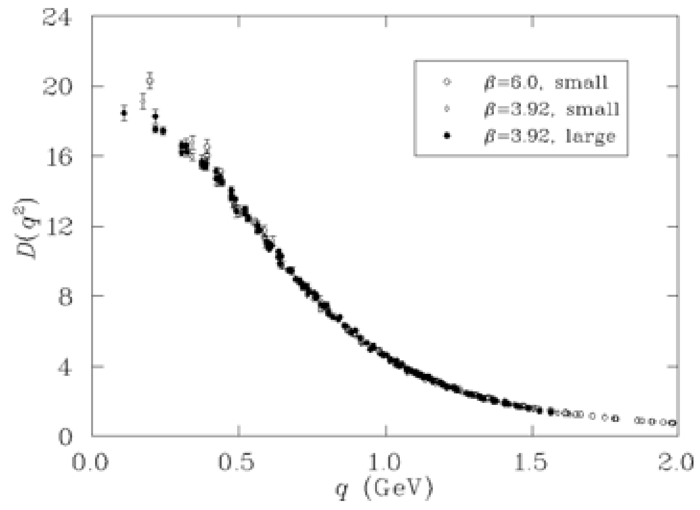


Figure 9.12: Comparison of the gluon propagator on the three different lattices. The volumes are $3.2^3 \times 6.4 \mathrm{fm}^4$, $3.5^3 \times 7.0 \mathrm{fm}^4$, and $5.6^3 \times 11.2 \mathrm{fm}^4$.

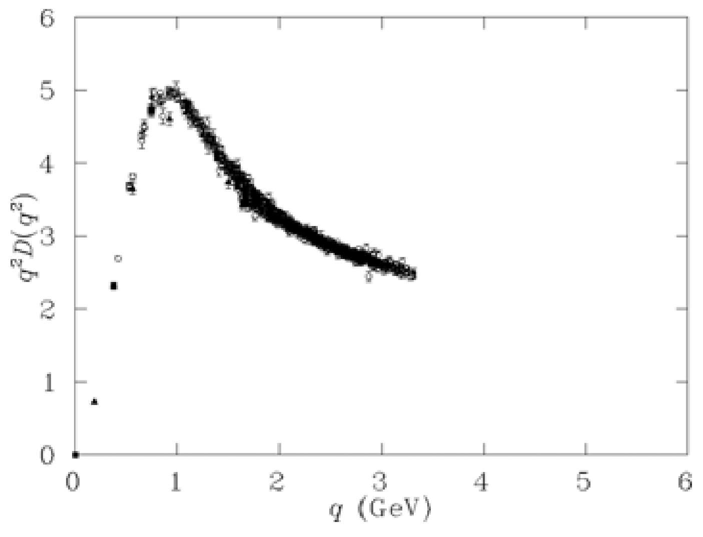


Figure 9.13: Gluon propagator from lattice 5 at $\beta = 4.10$, which has spacing $a \simeq 0.27$fm on $12^3 \times 24$. This has the same physical volume as lattice 3 of Fig. 9.15. The propagator is shown for all momenta (no data cuts) after tree-level correction.

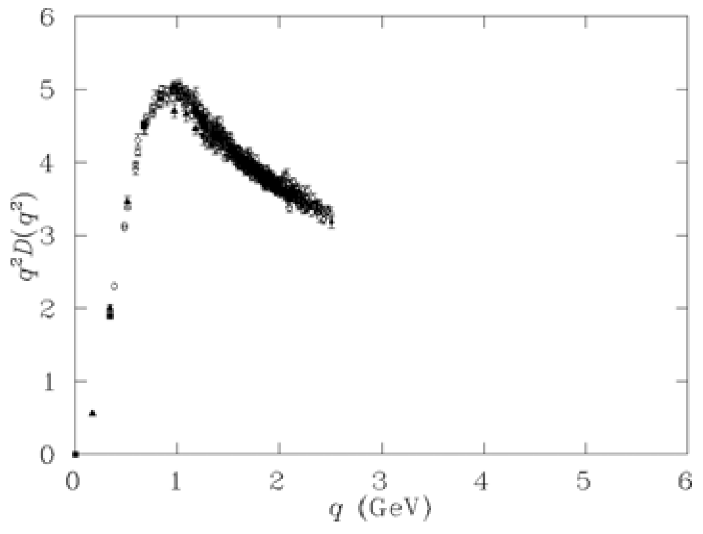


Figure 9.14: Gluon propagator from lattice 2, the smaller lattice at $\beta = 3.92$ which has spacing $a \simeq 0.35$ fm on a $10^3 \times 20$ lattice. Finite volume errors are just detectable as indicated by momenta along the time axis (filled triangles) falling below the rest of the data. Tree-level correction has been used, but no data cuts have been applied.

configurations on a $8^3 \times 16$ lattice (corresponding to a reasonable physical volume of $3.30^3 \times 6.60$). Nevertheless, it is an indication of a loss of scaling. The lattices at $\beta = 3.92$ and $\beta = 3.75$ having $a = 0.35$ and 0.41 fm respectively are too coarse for the tree-level correction to completely correct the entire Brillouin zone. The physics near the cutoff is examined here by these coarse lattice, and it is possible to conclude that such coarse lattices should be half-cut. Nevertheless, the propagators all agree in the infrared.

This gives us great confidence in the exploitation of improved actions on coarse lattice for studying the nonperturbative physics and motivated us for creating a lattice at $a = 0.35$ fm on a very large volume. The results are shown in Fig 9.17, no signs of significant finite volume artifacts are observed when compared with Fig. 9.14 which has the same lattice spacing, but a smaller volume.

We will use this large volumes to extrapolate to the infinite volume limit in conjunctions with more lattices with different spacing, see Sec. 9.6. Nevertheless, at this point in time, we see that at the very lowest momentum points, a decrease in the value of the propagator as the volume increases. This strongly suggests an infrared finite gluon propagator.

To summarize: the infrared behaviour of the gluon propagator calculated with an improved gauge field action remains in good agreement with the one calculated on a finer lattice with the standard Wilson gauge action. The increase of volume left the propagator largely unchanged, in particular the turn over in the infrared region observed in [62] is not a finite volume effect.

The q^{-4} behaviour popular in Dyson-Schwinger studies, and any infrared singularity appears to be unlikely. An infrared finite propagator is most plausible. The gluon propagator would need to drop rapidly for momenta below $\sim$ 350 MeV in order to vanish as suggested by Zwanziger[65] and others.

9.6 Infinite volume behaviour of the Propagator

In this section we are interested in making a contact with the infinite volume behaviour of the gluon propagator. Here, we are going to use the various lattices already mentioned in the previous section plus some other ones for completeness. In Table 9.1, the various lattice with their respective simulation parameter are summarised.

In this analysis the gluon propagator is defined in the same as in Sec. 9.2. Given that the gauge links $U_\mu(x)$ are expressed in terms of the continuum gluon fields as in Eq. (2.1), the dimensionless lattice gluon field $A_\mu(x)$ may be obtained from the gauge links via

$$A_\mu(x + \hat{\mu}/2) = \frac{1}{2igu_0}\{U_\mu(x) - U_\mu^\dagger(x), \}_{\text{traceless}}\,, \tag{9.21}$$

which is accurate to $\mathcal{O}(a^2)$. This is, of course, only one of many possible ways to calculate the gluon field on the lattice. In Eq. (9.21), A_μ is calculated at the midpoint of the link to remove $\mathcal{O}(a)$ terms. The tadpole factor was included to improve the normalization.

The gluon propagator in coordinate space is then calculated using Eq. (9.2) using Eq. (9.21). It is then Fourier transformed into momentum space via Eq. (9.5). To perform the infinite volume linear fit the propagator at $p = 0$ is used, i.e.

$$D(0) = \frac{1}{N_d}\sum_\mu \sum_y D_{\mu\mu}(y). \tag{9.22}$$

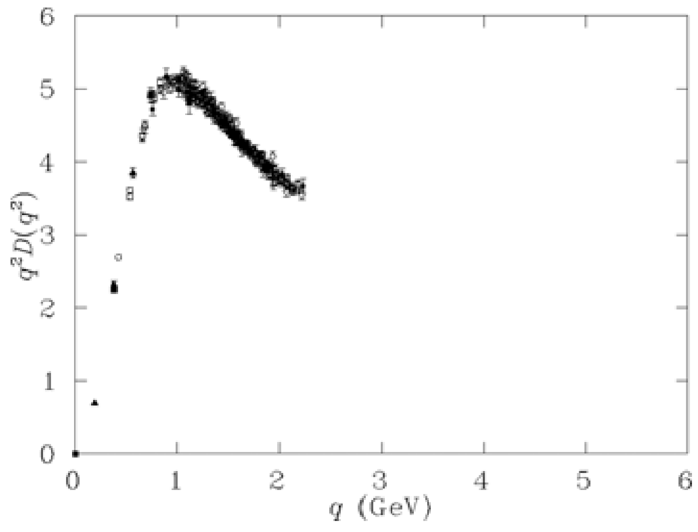


Figure 9.15: Gluon propagator from lattice 3 at $\beta = 3.75$, which has spacing $a \simeq 0.41$ fm on $8^3 \times 16$. The propagator is shown for all momenta (no data cuts) after tree-level correction. This propagator is consistent with that obtained on much finer lattices.

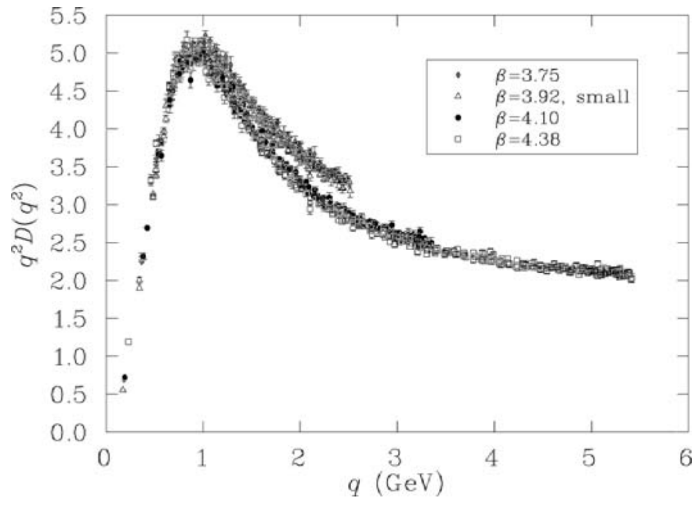


Figure 9.16: Comparison of the gluon propagator from lattices 1i ($\beta = 4.38$), 2 ($\beta = 3.92$, small), 3 ($\beta = 3.75$), and 5 ($\beta = 4.10$), which have a variety of lattice spacings. Data has been cylinder cut and tree-level correction has been applied. Data from the two finest improved lattices (0.165(2) and 0.27 fm) are consistent. A clear violation of scaling is seen in the coarsest two lattices (0.35 and 0.41 fm), where the spacing is too coarse for tree-level correction to completely restore the full Brillouin zone behavior.

To consider the volume dependence of the gluon propagator. We superimpose all of our improved propagators to examine the variation of the propagators as a function of the coupling, see Fig. 9.18. This plot, which shows the cylinder-cut data for the scalar function $D(q^2)$ for each of the improved lattices, provides a strong demonstration of lattice artifacts. In the ultraviolet all the propagators are in perfect agreement but as we move towards lower momentum values (below ~ 500 MeV) we notice that the points are starting to disentangle themselves from one another, showing clear differences in the points distribution. This is a sign of finite volume effect. As the volume increases the low momenta data points drop towards a plateau or flattens off. The grouping of points at about 400 MeV suggest that we are, with the two largest lattices, getting close to the infinite volume limit. The two largest lattices are consistent right through the momentum spectrum, even though the volume difference between the two is significant. This brings confidence in the result. For comparison, the tree-level, perturbative expression $D(q^2) = 1/q^2$ is also shown, suitably normalized.

In Fig. 9.16 we see for our coarse lattices a rise in the UV tail beyond $\sim$ 1 GeV showing a loss of scaling. This is not apparent when we plot the scalar part of the propagator. The multiplication of the propagator by q^2 is required in order to amplify this region of interest and to really pinpoint when the lattice spacing artifacts are present. Doing otherwise may be misleading as far as the effectiveness of improvement in the gluon propagator is concerned. Thus it is always best to plot $q^2D(q^2)$ versus q^2, when the hypercubic UV artifacts are of interest.

We can compare the results of a perturbative thee loop calculation [75] with the lattice data in a specific momentum window. There we can see where the transition from perturbative to non–perturbative physics take place. In Fig. 9.19, the perturbative expression for the free propagator is also shown. The parameters used here are from Ref. [66], where at the renormalization point, $\mu = 5.48$ GeV, the strong coupling constant was found to be $\alpha(\mu) = 0.255$. That was a quenched calculation, so this number should not be compared directly with experiment. The lattice data agree very well with three-loop perturbation theory down to $q \simeq 2.5$ GeV. Below $q \sim 2$ GeV we see that three-loop perturbation theory begins to fail.

9.6.1 Extrapolation to the Continuum Limit

In the previous sections we saw how a singular gluon propagator is highly unlikely and that in the deep infrared the value for the gluon propagator plateaued for the large lattices. In this case, an extrapolation to the continuum limit appears to be reasonable. Here we are going to use the gluon propagator calculated at zero momentum to extrapolate to the continuum limit. The values at zero momentum for the gluon propagator (Statistical errors are given in parentheses) are listed in Table 9.2 for each of the lattices created in this investigation.

The renormalization condition of Eq. (9.9) is enforced at the renormalization point $\mu = 4.0$ GeV, which sets the scale for $D(q^2)$. It is interesting to note that as the volume increases the value of the gluon propagator at zero momentum is reduced, as shown in Fig. 9.20, where I show the renormalized gluon propagator for five lattices. The zero momentum points are included in this plot. Fig. 9.21 illustrates the data with a linear fit in the inverse volume according to

$$D(0) = c\frac{1}{V} + D_\infty(0)\,. \tag{9.23}$$

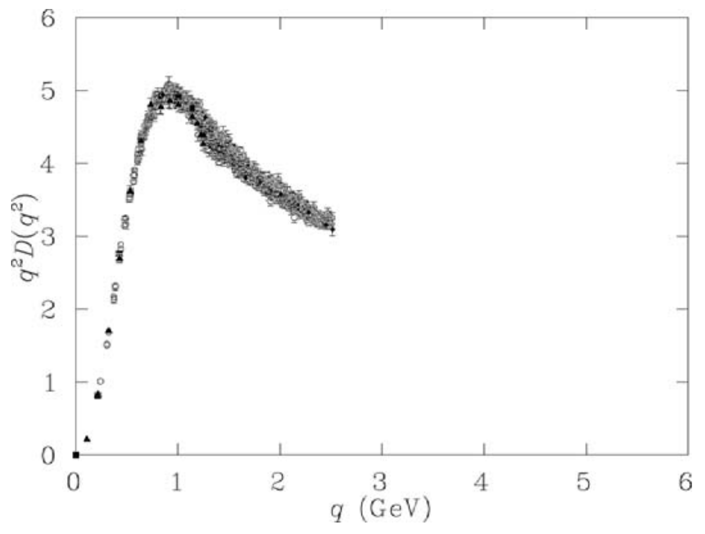


Figure 9.17: Gluon propagator from lattice 4, the larger lattice at $\beta = 3.92$, which has spacing $a \simeq 0.35$ fm on a $16^3 \times 32$ lattice providing the largest physical volume of any in this study. Tree-level correction has been used, but no data cuts have been applied.

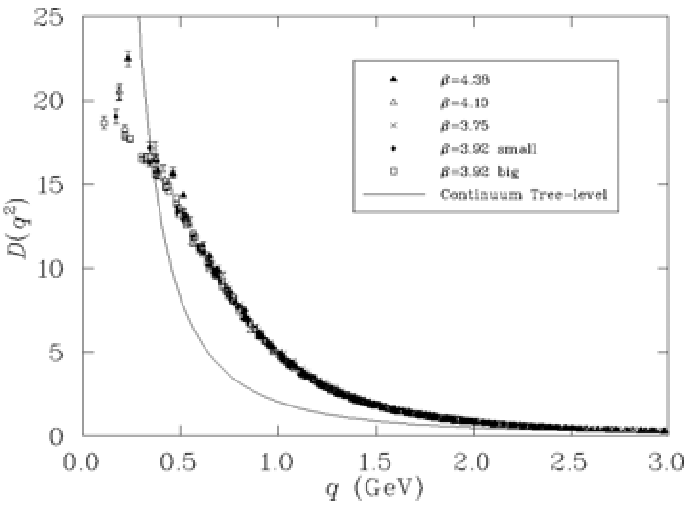


Figure 9.18: Comparison of the gluon propagator generated with an improved action on five different lattices. We find good agreement down to $q \simeq 500$ MeV. At the lowest accessible momenta the data points drop monotonically with increasing volume, but the lowest point (on the largest lattice) shows signs of having converged to its infinite volume value. For comparison with perturbation theory, a plot of the continuum, tree-level gluon propagator (i.e., $1/q^2$ appropriately scaled) has been included.

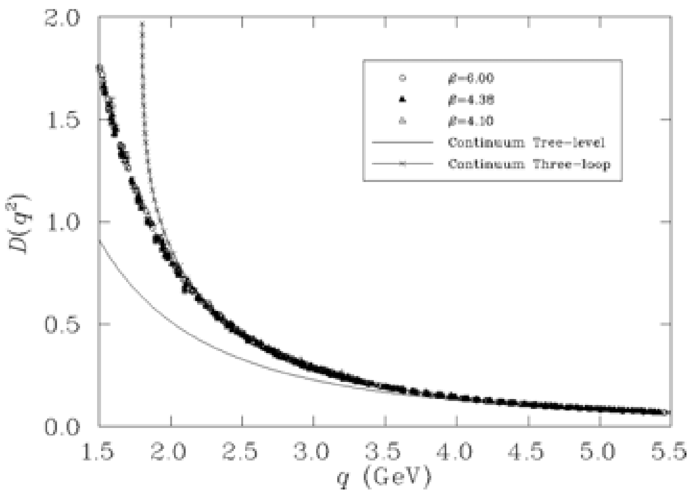


Figure 9.19: Comparison of the lattice gluon propagator with that obtained from perturbation theory, in the ultraviolet to intermediate regime. The continuum expressions are tree-level (i.e., $1/q^2$ appropriately scaled) and the three-loop expression used in Ref. [2].

The results were reported in Ref. [29], there we found that the parameter values $c = 245(22)\,\text{fm}^4\,\text{GeV}^{-2}$ and $D_\infty(0) = 7.95(13)$ GeV $^{-2}$, gave a reasonable fit. Here, $D_\infty(0)$ is the infinite volume limit of the zero-momentum gluon propagator. This brings further evidence for an infrared finite gluon propagator in the infrared.

The results presented in this section can be compared with recent calculations of the gluon propagator in Laplacian gauge [76]. Laplacian gauge is interesting because it is free of any gauge ambiguity. In that gauge, the asymptotic region resembles the one of Landau gauge giving an infrared finite propagator. The results obtained for the propagator in Laplacian gauge is seen to have the same behaviour as the one shown here.

Table 9.2: The value of gluon propagator at zero four-momentum for each of the lattices created in this investigation, in order of increasing volume. The raw (dimensionless) and physical values are given. In obtaining the physical values we have set the renormalization condition $D(\mu^2) = 1/\mu^2$ at $\mu = 4.0$ GeV. An estimate of the uncertainty in the last figure is given in parentheses.

Lattice	Dimensions	β	a fm	D(0)	D(0) (GeV^{-2})	Volume (fm^4)
1i	$16^3 \times 32$	4.38	0.165(2)	32.0 (8)	10.4 (2)	97.2
1w	$16^3 \times 32$	5.70	0.180(1)	24.0 (5)	10.0 (2)	137
5	$12^3 \times 24$	4.10	0.272(2)	10.6 (3)	9.0 (2)	227
3	$8^3 \times 16$	3.75	0.41(1)	4.3 (1)	8.9 (2)	231
2	$10^3 \times 20$	3.92	0.352(3)	5.7 (1)	8.6 (2)	307
4	$16^3 \times 32$	3.92	0.352(3)	5.4 (1)	8.2 (2)	2012

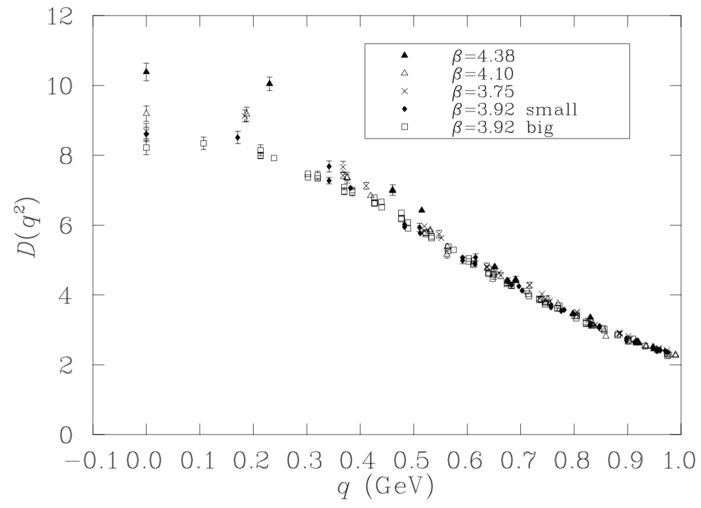


Figure 9.20: The renormalized gluon propagator is shown in the infrared region, including the points at zero four-momentum, from five lattices.

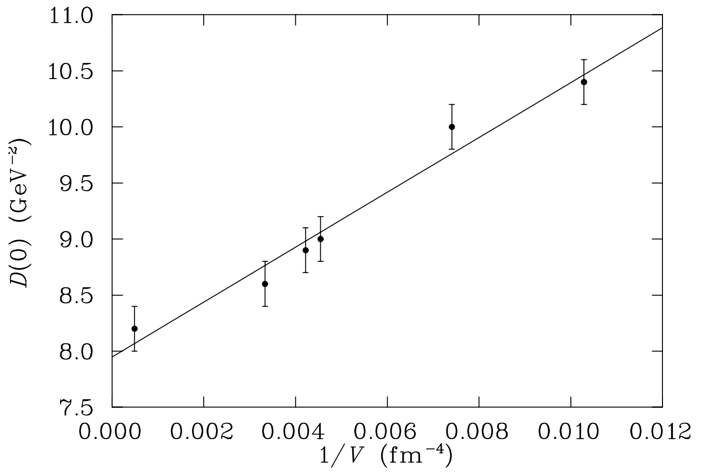


Figure 9.21: Values for the gluon propagator at zero four-momentum, $D(0)$, plotted as a function of the inverse lattice volume. The solid line represents a linear fit to the lattice results. The fit indicates the largest volume results are very close to the infinite volume limit and $D(0) = 7.95(13)$ GeV^{-2} in the infinite volume limit.

9.7 Conclusions

In this chapter the gluon propagator was calculated on various lattices with $\mathcal{O}(a^2)$ improved gauge action. The infrared behaviour of the gluon propagator in Landau gauge was clearly established. It was shown that the tree–level correction is a powerful tool that can be used to suppress the ultraviolet lattice artifacts and to reduce rotational symmetry breaking.

It was also shown, in Sec. 9.5, that $\mathcal{O}(a^2)$ improved action produces superior results compared with a standard Wilson gauge action. One of the main improvements was that one is able to go to much coarser lattices while remaining in perfect agreement.

In Fig. 9.10, we saw the comparison between a gluon propagator calculated on a $32^3 \times 64$ with standard gauge fields at $\beta = 6.00$ and an improved gauge field calculated on a $16^3 \times 32$ lattice at $\beta = 4.38$. The comparison showed that the turn over was consistently reproduced. This was also the case as we made the lattice coarser. This produces a robust result that shows that the turn over of the propagator is infrared finite. The increase of volume for the two largest volumes left the propagator unchanged indicating that finite volume effects are small for these. The tree-level corrected results from the $\beta = 3.92$ ($a = 0.353$ fm) $16^3 \times 32$ lattice with a physical volume of $5.65^3 \times 11.30 = 2038$ fm^4 may be regarded as an excellent estimate of the infinite volume. An extrapolation of $D(0)$ via a linear ansatz inversely proportional to the physical lattice volume provides a reasonable fit. Results from the largest volume lattice sit very close to the infinite volume limit. In a finite ensemble we will never find two configurations which are Gribov copies of each other, since there is an infinite number of gauge orbits. In that sense the issue of Gribov copies is not a relevant here. It is nevertheless interesting to compare with different gauge fixing schemes, such as Laplacian gauge, where Gribov copies are removed by construction.

Chapter 10

Quark Propagators in Lattice QCD Using Improved Actions

One of the special features of QCD is that at short distances the effective coupling constant vanishes. This is known as asymptotic freedom. Hence in that regime, the gluons and the quarks can be well described by perturbation theory.

The quark propagator is really a description of how the quark propagates in the QCD vacuum and is one of the most fundamental building blocks of QCD. By studying the momentum dependent quark mass function in the infrared region, the scalar part of the propagator, we can gain some insights into the mechanism of chiral symmetry breaking. Chiral symmetry is dynamically broken in the QCD vacuum. This gives rise to mass generation in the infrared.

The infrared and asymptotic behaviour of the quark propagator can be studied on the lattice but the level of accuracy in the result will strongly reflect the quality of the discretised quark action. In the deep infrared region, artifacts associated with the finite size of the lattice spacing become small. This is the most interesting region as non-perturbative physics lies here. However, the ultraviolet behaviour at large momentum of the lattice propagator will in general strongly deviate from the correct continuum behaviour. This behaviour will be action dependent. Some interesting progress has been made in improving the ultraviolet behaviour of the propagator.

A method has recently been developed to reduce the ultraviolet noise. The method is referred to as tree–level correction. Tree–level correction was first introduced in the gluon propagator work [29] as reviewed in Chapter 9. The method was then used in quark propagator studies [77] where the correction was performed to subtract the lattice artifacts. This method was later refined to a multiplicative correction in Ref. [78].

In this chapter I present the first study of the overlap quark propagator mass and renormalisation function. The overlap quark action is $\mathcal{O}(a)$–improved and respects exact chiral symmetry on the lattice. To really emphasise the need for the improved action and to show the superiority of the results, just like in Chap. 7 and 9, I compare it to an $\mathcal{O}(a)$ quark action, the Wilson fermion, Sec. 10.4.3. I make a small detour via NNN quark action, Sec. 10.2.2. No numerical work is performed with that quark action, but I present some analytical work which may be used in future works. Also, a systematic method is written down to perform tree–level correction for any quark action, Sec. 10.3.1. Finally I extract the quark condensate, Sec. 10.4.4, and compare the results to latest phenomenological studies.

10.1 Lattice Gauge Action for $SU_c(3)$

The tree–level $\mathcal{O}(a^2)$–improved action is defined as in Sec. 5.1. Fifty configurations have been generated on a $12^3 \times 24$ lattice at $\beta = 4.60$. Configurations are selected after 5000 thermalization sweeps from a cold start, and every 500 sweeps thereafter with a fixed mean link value. Lattice parameters are summarized in Table 10.1.

The gauge field configurations were gauge fixed to the Landau gauge using a Conjugate Gradient Fourier Acceleration [58] algorithm with an accuracy of $\theta \equiv \sum |\partial_\mu A_\mu(x)|^2 < 10^{-12}$. The gauge fixing method was used appropriately to keep the discretization errors to their minimum. The various functionals and methods used for the gauge fixing are described in Chapter 8.

10.2 Fermion Actions on the Lattice

10.2.1 Wilson Fermions Revisited

We saw in Sec. 2.1.3 that the simplest $\mathcal{O}(a)$ fermion action on the lattice with no fermion doubling problem was defined by the Wilson fermion, Eq. (2.25).

The quark propagator can always be derived directly from the quark action. The QCD gauge links can be expressed as $U_\mu(x) = \exp(igaA_\mu(x))$, where gluon field $A_\mu(x)$ is contained in the colour octet $SU_c(3)$ Lie Algebra, i.e., $A_\mu(x) = \sum_{a=1}^{8} t^a A^a_\mu(x)$. These gauge links may be expanded to extract the various n–point functions. For example, at the zeroth order in g the quark propagator will arise, at the order of g the quark–gluon vertex will appear and so on. We perform the expansion of $U_\mu(x)$, grouping the terms in zeroth order in g and then transforming the gluon and quark fields to momentum space using the following momentum prescription over the Brillouin zone, $[-\pi/a, \pi/a]$

$$\psi(x) = \int_{-\frac{\pi}{a}}^{\frac{\pi}{a}} \frac{d^4p}{(2\pi)^4} \tilde{\psi}(p) e^{ip\cdot x}, \text{ and } \overline{\psi}(x) = \int_{-\frac{\pi}{a}}^{\frac{\pi}{a}} \frac{d^4p'}{(2\pi)^4} \tilde{\overline{\psi}}(p') e^{-ip'\cdot x}, \tag{10.1}$$

which implies that

$$\psi(x + a\hat{\mu}) = \int_{-\frac{\pi}{a}}^{\frac{\pi}{a}} \frac{d^4p}{(2\pi)^4} \tilde{\psi}(p) e^{ip\cdot x} e^{ip_\mu a}, \text{ and } \overline{\psi}(x + a\hat{\mu}) = \int_{-\frac{\pi}{a}}^{\frac{\pi}{a}} \frac{d^4p'}{(2\pi)^4} \tilde{\overline{\psi}}(p') e^{-ip'\cdot x} e^{-ip'_\mu a}.$$

Similarly for the gluon field, $A_\mu(x)$, one would take the fourier transform in the same way as for the quark field:

$$A_\mu(x) = \int_{-\frac{\pi}{a}}^{\frac{\pi}{a}} \frac{d^4p''}{(2\pi)^4} \tilde{A}_\mu(p'') e^{ip''\cdot x}. \tag{10.2}$$

The Wilson quark action, at the zeroth order in g, takes the form

$$S_W^{(0)}[U, \overline{\psi}, \psi] \;=\; \sum_x \int_{-\frac{\pi}{a}}^{\frac{\pi}{a}} \frac{d^4p}{(2\pi)^4} \frac{d^4p'}{(2\pi)^4} \tilde{\overline{\psi}}(p') e^{ix\cdot(p-p')}$$

Table 10.1: Parameters of generated lattices.

Action	Volume	N_{Therm}	N_{Samp}	β	a fm	u_0	Physical Volume (fm)
Improved	$12^3 \times 24$	5000	1000	4.60	0.125	0.88888	$1.5^3 \times 3.00$

$$\times \quad \left[(m_q + 4r) - \frac{1}{2} \sum_\mu (r - \gamma_\mu) e^{ip_\mu a} + (r + \gamma_\mu) e^{-ip'_\mu a} \right] \tilde{\psi}(p). \quad (10.3)$$

One of the momentum integrals may be cancelled using the delta function, $\delta(p - p') = \sum_x e^{ix\cdot(p-p')}$. Moreover one may use trivial trigonometric identities to rewrite the exponential factor in Eq. (10.3),

$$\begin{aligned} e^{ip_\mu a} + e^{-ip'_\mu a} &= 2\cos(p_\mu a) = 2\cos\left(\frac{2p_\mu a}{2}\right) = 2\left[1 - 2\sin^2\left(\frac{p_\mu a}{2}\right)\right], \\ e^{-ip'_\mu a} - e^{ip_\mu a} &= -2i\sin(p_\mu a)\,. \end{aligned}$$

We then obtain the following expression

$$\begin{aligned} S_W^{(0)} &= \int_{-\frac{\pi}{a}}^{\frac{\pi}{a}} \frac{d^4p}{(2\pi)^4} \tilde{\bar{\psi}}(p) \left[(m_q + 4r) - \frac{1}{2} \sum_\mu r\left(2 - 4\sin^2\left(\frac{p_\mu a}{2}\right)\right) - 2i\sin(p_\mu a)\gamma_\mu \right] \tilde{\psi}(p) \\ &= \int_{-\frac{\pi}{a}}^{\frac{\pi}{a}} \frac{d^4p}{(2\pi)^4} \tilde{\bar{\psi}}(p) \left[m_q + 2r\sum_\mu \sin^2\left(\frac{p_\mu a}{2}\right) + i\sum_\mu \sin(ap_\mu)\gamma_\mu \right] \tilde{\psi}(p) \\ &= \int_{-\frac{\pi}{a}}^{\frac{\pi}{a}} \frac{d^4p}{(2\pi)^4} \tilde{\bar{\psi}}(p) D_W(p) \tilde{\psi}(p)\,. \end{aligned} \quad (10.4)$$

Where $D_W(p)$ is the Wilson fermion operator in momentum space. The quark propagator in momentum space is given by taking the inverse of the Wilson fermion operator in momentum space,

$$\begin{aligned} S^{(0)}(p) = [D_W(p)]^{-1} &= \frac{1}{m_q + 2r\sum_\mu \sin^2(\frac{p_\mu a}{2}) + i\sum_\mu \sin(p_\mu a)\gamma_\mu} \\ &= \frac{Z^{(0)}(p)}{ia\not{k} + M^{(0)}(p)}. \end{aligned} \quad (10.5)$$

Here $M^{(0)}(p)$ and $Z^{(0)}(p)$ are the tree–level mass and renormalization functions in momentum space. For Wilson fermions these are given by

$$M^{(0)}(p) = m_q + 2r\sum_\mu \sin^2(\frac{p_\mu a}{2}) \equiv M^{(ana)}(p), \qquad \text{and} \qquad Z^{(0)}(p) = 1. \quad (10.6)$$

The lattice momentum $k_\mu = (1/a)\sin(p_\mu a)$ is constructed from discrete momentum values for a lattice of size $N_i^3 \times N_t$. The boundary condition for the dicrete momentum is periodic in the spatial direction while being anti–periodic in the temporal direction. Then for $n_i = 1, .., N_i$ and $n_t = 1, .., N_t$ the discrete momentum values are

$$p_i = \frac{2\pi}{N_i a}\left(n_i - \frac{N_i}{2}\right), \qquad \text{and} \qquad p_t = \frac{2\pi}{N_t a}\left(n_t - \frac{1}{2} - \frac{N_t}{2}\right). \quad (10.7)$$

These values are distributed over the Brillouin zone. Eq.(10.5) is the analytical form for the quark propagator at tree–level.

10.2.2 Next–to–Nearest Neighbour Action

The Next–to–Nearest Neighbour fermion action is a specific case of the general class of the $D234$ actions [80] in which the improvement terms are calibrated to remove the second order chiral symmetry breaking Wilson term. This action also known as $D\chi 34$ [80] breaks chiral symmetry as well, but at the fourth order. This $\mathcal{O}(a^2)$–improved fermion action [81, 82] is defined as,

$$\begin{aligned}
S_I[U,\overline{\psi},\psi] &= \sum_x \Big[(m_q+4r)\overline{\psi}(x)\psi(x) \\
&- \frac{2}{3}\sum_\mu \overline{\psi}(x)(r-\gamma_\mu)\frac{U_\mu(x)}{u_0}\psi(x+a\hat{\mu}) + \overline{\psi}(x+a\hat{\mu})(r+\gamma_\mu)\frac{U_\mu^\dagger(x)}{u_0}\psi(x) \\
&+ \overline{\psi}(x)(-\frac{r}{4}+\frac{1}{8}\gamma_\mu)\frac{U_\mu(x)}{u_0}\frac{U_\mu(x+a\hat{\mu})}{u_0}\psi(x+2a\hat{\mu}) \\
&+ \overline{\psi}(x+2a\hat{\mu})(-\frac{r}{4}-\frac{1}{8}\gamma_\mu)\frac{U_\mu^\dagger(x+a\hat{\mu})}{u_0}\frac{U_\mu^\dagger(x)}{u_0}\psi(x)\Big] \\
&= \sum_{xy}\overline{\psi}(x)K_I(x,y)\psi(y), \qquad (10.8)
\end{aligned}$$

with $K_I(x,y)$ the fermion operator:

$$\begin{aligned}
K_I(x,y) &= (m+r)\delta_{xy} - \qquad (10.9) \\
&\frac{2}{3}\sum_\mu (r-\gamma_\mu)\frac{U_\mu(x)}{u_0}\delta_{y,x+a\hat{\mu}} + (r+\gamma_\mu)\frac{U_\mu^\dagger(x-a\hat{\mu})}{u_0}\delta_{y,x-a\hat{\mu}} + \\
&\left(-\frac{r}{4}-\frac{1}{8}\gamma_\mu\right)\frac{U_\mu(x)U_\mu(x+a\hat{\mu})}{u_0^2}\delta_{y,x+2a\hat{\mu}} + \\
&\left(-\frac{r}{4}+\frac{1}{8}\gamma_\mu\right)\frac{U_\mu^\dagger(x-a\hat{\mu})U_\mu^\dagger(x-2a\hat{\mu})}{u_0^2}\delta_{y,x-2a\hat{\mu}}.
\end{aligned}$$

The tree–level expression for the quark propagator in momentum space for the NNN quark action is given by expending the gauge links, $U_\mu(x)$, and only considering the terms involving the zeroth order in the coupling g and then transforming to momentum space. Hence, this quark action takes the form at tree–level of

$$\begin{aligned}
S_W^{(0)}[U,\overline{\psi},\psi] &= \sum_x \int_{-\frac{\pi}{a}}^{\frac{\pi}{a}} \frac{d^4p}{(2\pi)^4}\frac{d^4p'}{(2\pi)^4}\tilde{\overline{\psi}}(p')e^{ix\cdot(p-p')}\Bigg[(m_q+4r) - \frac{2}{3}\sum_\mu (r-\gamma_\mu)\frac{e^{ip_\mu a}}{u_0} + \\
&(r+\gamma_\mu)\frac{e^{-ip'_\mu a}}{u_0} + \left(-\frac{r}{4}+\frac{1}{8}\gamma_\mu\right)\frac{e^{i2p_\mu a}}{u_0^2} + \left(-\frac{r}{4}-\frac{1}{8}\gamma_\mu\right)\frac{e^{-i2p'_\mu a}}{u_0^2}\Bigg]\tilde{\psi}(p) \\
&= \int_{-\frac{\pi}{a}}^{\frac{\pi}{a}} \frac{d^4p}{(2\pi)^4}\tilde{\overline{\psi}}(p)\Bigg[(m_q+4r) - \frac{2}{3u_0}\sum_\mu r\left[\left(2-4\sin^2\left(\frac{p_\mu a}{2}\right)\right) - \frac{1}{4u_0}\left(2-4\sin^2(p_\mu a)\right)\right] \\
&\qquad - i\gamma_\mu\left[2\sin(p_\mu a) - \frac{1}{4u_0}\sin(2p_\mu a)\right]\Bigg]\tilde{\psi}(p) \\
&= \int_{-\frac{\pi}{a}}^{\frac{\pi}{a}} \frac{d^4p}{(2\pi)^4}\tilde{\overline{\psi}}(p)\left[i\sum_\mu f_\mu(p)\gamma_\mu + M_I^{(0)}(p)\right]\tilde{\psi}(p) = \int_{-\frac{\pi}{a}}^{\frac{\pi}{a}} \frac{d^4p}{(2\pi)^4}\tilde{\overline{\psi}}(p)K_I(p)\tilde{\psi}(p).
\end{aligned}$$

The quark propagator is then defined as:

$$S_I^{(0)}(p) = [K_I(p)]^{-1} = \frac{Z_I^{(0)}(p)}{i\sum_\mu f_\mu(p)\gamma_\mu + M_I^{(0)}(p)}. \tag{10.10}$$

Where the momentum dependent function $f_\mu(p)$ is given by

$$f_\mu(p) = \frac{4}{3u_0}\left[ak_\mu - \frac{1}{8u_0}\sin(2p_\mu a)\right], \tag{10.11}$$

and the tree–level mass function $M_I^{(0)}(p)$ is given by

$$M_I^{(0)}(p) = m_q + r\left[4 - \frac{16}{3u_0}\left(1 - \frac{1}{4u_0}\right)\right] + \frac{2a^2 r}{3u_0}\left[\tilde{k}^2 - \frac{k^2}{u_0}\right]. \tag{10.12}$$

Here the lattice momentum k_μ and $\tilde{k}_\mu$ are defined as:

$$k_\mu = \frac{1}{a}\sin(p_\mu a) \text{ and } \tilde{k}_\mu = \frac{2}{a}\sin\left(\frac{p_\mu a}{2}\right).$$

10.2.3 Overlap Fermions

The overlap fermion [84] realizes exact chiral symmetry on the lattice, and can be employed to derive chiral symmetry relations and the anomaly at finite lattice spacing. This action produces propagators that are $\mathcal{O}(a)$ improved. Neuberger's polar decomposition approximation to the overlap Dirac operator for the auxiliary Hamiltonian, H, has the following form for the massive case [85]

$$D(m_0) = 1 + \frac{m_0 a}{2} + \left(1 - \frac{m_0 a}{2}\right)\gamma_5\epsilon(H), \tag{10.13}$$

where $\epsilon(H) = H/\sqrt{H^2}$ is the matrix sign function of H which we take to be the Hermitian Wilson–Dirac operator, i.e. $H = \gamma_5 D_w$. Here D_w is the Wilson fermion operator:

$$\begin{aligned} D_w(x,y) &= (m_q + 4r)\delta_{x,y} - \\ \frac{1}{2}\sum_\mu &(r-\gamma_\mu)U_\mu(x)\delta_{y,x+a\hat{\mu}} + (r+\gamma_\mu)U_\mu^\dagger(x-a\hat{\mu})\delta_{y,x-a\hat{\mu}}, \end{aligned} \tag{10.14}$$

except with a negative mass parameter which corresponds to $\kappa_c < \kappa < 0.25$. The massless operator $D(m_0 = 0)$ is shown [86] to satisfy the Ginsparg–Wilson relation [87]:

$$\{\gamma_5, D(0)\} = D(0)\gamma_5 D(0), \tag{10.15}$$

as a consequence the fermion propagator anti–commutes with γ_5 and chiral symmetry is exactly preserved. The bare mass parameter m_0 is proportional to the quark mass without any additive constant [88]. The sign function is approximated using the optimal rational approximation [89] with a ratio of polynomials of degree 12 in the Remez algorithm. The matrix, $V = \gamma_5\epsilon(H)$, is unitary [88]. It was found that for a matrix equation of the type $Vx = b$, then $\left|x^\dagger x - b^\dagger b\right| \sim 10^{-9}$. Since V is unitary one can exploit the identity: $(1+V^\dagger)(1+V) = 2 + V^\dagger + V$, in order to use the conjugate gradient algorithm on the hermitian matrix $V^\dagger + V$ instead of $V^\dagger V$ which has a higher condition number. Furthermore, since $[V + V^\dagger, \gamma_5] = 0$, one can use a chiral source, i.e. $\gamma_5 b = \pm b$ to save one matrix multiplication [89] per iteration.

The massive overlap Dirac operator may be rewritten as:

$$D(\mu) \;=\; \frac{1}{2}\left[1 + \mu + (1-\mu)\frac{D_w}{\sqrt{D_w^\dagger D_w}}\right],$$

with $0 \leq \mu \leq 1$ describing fermions with a +ve mass from 0 to ∞. For Small μ, μ is proportional to the fermion mass. In simulation, μ, is the tuneable parameter in analogy with the κ value for the Wilson fermion. In the overlap formalism, the hopping parameter is given by

$$\kappa = \frac{1}{-2m + 8r} \;\text{ and }\; \mathrm{m} \equiv -\frac{1}{2}\left(\frac{1}{\kappa} - \frac{1}{\kappa_{\mathrm{cr}}}\right).$$

To describe a single massless Dirac Fermion for $D(0)$ we must have

$$m_c \leq m \leq 2\,,$$

for a critical Wilson Dirac mass, m_c, at tree level.

It may be shown [90], that the physical overlap propagator is given by

$$\tilde{D}^{-1}(\mu) = (1-\mu)^{-1}[D^{-1}(\mu) - 1],$$

which is related to the continuum propagator (the inverse of the matrix operator) by

$$D_c^{-1}(m_q) = \mathcal{Z}_\psi \tilde{D}^{-1}(\mu), \;\text{ with }\; \mathrm{m_q} = \mathcal{Z}_{\mathrm{m}}^{-1}\mu. \tag{10.16}$$

The constants $\mathcal{Z}_\psi$ and $\mathcal{Z}_m$ are the mass and wave function renormalization constants. At tree–level it is found [91] that:

$$\mathcal{Z}_\psi^{(0)} = \mathcal{Z}_m^{-1} = 2m = \frac{1}{\kappa_{cr}^{(0)}} - \frac{1}{\kappa}, \tag{10.17}$$

where the $\kappa_{cr}^{(0)}$ takes the same value as in the Wilson fermion case at tree–level, i.e. $\kappa_{cr}^{(0)} = 1/8$. However, in the interacting case κ_{cr} is defined as the value of κ at which the pion mass vanishes. Therefore, at tree–level, the free propagator becomes:

$$\left[D_c^{(0)}(m_q)\right]^{-1} = \left[\mathcal{Z}_\psi^{(0)}\tilde{D}^{(0)}(\mu)\right]^{-1},$$

and when the interactions are turned on we have:

$$\left[D_c(m_q)\right]^{-1} = \left[\mathcal{Z}_\psi \tilde{D}(\mu)\right]^{-1}.$$

10.3 The Quark Propagator on the Lattice

In a Lorentz covariant gauge, the most general structure for the quark propagator is

$$S(p) = \frac{1}{i\gamma\cdot p A(p^2;\alpha^2) + B(p^2;\alpha^2)} = \frac{Z(p^2;\alpha^2)}{i\gamma\cdot p + M(p^2;\alpha^2)}. \tag{10.18}$$

Here, the parameter α represents the renormalization point. The functions $A(p^2;\alpha^2)$ and $B(p^2;\alpha^2)$ carries all the effects of vector and scalar quark dressing induced by the quark interactions with its own gluon field. Using these functions we may extract the functions of interest, namely the $Z(p^2;\alpha^2)$ and $M(p^2;\alpha^2)$ for various quark actions.

On the lattice we expect the bare quark propagators, in momentum space, to have a similar form as in the continuum [77, 78, 79]. Hence, the dimensionless inverse lattice bare quark propagator takes the general form of:

$$S^{-1}(p) \equiv i\left(\sum_{\mu} C_{\mu}(p)\gamma_{\mu}\right) + B(p), \tag{10.19}$$

with $\mathcal{C}^2(p) = \sum_{\mu}(\mathcal{C}_{\mu}(p))^2$. Then the quark propagator may be written as

$$S(p) = \frac{-i\left(\sum_{\mu} C_{\mu}(p)\gamma_{\mu}\right) + B(p)}{C^2(p) + B^2(p)} \equiv -i\left(\sum_{\mu} \mathcal{C}_{\mu}(p)\gamma_{\mu}\right) + \mathcal{B}(p). \tag{10.20}$$

Taking the trace we obtain:

$$\mathcal{C}_{\mu}(p) = \frac{i}{4N_c}\mathrm{Tr}[\gamma_{\mu}S(p)], \qquad \text{and} \qquad \mathcal{B}(p) = \frac{1}{4N_c}\mathrm{Tr}[S(p)], \tag{10.21}$$

which is used to construct the $C_{\mu}(p)$ and $B(p)$

$$C_{\mu}(p) = \frac{\mathcal{C}_{\mu}(p)}{\mathcal{C}^2(p) + \mathcal{B}^2(p)}, \tag{10.22}$$

$$B(p) = \frac{\mathcal{B}(p)}{\mathcal{C}^2(p) + \mathcal{B}^2(p)}. \tag{10.23}$$

At tree–level, that is when all the gauge links are set to the identity, the inverse lattice quark propagator takes the same form as Eq.(10.19), so we have

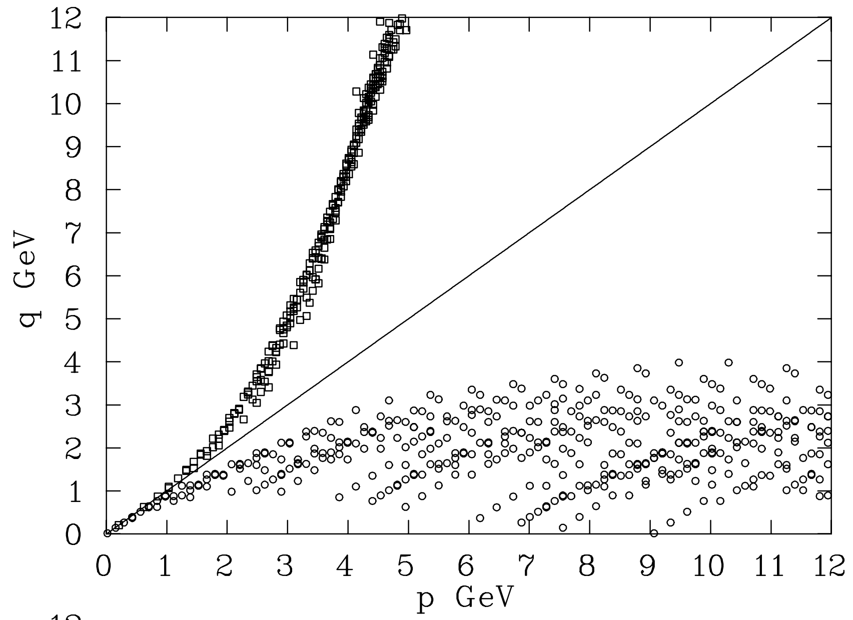
$$(S^{(0)}(p))^{-1} \equiv i\left(\sum_{\mu} C^{(0)}_{\mu}(p)\gamma_{\mu}\right) + B^{(0)}(p) = \frac{i\left(\sum_{\mu} \mathcal{C}^{(0)}_{\mu}(p)\gamma_{\mu}\right) + \mathcal{B}^{(0)}(p)}{(\mathcal{C}^{(0)}(p))^2 + (\mathcal{B}^{(0)}(p))^2}. \tag{10.24}$$

The analytical form of $C^{(0)}_{\mu}(p)$ and $B^{(0)}(p)$ have the same form as Eq.(10.22) and Eq.(10.23) respectively.

It is then possible to extract the lattice momentum directly from numerical simulations by calculating $C^{(0)}_{\mu}(p)$. At tree–level we identify the function $A^{(0)}(p)$ to be 1 for all p. Hence the lattice momentum q_{μ} is given by

$$q_{\mu} \equiv C^{(0)}_{\mu}(p) = \frac{\mathcal{C}^{(0)}_{\mu}(p)}{(\mathcal{C}^{(0)}(p))^2 + (\mathcal{B}^{(0)}(p))^2}. \tag{10.25}$$

We know that the momentum extracted from the lattice, Eq. (10.25), at low momenta should be equal to the discrete momentum, Eq. (10.7). Hence, this can be verified by plotting one versus the other. In Fig. 10.1 I show the relationship between the two definitions over a wide range of the momentum spectrum, the data is uncut. We can see that at low momentum the discrete momentum is equal to the lattice momentum. In Fig. 10.1 I also show the two momentum plotted versus each other but this time



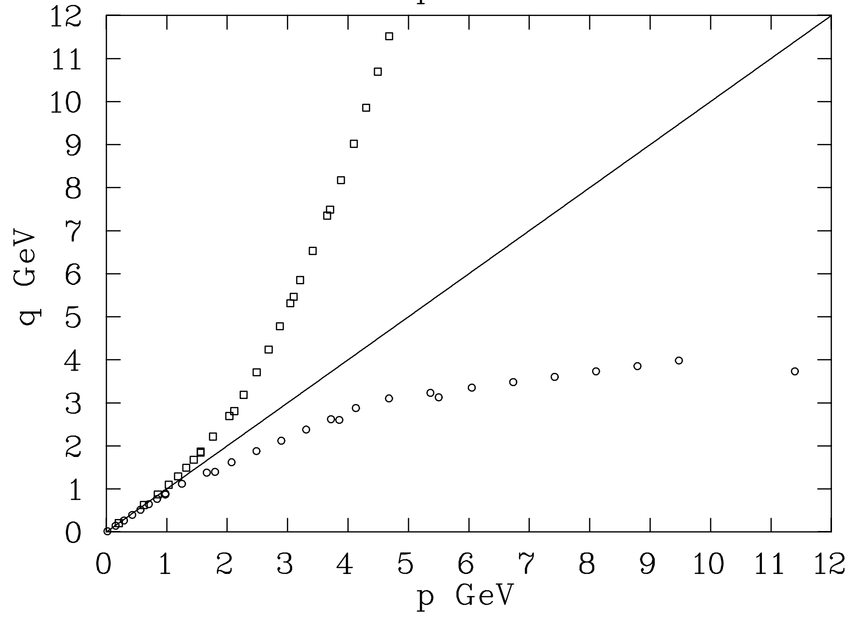


Figure 10.1: The lattice momentum q versus the discrete momentum p, both in GeV. The points going up are for the overlap fermion ($\Box$) and those heading down ($\circ$) with a sinusoidal shape are for the Wilson fermion. Top graph is for the full data and the bottom graph is for the cylinder cut data.

with a cylinder cut applied onto the data. The resulting graph is cleaner and gives a much better view on the relationship. Using these two figures, Fig. 10.1, it is therefore possible to conclude that we have extracted the momentum correctly from the lattice for the overlap fermion. From these figures we can also observe how the two momentum definition diverge from one another. It therefore becomes a reasonable question to ask which definition of the momentum should one plot the functions $M(q^2)$ and $Z^{(\mathrm{R})}(q^2)$ against. This question should be the subject of a short discussion in the next section where the numerical results are presented.

In Fig. 10.2 I show the mass function for the Wilson fermion. It is possible to see from these two figures, Fig. 10.2 and 10.3, the irrelevant operators impose some

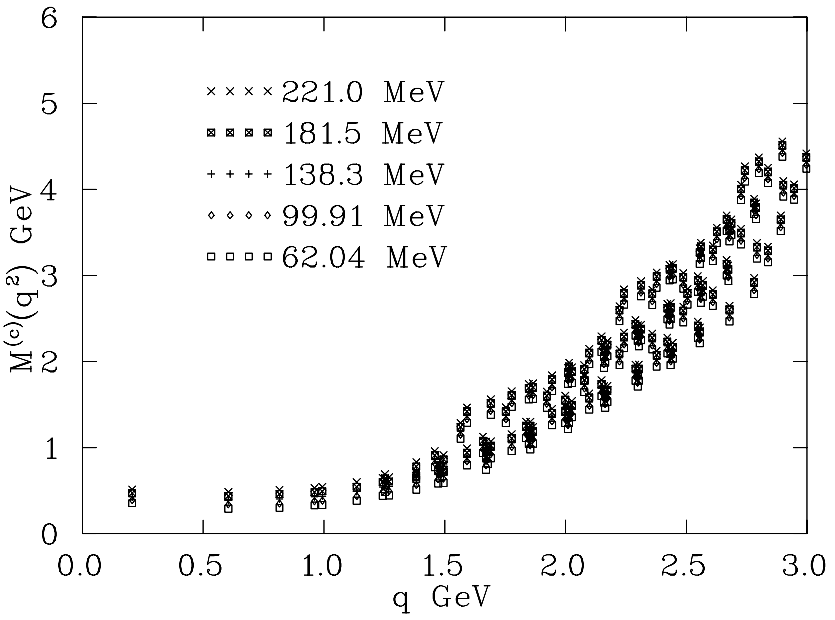


Figure 10.2: The uncorrected mass function $M(q^2)$ for the Wilson fermions.

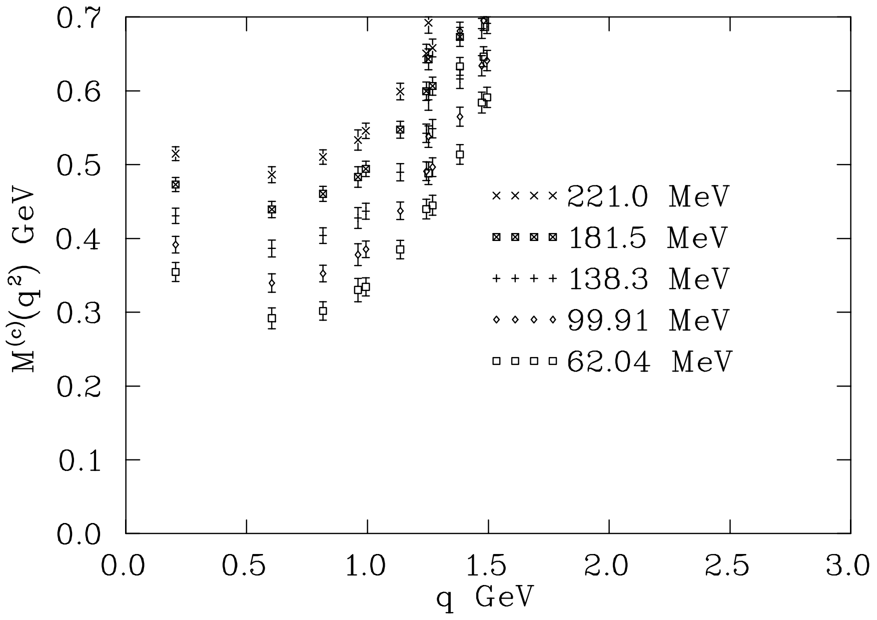


Figure 10.3: The uncorrected mass function $M(q^2)$ for the Wilson fermions, on a smaller scale.

strong divergence from the current quark mass in the ultraviolet region. Moreover, past 500 MeV the mass function becomes unreliable. In order to make it possible to infer anything from these fermions it is therefore necessary to perform some corrections.

10.3.1 Corrected Mass and Renormalization Functions

The dimensionless inverse lattice bare quark propagator takes the form

$$S^{-1}(p) \equiv ia\not{q}A(p) + B(p) = [Z^{\mathrm{L}}(p)]^{-1}\left(ia\not{q} + M^{\mathrm{L}}(p)\right). \tag{10.26}$$

The momentum function $M^{\rm L}(p) = B(p)/A(p)$ is the lattice mass function and $Z^{\rm L}(p) = 1/A(p)$ the lattice renormalization function. The functions $A(p)$ and $B(p)$ may be written as:

$$A(p) = \frac{\mathcal{A}(p)}{\mathcal{A}(p)q^2 + \mathcal{B}(p)}, \qquad \text{and} \qquad B(p) = \frac{\mathcal{B}(p)}{\mathcal{A}(p)q^2 + \mathcal{B}(p)}. \tag{10.27}$$

Hence we can write an equivalent definition for the quark propagator as

$$S(p) \equiv -ia\not{q}\mathcal{A}(p) + \mathcal{B}(p). \tag{10.28}$$

The lattice momentum, q_μ, is given by Eq.(10.25). Extracting the functions $\mathcal{A}(p)$ and $\mathcal{B}(p)$ is simply done by taking the trace of the $S(p)$:

$$\mathcal{A}(p) = \frac{i}{4N_c a q^2}\mathrm{Tr}[\not{q}S(p)], \qquad \text{and} \qquad \mathcal{B}(p) = \frac{1}{4N_c}\mathrm{Tr}[S(p)]. \tag{10.29}$$

It is then possible to use Eq.(10.29) to construct Eq.(10.27).

The tree–level correction is performed on each of the $A(p)$ and $B(p)$. The correction is not affected by the lattice momenta chosen because it is divided out during the correction, as shown in the previous section. The tree–level multiplicative correction is given by taking the ratio of the lattice function calculated over the ensemble average and the function calculated when the links are set to the identity times the expected form obtained from the continuum limit:

$$A^{(c)}(p) = \frac{A(p)}{A^{(0)}(p)}1, \qquad \text{and} \qquad B^{(c)}(p) = \frac{B(p)}{B^{(0)}(p)}m_q. \tag{10.30}$$

Where $A^{(0)}(p)$ and $B^{(0)}(p)$ are the tree–level corrected functions, obtained when all the gauge links $U_\mu(x)$ are set to the identity. The quark mass is the bare lattice quark mass. The deviation of $A(p)$ from $A^{(0)}(p)$ is a direct measure of the gluonic interaction on the quarks. The tree–level corrected mass function $M^{(c)}(p)$ and the renormalization function $Z^{(c)}(p)$ can then be constructed using Eq.(10.30):

$$M^{(c)}(p) = \left(\frac{B^{(c)}(p)}{A^{(c)}(p)}\right), \qquad \text{and} \qquad Z^{(c)}(p) = \frac{1}{A^{(c)}(p)}. \tag{10.31}$$

In the case of the Wilson Fermions it is certainly true that the lattice momentum is given by $k_\mu = (1/a)\sin(a\,p_\mu)$, then $C^{(0)}_\mu(p)$ is just equal to k_μ. But in the case of a more elaborate quark action such as the overlap fermions and the $D234$ fermions, an analytical expression at tree–level is cumbersome to extract. Hence the lattice momenta definition becomes more obscure and diverges from $k_\mu = (1/a)\sin(a\,p_\mu)$. It is then more appropriate to extract the momentum numerically from Eq.(10.24) by just setting all the links to the identity (i.e. $U_\mu(x) = I$) according to Eq.(10.25).

10.4 Numerical Results

The quark propagator has been calculated using 2 different fermion actions on the same lattice. In both cases I have used the same improved glue defined in Section 10.1, see Table 10.1. For the Wilson fermion we have used five different masses, Table 10.2. starting from about 220 MeV to about 62 MeV, these are roughly equally spaced. One

Table 10.2: Summary of the lattice parameters for the quark propagators. $N_{[U]}$ is the number of gauge field configurations considered in the simulation. The second kappa was selected to match the strange quark mass. The corresponding masses for the $12^3 \times 24$ are $\{221, 181.5, 138.3, 99.91, 62.04\}$ MeV.

Quark Action	β	$N_{[U]}$	u_0	κ	κ_{cr}
Wilson	4.60	100	0.88888	0.1337,0.1346,0.1356,0.1365,0.1374	0.1389
Wilson Tree	4.60	1	0.88888	0.1207,0.1215,0.1223,0.1230,0.1237	0.1250

of them was selected to match the strange quark mass. For the overlap ten different masses were used, see Table 10.3.

The standard procedure that was adopted for the overlap, was to extract the momentum q directly from the lattice via Eq. (10.25). Using this definition of the momentum I then moved on to calculate the functions $M(q^2)$ and $Z^{(\mathrm{R})}(q^2)$. The overall renormalization of the lattice propagator tells us that for the interacting case it should be multiplied by $\mathcal{Z}_\psi$ to get the continuum propagator. The tree–level lattice propagator is used to extract the momentum q_μ with an overall renormalization of $\mathcal{Z}_\psi^{(0)}$. In the case of the Wilson fermions, the tree–level propagators were calculated with an exact correspondence of the κ values from the interacting to the free case with κ_{cr} set to its tree value.

The Wilson fermion required tree–level correction, unlike the overlap fermion which did not require tree–level correction to extract a well behaved $M(q^2)$ and $Z^{(\mathrm{R})}(q^2)$ function. The tree–level form of $A^{(0)}(p)$ and $B^{(0)}(p)$ can be verified numerically. Indeed, in Fig. 10.4 I show the numerical results for $A^{(0)}(p)$ and $B^{(0)}(p)$ in physical units for all of the different μ values. It can therefore be concluded that tree–level form of $A^{(0)}(p)$ and $B^{(0)}(p)$ are just constants, i.e.

$$B^{(0)}(p) = \mathcal{Z}_\psi^{(0)}\mu = m_q \quad \text{and} \quad A^{(0)}(p) = 1, \tag{10.32}$$

hence, when we perform a tree–level correction on $M(q^2)$, we see that the corrected

Table 10.3: Parameters for Overlap fermions. $N_{[U]}$ is the number of gauge field configurations considered in the simulation. The corresponding masses, $m_q = \mathcal{Z}_\psi^{(0)}\mu$ for the $12^3 \times 24$ are $\{103.69, 120.97, 138.25, 172.81, 207.38\}$ MeV for the first five and $\{259.22, 345.63, 432.04, 518.45, 604.85\}$ MeV for the remaining five.

Quark Action	a fm	$N_{[U]}$	u_0	μ	$\kappa_{cr}^{(0)}$
Overlap Wilson	0.125	50	0.88888	0.024,0.028 0.032,0.400 0.048,0.060 0.080,0.100 0.120,0.140	0.1250

	κ	$m = \frac{1}{2}[(\kappa_{cr}^{(0)})^{-1} - \kappa^{-1}]$	$\mathcal{Z}_\psi^{(0)} = 2m$	$\mathcal{Z}_\psi$	κ_{cr}
	0.19	1.36842	2.73684	1.93101	0.139

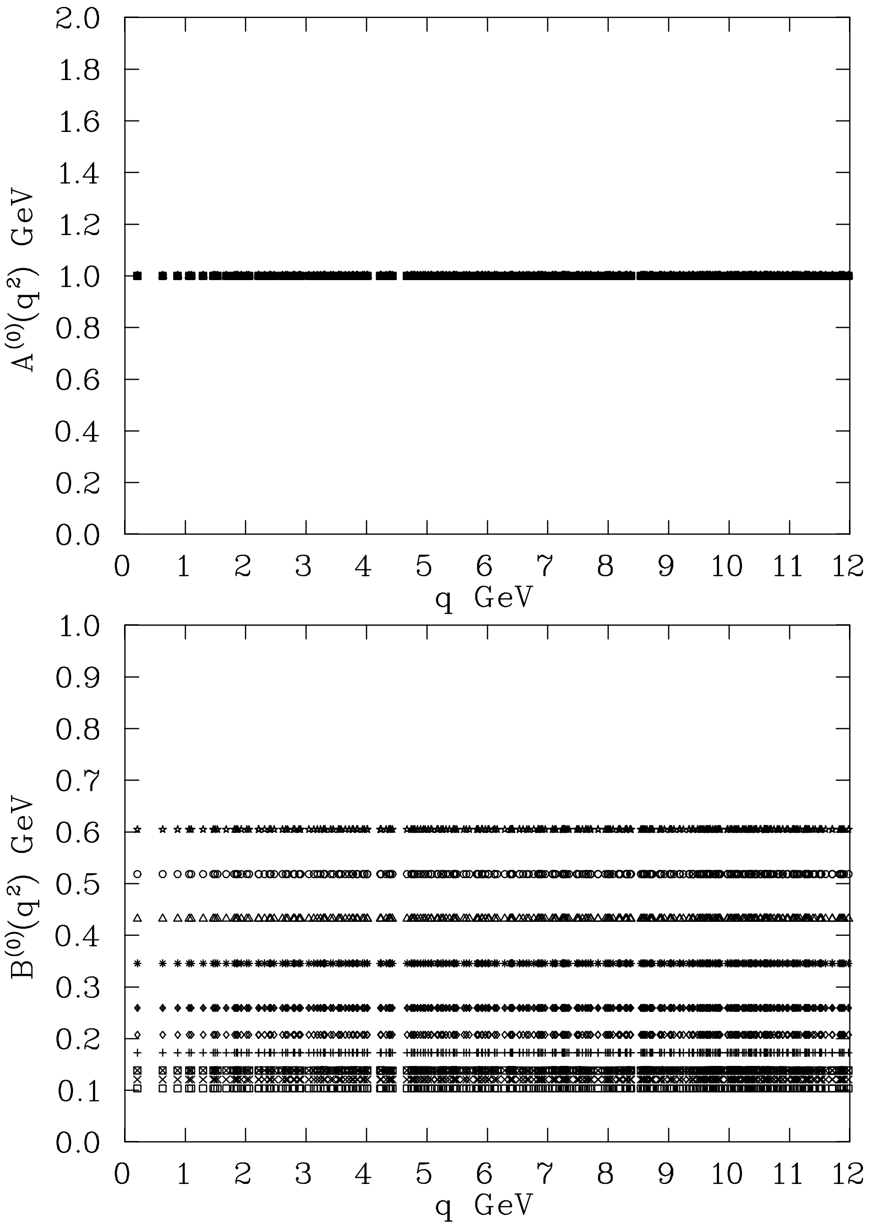


Figure 10.4: The tree–level values for the $A^{(0)}(p)$ and $B^{(0)}(p)$ functions in GeV versus the lattice momentum q. Showing that $B^{(0)}(p) = \mathcal{Z}_{\psi}^{(0)}\mu = m_q$ and $A^{(0)}(p) = 1$. The top strip of points corresponds to the heaviest mass as opposed to the bottom band which corresponds to the lightest m_q.

mass becomes:

$$M^{(c)}(p) = \frac{B(p)}{B^{(0)}(p)} m_q \frac{A^{(0)}(p)}{A(p)} = \frac{B(p)}{A(p)} = M(q^2), \tag{10.33}$$

showing that the mass function is not affected by the tree–level correction procedure.

All of the propagators are plotted in physical units versus $p = \sqrt{\sum p_\mu^2}$ obtained from the discrete momentum value, Eq. (10.7). The scale has been determined by the static quark potential with a string tension of $\sqrt{\sigma} = 440$ MeV [28]. Just as for the gluon propagator work in Chap. 9

It is possible to use the fact that the hypercubic lattices preserves the subgroup of discrete rotations $Z(4)$. However the lattice size used in this analysis reduces this symmetry to a $Z(3)$ symmetry. This symmetry may be used to improve the statistics and reduce the number of identical momentum points. Since we are plotting versus $p = \sqrt{\sum p_\mu^2}$, that is a squared momentum value, we can also perform a reflection average. This average treats the negative momentum combinations in the same way as the positive ones. For example, consider $S(1, 2, 3, 4)$, then

$$\begin{aligned}
&S(1,2,3,4) = S(-1,2,3,4) = S(1,-2,3,4) = S(1,2,-3,4) = \\
&S(1,2,3,-4) = S(-1,-2,3,4) = S(-1,2,-3,4) = S(-1,2,3,-4) = \\
&S(-1,-2,-3,4) = S(-1,-2,3,-4) = S(-1,2,-3,-4) = S(1,-2,-3,4) = \\
&S(1,-2,3,-4) = S(1,2,-3,-4) = S(1,-2,-3,-4) = S(-1,-2,-3,-4).
\end{aligned}$$

The propagator is calculated for each of these values and then averaged.

10.4.1 Data Cuts

Having identified possible lattice artifacts, cuts may be applied to clean up the data, making it easier to draw conclusions about continuum physics. I employ the same method to cut out data as in Refs. [77, 78, 62, 53]. The method is briefly reviewed in Sec. 9.5 .

10.4.2 The Results for the Overlap Fermion

I first start by showing the mass and renormalization function, $M(q^2)$ and $Z^{(\mathrm{R})}(q^2)$ respectively, calculated at ten different masses, Fig. 10.5. The functions are plotted versus the discrete momentum p and no tree–level correction is applied here. The functions have been half cut over the Brillouin zone, the mass function is shown in the top half of the figure whereas $Z^{(\mathrm{R})}(q^2)$ is shown in the bottom half.

QCD is a renormalizable theory which means that the bare propagator is related to the renormalized one by the quark wavefunction renormalization constant, Z_2

$$S^{\mathrm{bare}}(a;p) \equiv Z_2(\zeta;a) S(\zeta;p). \tag{10.34}$$

Here ζ is the renormalization point. In such theory, the quantities that are renormalizable become independent of the regularization parameter in the continuum limit. Lattice simulation produces the regularized (but not renormalized) propagator which means that the scale of the field renormalization function, $Z^{(\mathrm{R})}(q^2)$, is dependent of the renormalization point unlike the mass function, $M(q^2)$. Therefore a change of renormalization point is just an overall rescaling of the Z function [77]. In this simulation, I set the renormalization point for $Z^{(\mathrm{R})}(q^2)$ at $\zeta = 3.9$ GeV when the data is plotted versus p.

From the mass function graph we can see the behaviour of the function is very well behaved, producing very clear trajectories all the way to 5 GeV. The top set of points comes from the largest μ value corresponding to the heaviest quark mass amongst the parameter set as opposed to the lower set of points corresponding to the lightest quark mass. We can also see that the fall off in the mass function becomes sharper as the quark mass is driven towards lighter values. This is confirmed in the behaviour of the $Z^{(\mathrm{R})}(q^2)$ function where the size of the dip is a lot more pronounced at lighter quark mass. The size of the dip, in the deep infrared, is a clear indication of the amplitude of the chiral symmetry breaking taking place in the mass function. I also plot these

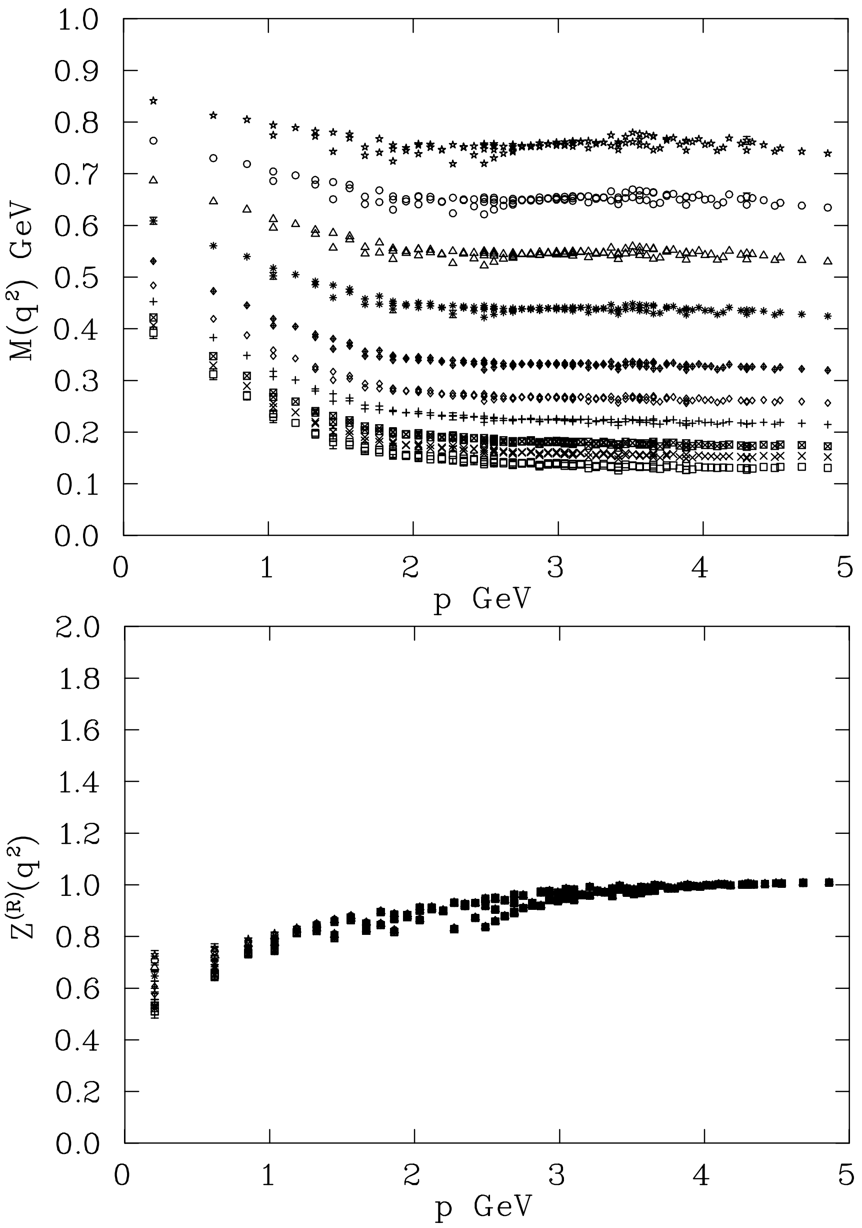


Figure 10.5: The functions $M(q^2)$ and $Z^{(\mathrm{R})}(q^2)$, (half cut data), for the Overlap fermions plotted versus the discrete momentum values defined in Eq. (10.7), $p = \sqrt{\sum p_\mu^2}$, over the interval of [0,5] GeV. The $Z^{(\mathrm{R})}(q^2)$ function has been renormalized at $\zeta \sim 3.9$ GeV. The μ values, from top to bottom set of points, are $\mu = \{0.14, 0.12, 0.10, 0.08, 0.06\}$ for the first five and $\{0.048, 0.040, 0.032, 0.028, 0.024\}$ for the other five, on the $12^3 \times 24$ at $\beta = 4.60$.

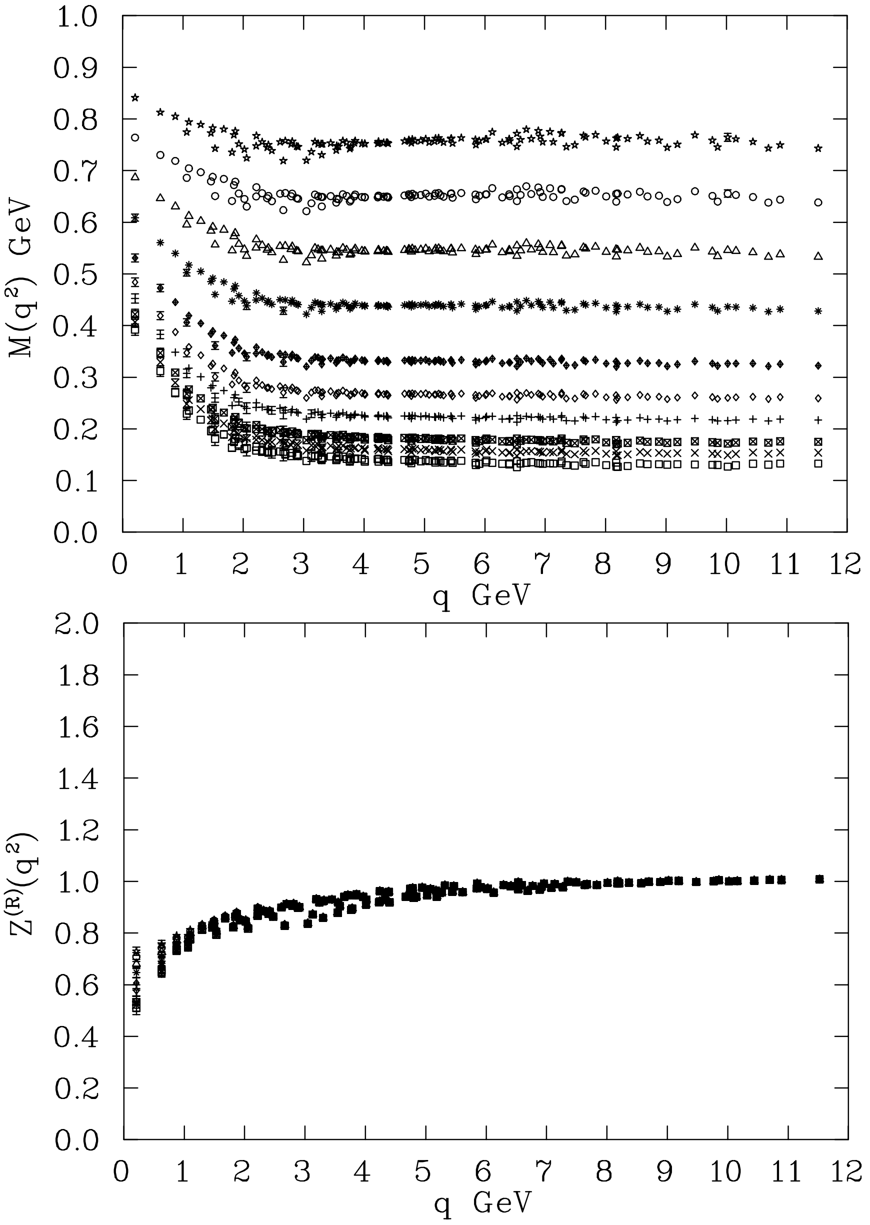


Figure 10.6: The functions $M(q^2)$ and $Z^{(\mathrm{R})}(q^2)$, (half cut data), for the Overlap fermions plotted versus the lattice momentum values defined in Eq. (10.25), $q = \sqrt{\sum q_\mu^2}$, over the interval of [0,12] GeV. The $Z^{(\mathrm{R})}(q^2)$ function has been renormalized at $\zeta \sim 8.2$ GeV. The μ values, from top to bottom set of points, are $\mu = \{0.14, 0.12, 0.10, 0.08, 0.06, 0.048, 0.040, 0.032, 0.028, 0.024\}$, on the $12^3 \times 24$ at $\beta = 4.60$.

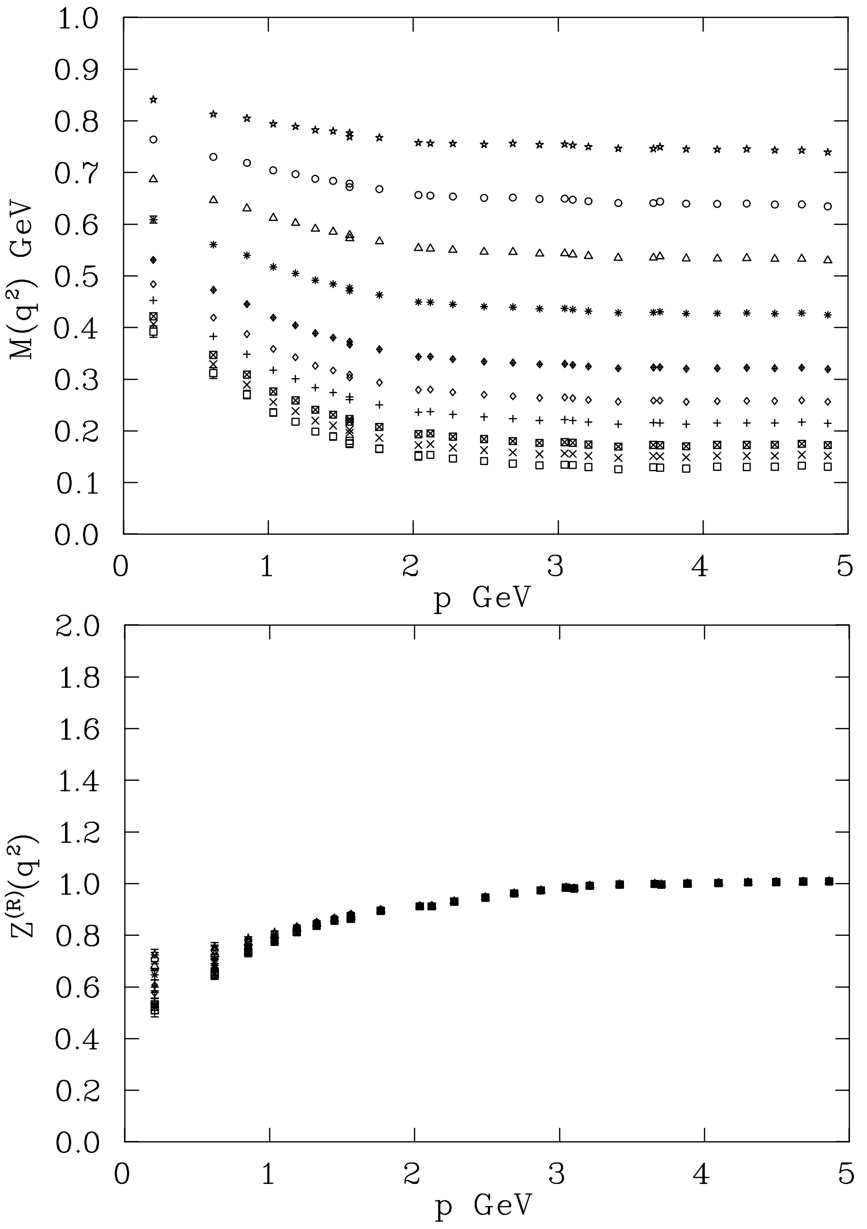


Figure 10.7: The function $M(q^2)$ and $Z^{(\mathrm{R})}(q^2)$, (cylinder cut) for the Overlap fermions plotted versus the discrete momentum values defined in Eq. (10.7), $p = \sqrt{\sum p_\mu^2}$, over the interval of [0,5] GeV. The $Z^{(\mathrm{R})}(q^2)$ function has been renormalized at $\zeta \sim 3.9$ GeV. The μ values, from top to bottom set of points, are $\mu = \{0.14, 0.12, 0.10, 0.08, 0.06\}$ for the first five and $\{, 0.048, 0.040, 0.032, 0.028, 0.024\}$ for the other five, on the $12^3 \times 24$ at $\beta = 4.60$.

functions versus the lattice momentum q, this is shown in Fig. 10.6. In that case we see that the points are pushed out further along the momentum axis, all the way to 12 GeV. Comparing Figs. 10.5 and 10.6, shows that the infrared region remains totally consistent in both cases. In Fig. 10.6, $Z^{(\mathrm{R})}(q^2)$ has been renormalized at $\zeta = 8.2$ GeV.

In Fig. 10.7 and Fig. 10.8, I show the same data but this time with a cylinder cut applied onto it, the trajectories become clearer and distinct in both of the cases when plotted versus p and q. So far it is not very clear versus which momentum one should be plotting the propagators in order to keep the anisotropy under control. One

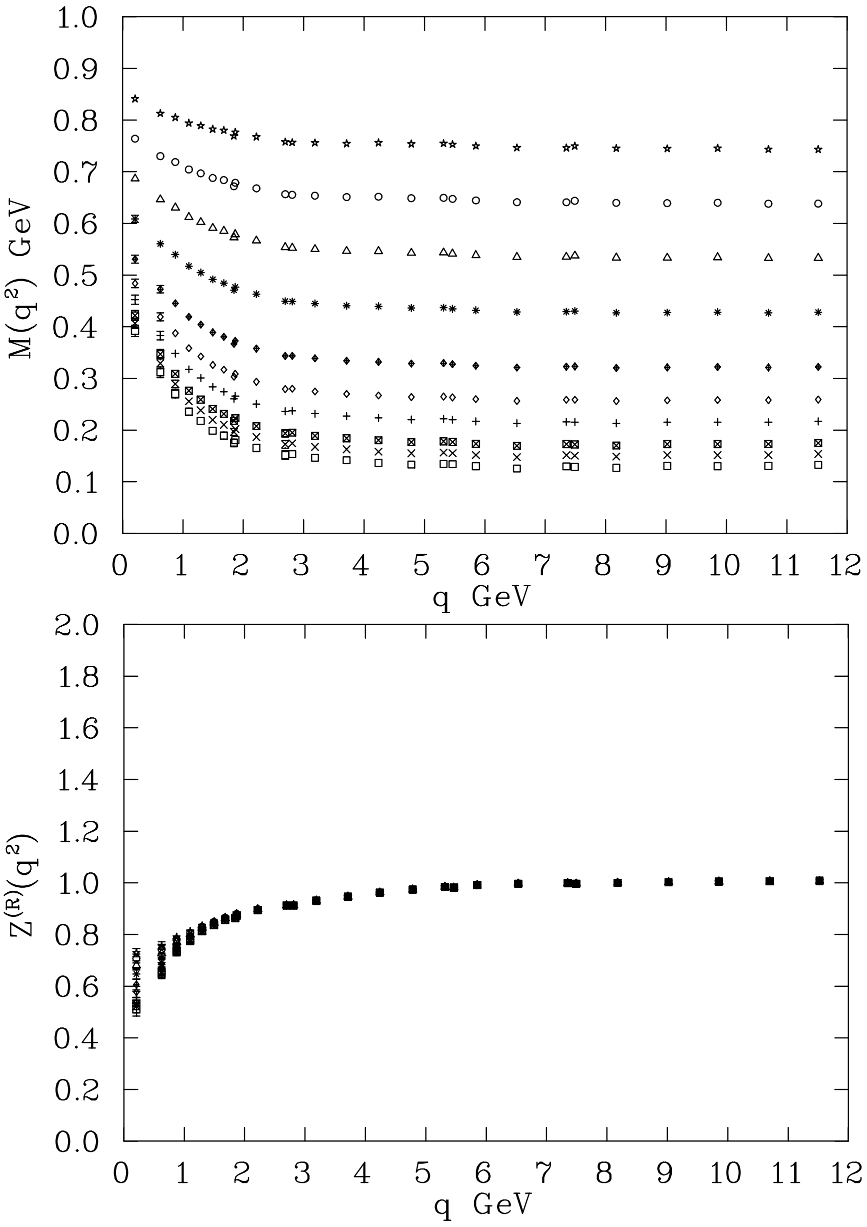


Figure 10.8: The functions $M(q^2)$ and $Z^{(\mathrm{R})}(q^2)$, (cylinder cut) for the Overlap fermions plotted versus the lattice momentum values defined in Eq. (10.25), $q = \sqrt{\sum q_\mu^2}$, over the interval of [0,12] GeV. The $Z^{(\mathrm{R})}(q^2)$ function has been renormalized at $\zeta \sim 8.2$ GeV. The μ values, from top to bottom set of points, are $\mu = \{0.14, 0.12, 0.10, 0.08, 0.06\}$ for the first five and $\{0.048, 0.040, 0.032, 0.028, 0.024\}$ for the other five, on the $12^3 \times 24$ at $\beta = 4.60$.

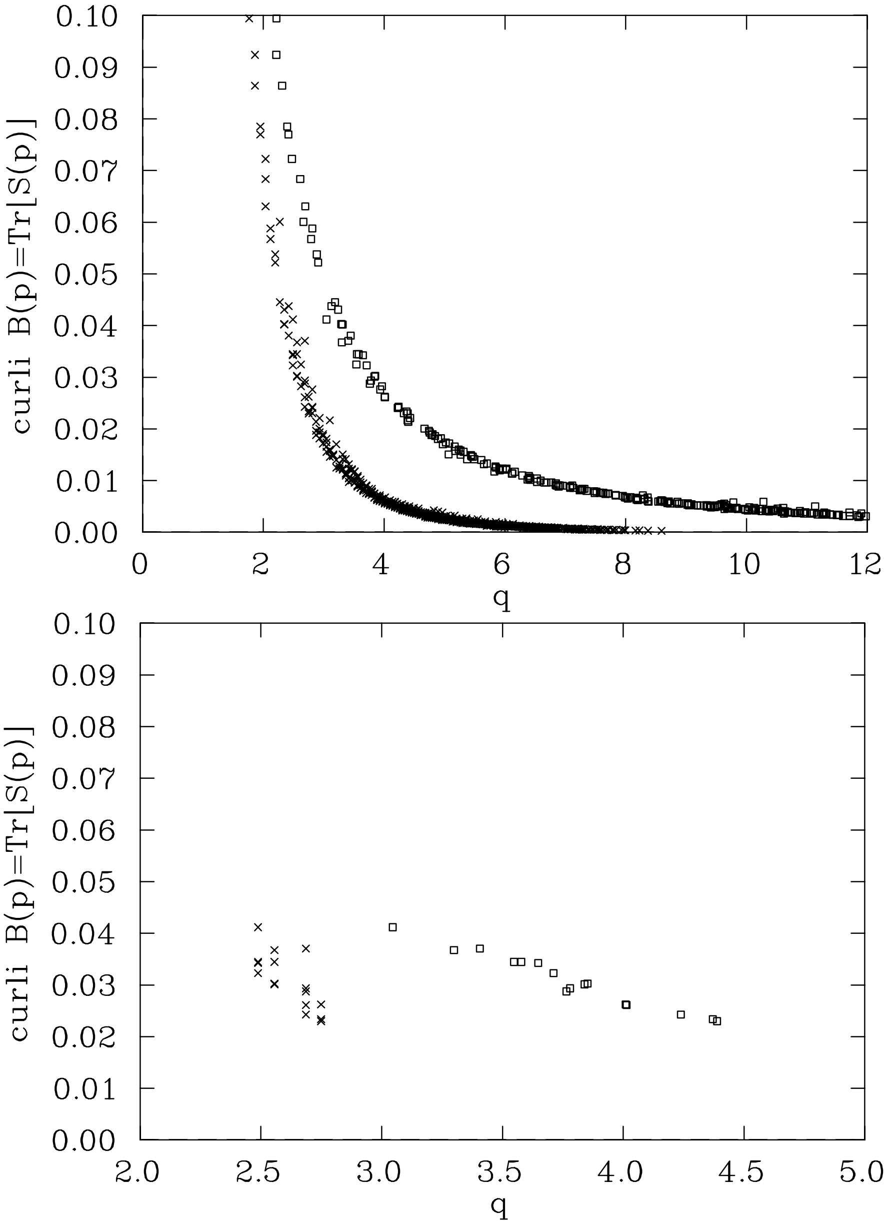


Figure 10.9: The function $\mathcal{B}(p)$, (full data) plotted versus both the discrete momentum ($\times$) and the lattice momentum values defined in Eq. (10.25) ($\Box$), over the interval of [0,12] GeV. The graph corresponds to $\mu = 0.0240$. The bottom graph shows an enlargement of a subset of points. The points ($\Box$) are more stretched out and form a smoother curve than when the function is plotted versus p ($\times$).

way to see which possesses the least anisotropy is to select a small subset of points and examine the spread of points. For example in Fig. 10.9, I plot the trace of the propagator, $\mathcal{B}$, versus both p ($\times$) and versus q ($\Box$) when $\mu = 0.0240$ for a given subset of the momentum points that were the most recurrent. The spread of points appears to be reduced when $\mathcal{B}$ is plotted versus the lattice momentum, Eq. (10.25), therefore suggesting that there is more anisotropy when the data is plotted versus p than when it is plotted versus q.

10.4.3 Wilson Fermion versus the Overlap Fermion

The technique of tree–level correction is a powerful tool when the quark action possesses large $\mathcal{O}(a)$ errors. The data presented in Figures 10.2, 10.3 and 10.10 is over half

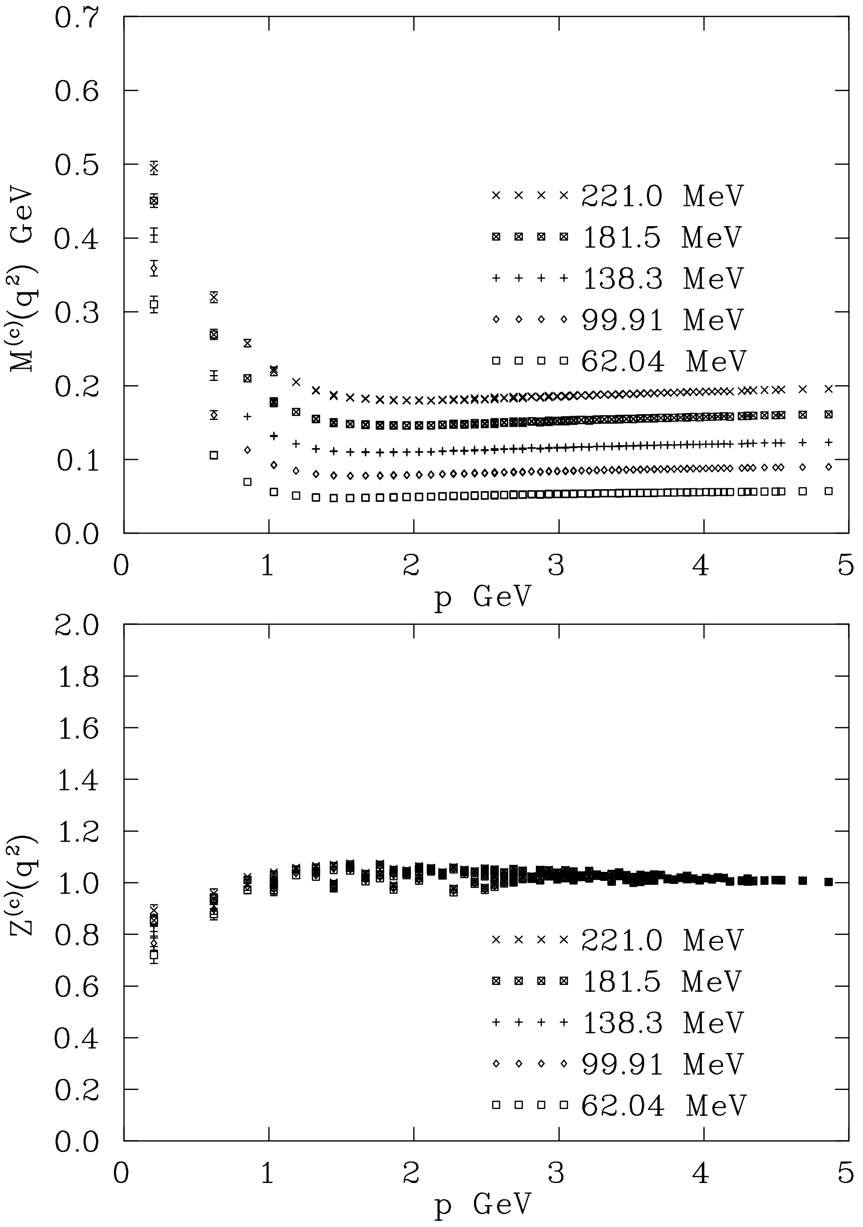


Figure 10.10: The corrected mass function, $M^{(c)}(p)$, and $Z^{(c)}(p)$ (half cut) for the Wilson fermions plotted versus the discrete momentum values defined in Eq.(10.7), $p = \sqrt{\sum p_\mu^2}$, over the interval of [0,5] GeV. The kappa values, from top to bottom set of points, are $\kappa = 0.1337, 0.1346, 0.1356, 0.1365, 0.1374$ corresponds to a current quark mass of $m_q{\sim}221, 181.5, 138.3, 99.91, 62.04$ MeV respectively, on the $12^3 \times 24$ at $\beta = 4.60$.

the momentum region, i.e. half cut. A direct comparison of Figs. 10.3 and 10.10 clearly shows the effectiveness of the technique and that it is possible to obtain a mass function that converges to the current quark mass (up to logarithmic corrections) in the ultraviolet region. A similar scenario is seen in comparing Figs. 10.2 and 10.11 where

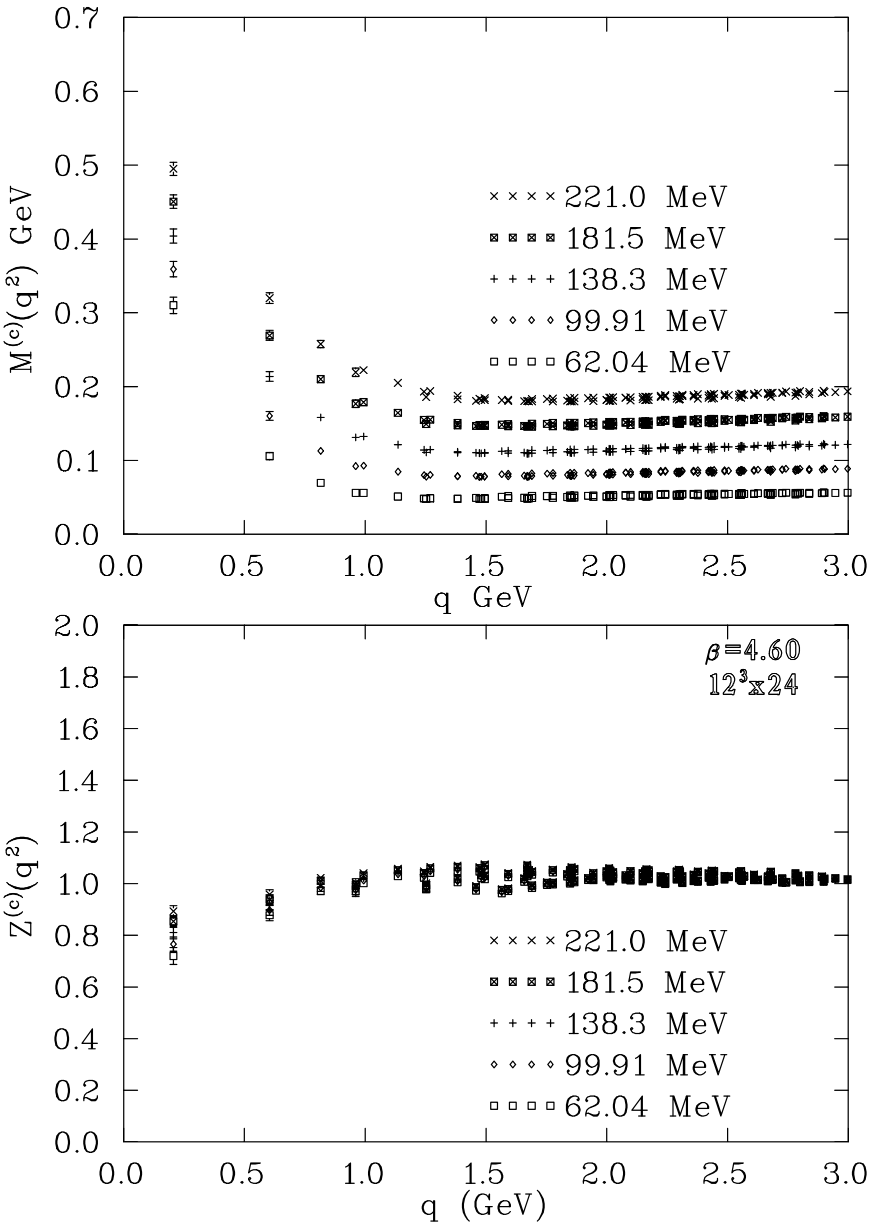


Figure 10.11: The corrected mass function, $M^{(c)}(p)$, and $Z^{(c)}(p)$ (half cut) for the Wilson fermions plotted versus the discrete momentum values defined in Eq.(10.25), $q = \sqrt{\sum q_\mu^2}$, over the interval of [0,5] GeV. The kappa values, from top to bottom set of points, are $\kappa = 0.1337, 0.1346, 0.1356, 0.1365, 0.1374$ corresponds to a current quark mass of $m_q{\sim}221, 181.5, 138.3, 99.91, 62.04$ MeV respectively, on the $12^3 \times 24$ at $\beta = 4.60$.

the lattice momentum q has been used instead of discrete momentum p. In Chapter 9, we saw that in order to recover a well behaved gluon propagator in the ultraviolet region, tree–level correction was a crucial step. On the other hand, even without any tree–level correction, it was possible to extract from the untouched propagator some results in the infrared region. Here however, the $\mathcal{O}(a)$ errors for the Wilson quark action are relatively large making it difficult to say anything definitive about either the infrared and the ultraviolet region. Comparing these results with Figs. 10.5 and 10.6 where no tree–level correction was required, we see clear evidence of the effects of action improvement for the overlap case.

It is interesting to note from Fig. 10.10, where both functions are plotted versus p and from Fig. 10.11 where everything is plotted versus q, the way the points are shifted along the x–axis. When the functions are plotted versus the discrete momentum, the points are moved away from the origin as opposed to the case when the functions are plotted versus the lattice momentum q. This is completely the opposite to the overlap as we saw from Figs. 10.5 and 10.6. This may be understood by looking at Fig. 10.1, for the overlap a graph of p against q ($\Box$) drive the set of points sharply above the line $p = q$, contrarily to the wilson fermion ($\circ$) with its sinusoidal momentum structure for the lattice momentum.

It is therefore crucial, in order to understand the true nature of the behaviour for both of these functions to first get some insights on the lattice momentum extracted from Eq. (10.25).

In fig. 10.12, I show the same data as in Fig. 10.10 but cylinder cut this time. A clear signal for each of the different masses is obtained and we can really see the structure of the curve. The renormalization function $Z^{(c)}(p)$ for the Wilson fermions is not affected by tree–level correction because $A^{(0)}(p) = 1$, but we can see a dip in the deep infrared as the quark mass is driven towards its chiral limit. This is perfectly consistent with the overlap fermion and previous studies. A particular aspect of the mass function that one should notice is that as $m_q \longrightarrow 0$ a dip in the infrared is enhanced. The second thing is the significant curvature of the function in the region between 1 to 3 GeV for the Wilson. This curvature appears to be part of the action and was also observed in some work done with non–perturbatively improved action [77, 78].

10.4.4 Linear Extrapolated Values

In this section I report the results from a linear extrapolation to the chiral limit, for both $M(q^2)$ and $Z^{(\mathrm{R})}(q^2)$. In the infrared region both $Z^{(\mathrm{R})}(q^2)$ and $M(q^2)$ strongly deviate from their perturbative behaviour, this enhancement is a characteristic of dynamical chiral symmetry breaking. Similarly for $Z^{(\mathrm{R})}(q^2)$ at large p, $Z^{(\mathrm{R})}(q^2) = 1$, up to small logarithmic corrections. As in DSE studies it is typically found that $Z^{(\mathrm{R})}(q^2)$, at small p, the function is approximately 0.5 or so [94].

In the top graph of Fig. 10.13, the linearly extrapolated mass function, $M(q^2)$, is plotted versus p ($\times$) and in the bottom graph the same data but plotted versus q ($\Box$). Both graphs contain all the available data (full data). Comparing the two using the cylinder cut data shown in Fig. 10.14, shows a more rapid fall off in the chiral limit when plotted versus p than when plotted versus q. In the deep infrared I find $M(0) = 297(11)$ MeV when plotted versus either. A similar effect is observed for the linearly extrapolated $Z^{(\mathrm{R})}(q^2)$ function, shown in the bottom half of Fig. 10.14. In the low momentum region $Z^{(\mathrm{R})}(q^2)$ reports a value of 0.48(2).

The Wilson fermion reports a slightly different value for the extrapolated $Z^{(c)}(p)$, 0.65(3), Fig. 10.15. There is a clear distinction between the two actions in the way they behave. The Wilson has this obvious bump in the region from 1 to 2.5 GeV, this bump gets carried over to the mass function. Hence there is a clear signal of the $\mathcal{O}(a)$ lattice artifacts in the $A(p)$ function for the Wilson fermions. Looking at Fig. 10.15 the linearly extrapolated mass function, clearly shows the transcription from a bump to a dip. This dip was also observed in the work with $\mathcal{O}(a)$ improved quark propagators used in [77, 78]. These propagators ignore the gauge non–invariant terms. A discussion of this procedure may be found in [92, 93]. The Wilson action is not really suitable

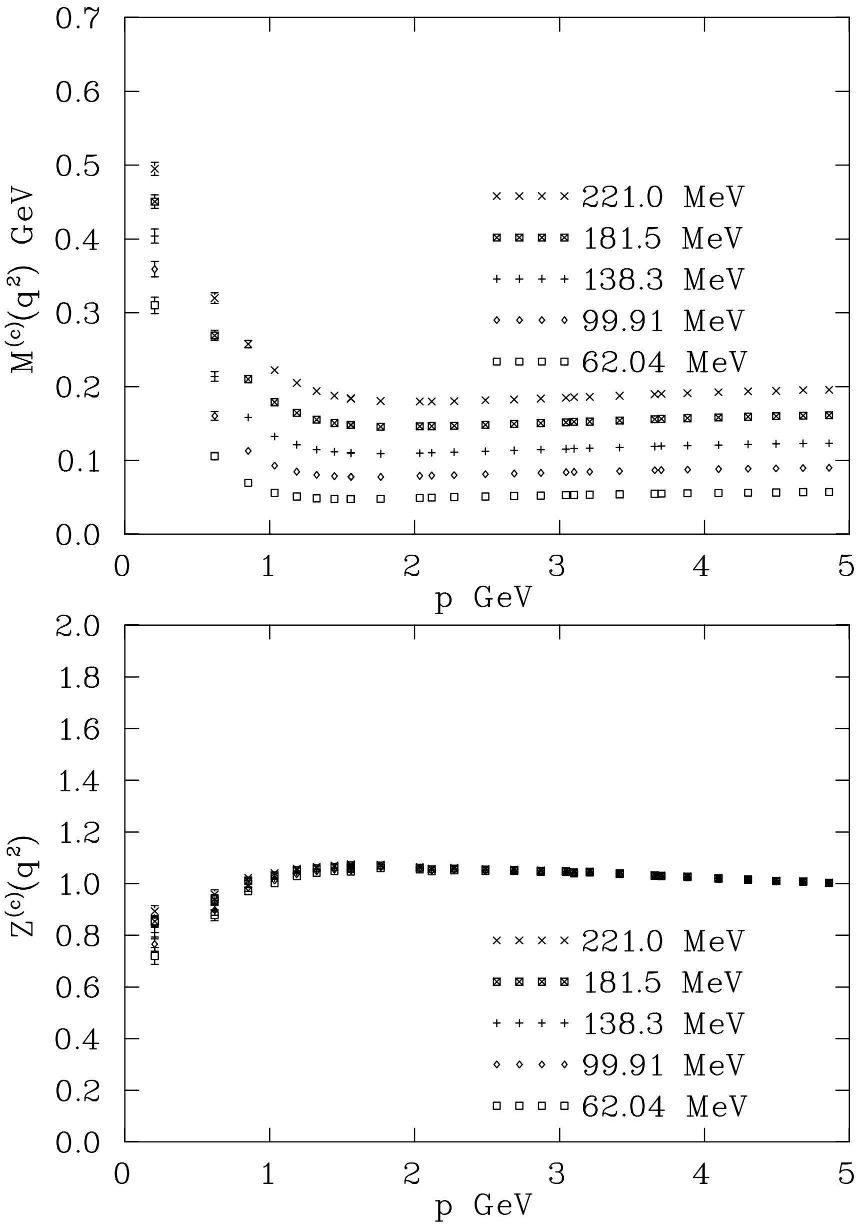
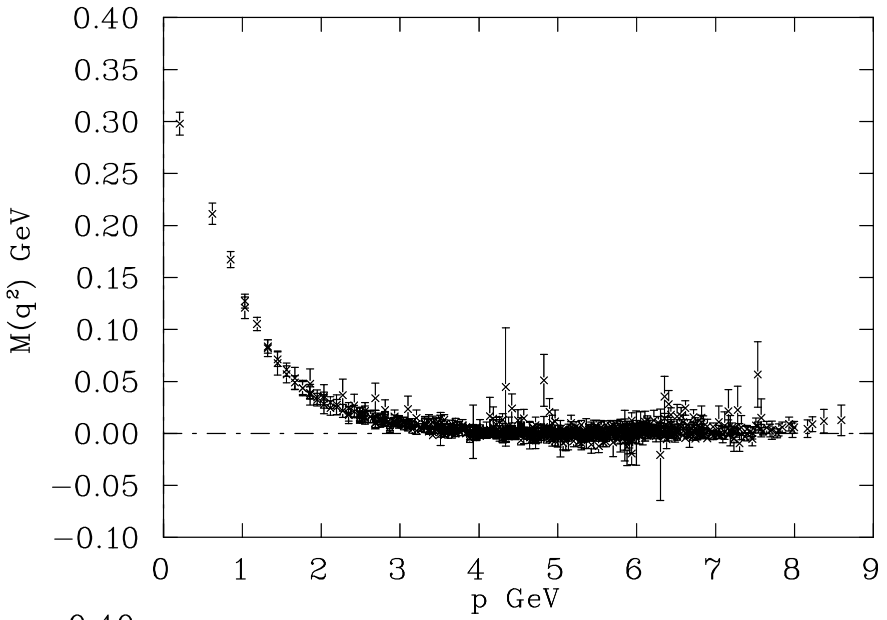


Figure 10.12: The corrected mass function, $M^{(c)}(p)$, and $Z^{(c)}(p)$ (cylinder cut) for the Wilson fermions plotted versus the discrete momentum values defined in Eq.(10.25), $p = \sqrt{\sum p_\mu^2}$, over the interval of [0,5] GeV. The kappa values, from top to bottom set of points, are $\kappa = 0.1337, 0.1346, 0.1356, 0.1365, 0.1374$ corresponds to a current quark mass of $m_q{\sim}221, 181.5, 138.3, 99.91, 62.04$ MeV respectively, on the $12^3 \times 24$ at $\beta = 4.60$.

for extracting the quark condensate. So from now on I will concentrate on the overlap results.

In the continuum, at one loop order in perturbation theory, the mass function in the assymptotic regime is given by

$$M(p^2)^{p^2 \overset{\longrightarrow\infty}{=}} \frac{\hat{m}}{\left(\frac{1}{2}\ln\left[\frac{p^2}{\Lambda^2_{\rm QCD}}\right]\right)^{\gamma_m}}, \tag{10.35}$$



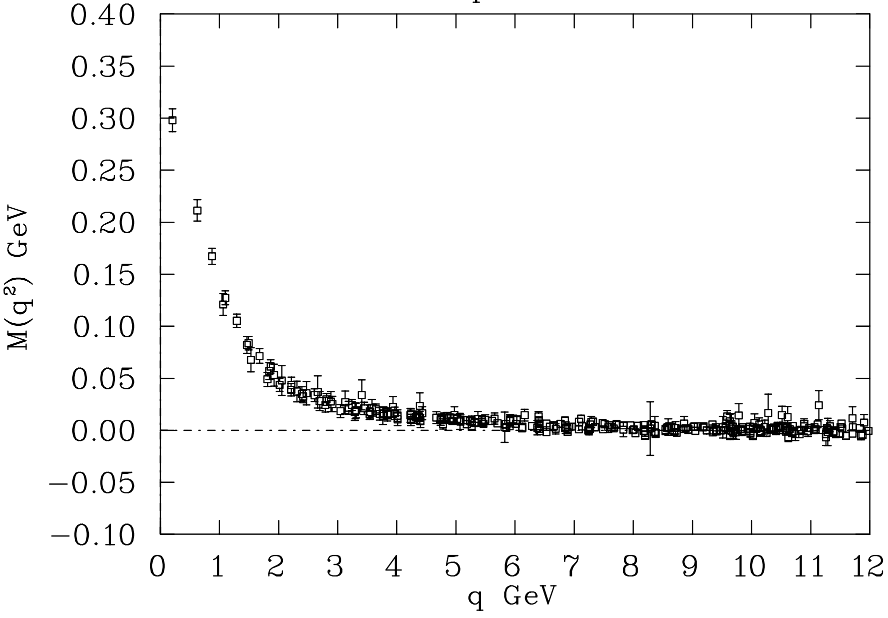


Figure 10.13: The linearly extrapolated mass function, $M(q^2)$, (full data) for the Overlap fermions plotted versus both the discrete momentum values defined in Eq. (10.7) and the momentum extracted from the lattice, Eq. (10.25). Here $M(0) = 297(11)$ MeV.

with $\gamma_m = 12/(33-2N_f)$. This is referred to as the running quark mass, $m(\mu) \equiv M(\mu^2)$. Here the number of flavours is zero, i.e. $N_f = 0$, since we are working in quenched QCD and $\hat{m}$ is the renormalization point independent current quark mass. In QCD, γ_m is independent of the gauge parameter to all orders and the chiral limit is defined by taking $\hat{m} = 0$ in the chiral limit $M(p^2) \neq 0$ is only possible if and only if the quark condensate is non–vanishing. In the presence of explicit chiral symmetry breaking, Eq. (10.35) describes the form of $M(p^2)$ for $p^2 > 1\,\mathrm{GeV}^2$. In the chiral limit, however the ultraviolet behaviour [4, 94] is given by

$$M^0(p^2) \overset{p^2 \to \infty}{=} \frac{4\pi^2\gamma_m}{3} \frac{\left(-\langle \overline{q}q\rangle^0\right)}{p^2\left(\ln\left[\frac{p^2}{\Lambda^2_{\mathrm{QCD}}}\right]\right)^{1-\gamma_m}}, \tag{10.36}$$

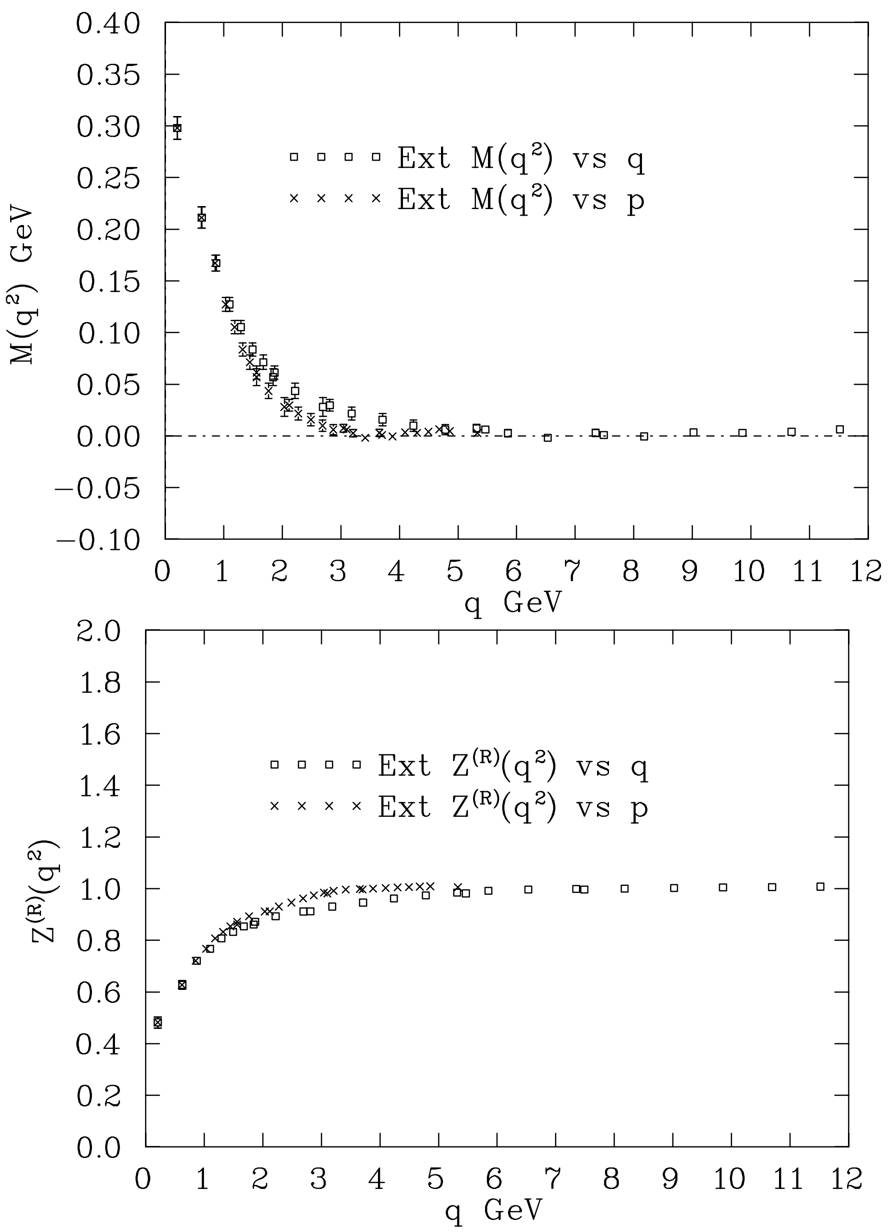


Figure 10.14: The linearly extrapolated mass function, $M(q^2)$, (cylinder cut) for the Overlap fermions plotted versus both the discrete momentum values defined in Eq. (10.7) and the momentum extracted from the lattice, Eq. (10.25). Here $M(0) = 297(11)$ MeV and $Z(0) = 0.48(2)$.

where $(-\langle \overline{q}q\rangle^0)$ is the renormalization point independent vacuum quark condensate.

A gauge invariant expression for the renormalization–point–dependent vacuum quark condensate was derived in [96]. At one–loop order the renormalization point dependent vacuum quark condensate is given by

$$\langle \overline{q}q\rangle_\zeta = \left[\ln\left(\frac{\zeta^2}{\Lambda_{\rm QCD}^2}\right)\right]^{\gamma_m} (-\langle \overline{q}q\rangle^0). \tag{10.37}$$

Using the lattice data I extract $(-\langle \overline{q}q\rangle^0)$ from Eq. (10.36). Here I use the full data, and a QCD scale of $\Lambda_{\rm QCD} = 0.234$ GeV [94, 95] as the major scale parameter on various fitting regions. I also explore different scales like $\Lambda_{\rm QCD} = 200$, 300 and 380 MeV.

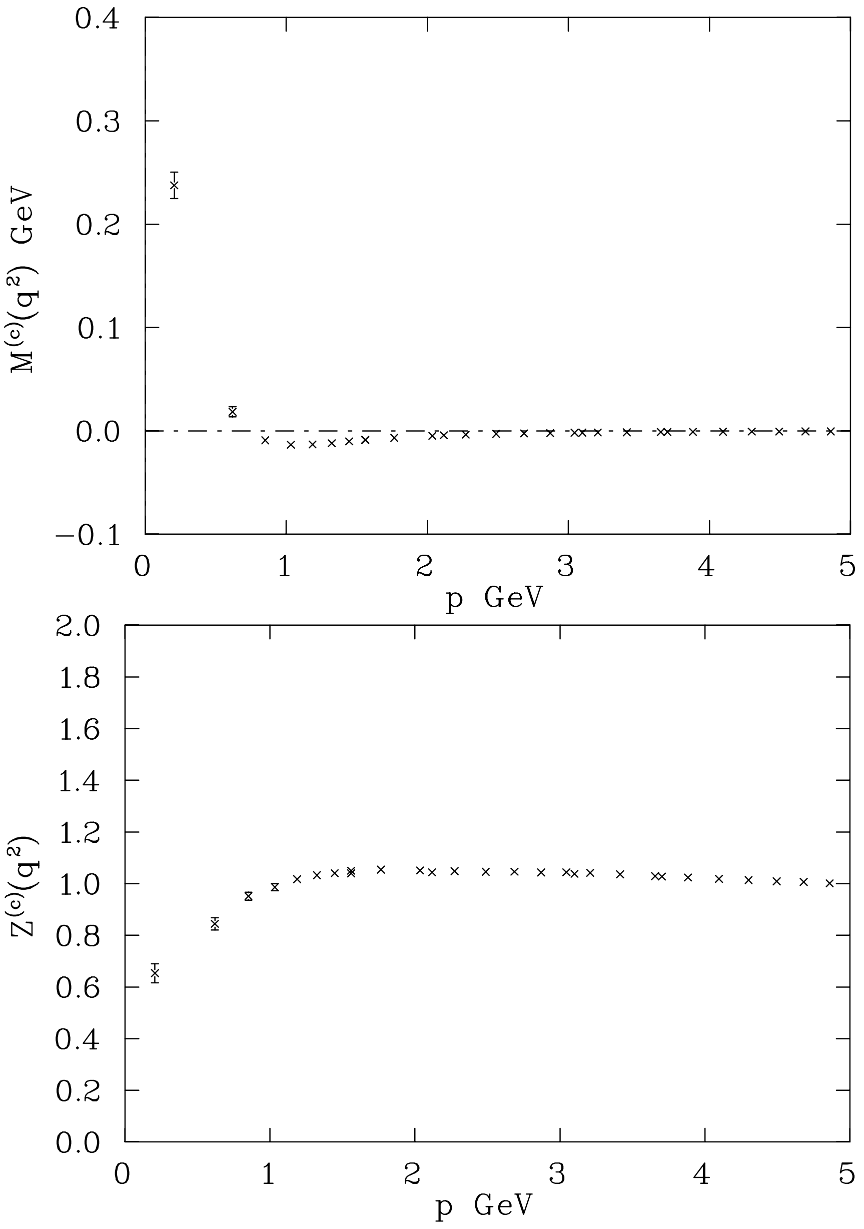
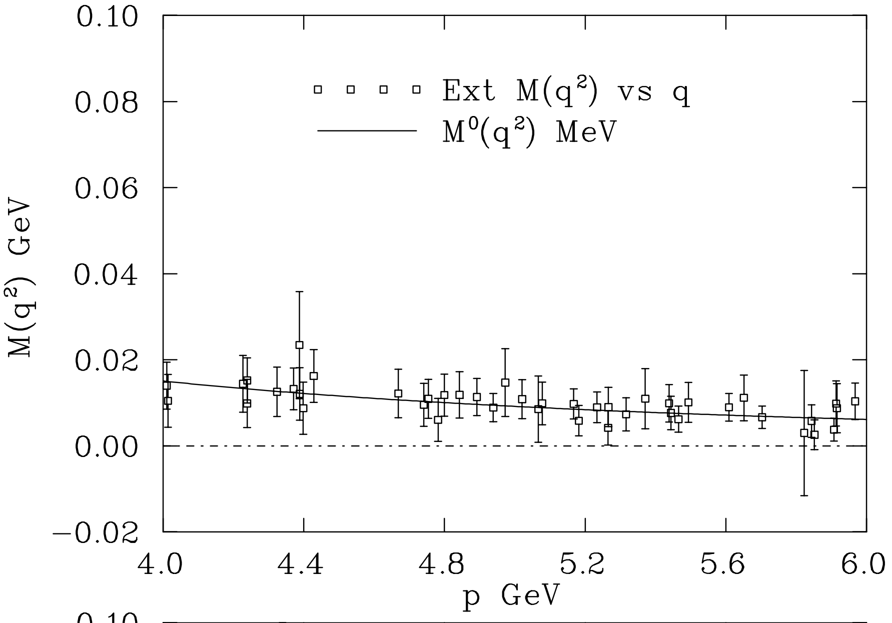


Figure 10.15: The linearly extrapolated mass, $M(q^2)$, and $Z^{(c)}(p)$ function (cylinder cut), for the Wilson fermions plotted versus the discrete momentum values defined in Eq.(10.25), $p = \sqrt{\sum p_\mu^2}$. Here $M(0) = 237(13)$ MeV. The lowest momentum value gives $Z(0) = 0.65(3)$.

I have fitted the lattice data versus both definitions of the momentum, and found that when the fits were performed versus the discrete momentum the quark condensate was closer to its continuum, i.e., when calculated from phenomenological studies. A summary of the fitting results is shown in Table 10.4 for various fitting regions and scales when the data is plotted versus the discrete momentum and in Table 10.5 for the lattice momentum. A typical fitting window for the quark condensate is chosen to be $p = [4, 6]$ GeV, the resulting fit is shown in Fig. 10.16. Setting the renormalization point at $\zeta = 1$ GeV and using Eq. (10.37), we obtain a value for the renormalization point dependent quark condensate of $\langle \overline{q}q \rangle_{\zeta=1\,GeV} = -(286\,\mathrm{MeV})^3$ when the data is plotted versus the discrete momentum p and a condensate of $\langle \overline{q}q \rangle_{\zeta=1\,GeV} = -(608\,\mathrm{MeV})^3$ when



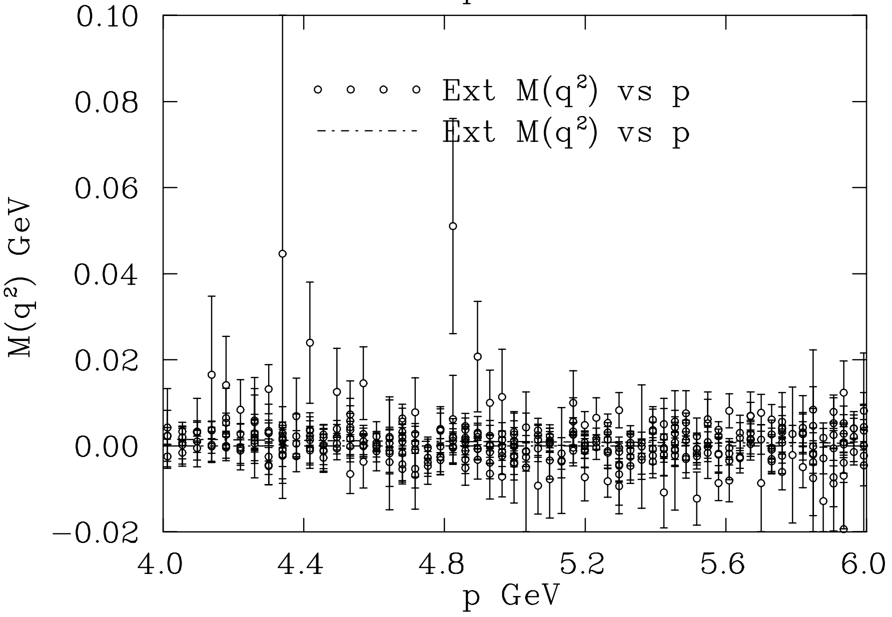


Figure 10.16: The linearly extrapolated mass function, $M(q^2)$, (full data) for the Overlap fermions plotted versus the lattice momentum (□) defined in Eq. (10.25), $q = \sqrt{\sum q_\mu^2}$ (top graph). The quark condensate estimated from Eq. (10.36), gives $\langle \overline{q}q \rangle_{\zeta=1\,GeV} = -(607\,\mathrm{MeV})^3$. In the bottom graph we show the graph when the $M(q^2)$ is plotted and fitted versus the discrete momentum $p = \sqrt{\sum p_\mu^2}$, the resulting condensate is $\langle \overline{q}q \rangle_{\zeta=1\,GeV} = -(286\,\mathrm{MeV})^3$. $\Lambda_{\mathrm{QCD}} = 0.234$ GeV.

plotted versus the lattice momentum q.

The programs used to carry out the above analysis may be found in Appendix E.23 and E.24.

Table 10.4: Summary of the results for $\langle\bar{q}q\rangle_\zeta$ extracted from Eq. (10.37) in MeV and renormalized at $\zeta = 1.0$ GeV. The fit was done using Eq. (10.36) at various scales $\Lambda_{\rm QCD} = 200, 234, 300$ and 380 MeV on various momentum windows for the discrete momentum p in GeV.

$\Lambda_{\rm QCD}$	200 MeV		234 MeV		300 MeV		380 MeV	
p	$\langle\bar{q}q\rangle_\zeta$	χ^2/df	$\langle\bar{q}q\rangle_\zeta$	χ^2/df	$\langle\bar{q}q\rangle_\zeta$	χ^2/df	$\langle\bar{q}q\rangle_\zeta$	χ^2/df
3-5	342.66(30)	0.92	334.60(29)	0.92	320.79(28)	0.92	306.14(27)	0.92
3-6	337.64(34)	0.97	329.76(33)	0.97	316.27(31)	0.97	301.94(30)	0.97
3-7	339.97(32)	1.01	332.03(31)	1.01	318.46(29)	1.01	304.04(28)	1.01
3-8	340.37(31)	0.98	332.43(30)	0.98	318.83(29)	0.98	304.40(28)	0.98
3-9	340.48(31)	0.98	332.53(30)	0.98	318.93(29)	0.98	304.49(28)	0.98
4-5	289.64(98)	0.74	282.95(96)	0.74	271.51(92)	0.74	259.36(88)	0.74
4-6	293.49(91)	0.89	286.75(89)	0.89	275.23(85)	0.89	262.99(82)	0.89
4-7	303.20(77)	0.96	296.24(75)	0.96	284.32(71)	0.96	271.67(69)	0.96
4-8	304.60(74)	0.94	297.60(72)	0.94	285.62(69)	0.94	272.91(66)	0.94
4-9	304.87(74)	0.93	297.86(72)	0.93	285.88(69)	0.93	273.15(66)	0.93
5-6	302.39(92)	1.03	295.56(90)	1.03	283.89(86)	1.03	271.49(83)	1.03
5-7	326.06(65)	1.09	318.72(63)	1.09	306.16(61)	1.09	292.82(59)	1.09
5-8	328.95(64)	1.03	321.54(62)	1.03	308.87(60)	1.03	295.41(57)	1.03
5-9	329.61(63)	1.03	322.18(62)	1.03	309.49(60)	1.03	295.99(57)	1.03

10.5 Conclusion

In this Chapter, I have presented the first study of the quark propagator using an overlap quark action. This study is based on a set of ten different quark masses. We can see considerable improvement in the behaviour of the quark propagator when the Dirac Wilson operator is inserted into the overlap fermion formalism which produces an $\mathcal{O}(a)$ improved quark action with an exact chiral symmetry.

The overlap quark propagator did not require any tree–level correction unlike the Wilson fermion quark action or any other actions resembling to the Wilson fermions. The overlap quark action produces remarkable results for both functions $Z^{(\rm R)}(q^2)$ and $M(q^2)$. From just a linear extrapolation I was able to produce consistent results calculated in DSE studies. In the deep infrared in the chiral limit, we get for the mass and renormalization function, $M(0) = 297(11)$ MeV and $Z(0) = 0.48(2)$ respectively. Comparing the results coming from the two quark actions clearly shows the superiority of the overlap formalism.

Using the extrapolated data, I was able to extract the quark condensate using the known one–loop renormalization group UV behaviour of chiral QCD preserved by DSE model, Eq. (10.36). From there the one–loop renormalized point dependent vacuum quark condensate at various QCD scales was calculated. At $\Lambda_{\rm QCD} = 0.234$ GeV with a fitting window of $p = [4, 6]$ GeV we obtain at a renormalization point of $\zeta = 1$ GeV, a condensate of $\langle\bar{q}q\rangle_{\zeta=1\,GeV} = -(286\,{\rm MeV})^3$ compared with a momentum window of $p = [3, 6]$ GeV giving a slightly higher condensate of $\langle\bar{q}q\rangle_{\zeta=1\,GeV} = -(330\,{\rm MeV})^3$ when the data is plotted versus the discrete momentum p compared with a condensate of $\langle\bar{q}q\rangle_{\zeta=1\,GeV} = -(608\,{\rm MeV})^3$ and $\langle\bar{q}q\rangle_{\zeta=1\,GeV} = -(599\,{\rm MeV})^3$ for the respective fitting window when the data is plotted versus the lattice momentum q. These results, obtain from a quenched simulation, are to be compared with earlier work [77, 78] giving a

Table 10.5: Summary of the results for $\langle \overline{q}q \rangle_\zeta$ extracted from Eq. (10.37) in MeV and renormalized at $\zeta = 1.0$ GeV. The fit was done using Eq. (10.36) at various scales $\Lambda_{\text{QCD}} = 200, 234, 300$ and 380 MeV on various momentum windows for the lattice momentum q in GeV.

Λ_{QCD}	200 MeV		234 MeV		300 MeV		380 MeV	
q	$\langle \overline{q}q \rangle_\zeta$	χ^2/df	$\langle \overline{q}q \rangle_\zeta$	χ^2/df	$\langle \overline{q}q \rangle_\zeta$	χ^2/df	$\langle \overline{q}q \rangle_\zeta$	χ^2/df
3- 5	610.80(76)	0.25	596.20(74)	0.25	571.22(71)	0.25	544.71(67)	0.26
3- 6	613.06(77)	0.28	598.60(76)	0.28	573.85(72)	0.28	547.57(69)	0.29
3- 7	603.21(71)	0.57	589.18(69)	0.57	565.18(66)	0.56	539.67(63)	0.56
3- 8	603.70(66)	0.64	589.75(64)	0.64	565.89(62)	0.64	540.52(59)	0.64
3- 9	599.89(61)	0.70	586.11(60)	0.70	562.52(57)	0.70	537.44(55)	0.70
3-10	599.97(56)	0.69	586.22(54)	0.69	562.69(53)	0.68	537.67(50)	0.68
3-11	599.12(51)	0.68	585.43(50)	0.68	561.99(48)	0.68	537.06(46)	0.68
3-12	598.14(48)	0.73	584.50(47)	0.73	561.14(45)	0.73	536.29(43)	0.73
4- 5	622.98(89)	0.23	608.54(87)	0.23	583.84(84)	0.24	557.62(80)	0.24
4- 6	621.79(87)	0.29	607.57(85)	0.29	583.27(81)	0.29	557.45(78)	0.29
4- 7	602.09(73)	0.66	588.52(72)	0.66	565.31(69)	0.66	540.65(66)	0.66
4- 8	603.06(66)	0.73	589.56(64)	0.73	566.45(62)	0.73	541.89(59)	0.73
4- 9	597.10(59)	0.78	583.80(57)	0.78	561.04(55)	0.78	536.84(53)	0.77
4-10	597.36(52)	0.74	584.09(51)	0.74	561.38(49)	0.74	537.24(47)	0.74
4-11	596.15(47)	0.73	582.94(46)	0.73	560.34(44)	0.73	536.30(42)	0.72
4-12	594.71(43)	0.78	581.56(42)	0.78	559.06(40)	0.77	535.11(39)	0.77
5- 6	620.49(85)	0.34	606.52(83)	0.34	582.63(80)	0.34	557.26(77)	0.34
5- 7	589.63(66)	0.82	576.53(65)	0.82	554.12(62)	0.82	530.30(59)	0.81
5- 8	593.97(58)	0.86	580.84(57)	0.86	558.39(55)	0.85	534.52(52)	0.85
5- 9	586.49(51)	0.87	573.60(50)	0.87	551.54(48)	0.87	528.09(46)	0.87
5-10	587.62(46)	0.81	574.74(45)	0.81	552.70(43)	0.81	529.26(42)	0.81
5-11	586.42(42)	0.78	573.60(41)	0.77	551.67(40)	0.77	528.33(38)	0.77
5-12	584.70(40)	0.82	571.94(39)	0.82	550.11(38)	0.82	526.89(36)	0.82

condensate $\langle \overline{q}q \rangle_{\zeta=1\,GeV} = -(248\,\mathrm{MeV})^3$ for the lattice fitted data and $\langle \overline{q}q \rangle_{\zeta=1\,GeV} = -(241\,\mathrm{MeV})^3$ from the DSE model [4, 95], moreover comparing the results to QCD sum rule phenomenological studies obtained a value of $-(236(8)\,\mathrm{MeV})^3$.

If we compare the results for the quark condensate obtained from the overlap and that obtained phenomenologically one, we observe that the condensate for the overlap is higher. As pointed out in Ref. [97], for a small quark mass, the quark condensate has the form

$$\langle \overline{q}q \rangle = \frac{\langle |Q| \rangle}{\mu V} + c_0 + c_1 \mu. \tag{10.38}$$

A fit of Eq. (10.38) was shown to be in agreement with the lattice data[1]. The first term in Eq. (10.38) arises from the zero modes. This effect is a direct consequence of the quenched approximation which gives rise to too many instantons. These instantons increase the number of zero modes which imply a high condensate. This effect is expected to be suppressed when the determinant of the fermion matrix is included in the dynamical fermion simulations.

This calculation was done at a relatively small volume with only one lattice. The objective of the calculation was to get some insight into the structure of the quark propagator in the overlap formalism. Further work is underway which will examine the volume dependence in this formalism.

Extensive improvement in the quality of the results may really be seen when examining finer lattices and larger volumes with or without inserting a clover term especially in the extraction of the quark condensate.

[1]The figure of interest is Fig. 1 and 2 in the first and second article of Ref. [97] respectively.

Chapter 11

Summary and Prospects

11.1 Summary

In this thesis, I have shown numerical results for the $2D$ ising model and for both $SU_c(2)$ and $SU_c(3)$ gauge field configurations. These results match perfectly within statistical error those published in previous studies, [98]. I have also shown the effect of $\mathcal{O}(a^2)$ improvement.

Smoothing algorithms are now widely used in various ways in lattice simulations. For example, APE smearing is used to smear the spatial gauge-field links in studies of glueballs, the static quark potential, etc. APE smearing is of particular interest in the construction of fat-link actions and of improved operators. Using the Wilson gauge action, Eq. (2.19), I have shown that to a good approximation, the two-dimensionful parameter space of the number of smearing sweeps $n_{\rm ape}(\alpha)$ and the smearing fraction α, may be reduced to a single dimension via the constraint

$$\frac{n_{\rm ape}(\alpha')}{n_{\rm ape}(\alpha)} = \frac{\alpha}{\alpha'}\,, \tag{11.1}$$

for α and α' in the range 0.3 to 0.7.

This result is in agreement with fat-link perturbation theory expectations, and survives for up to 200 sweeps over the lattice. This relation is expected to hold over the entire APE smearing range $0 < \alpha < 3/4$. I found the same relation for AUS smearing provided $\alpha \leq 0.85$. I have found that cooling, APE smearing and AUS smearing produce qualitatively similar smoothed gauge field configurations at $\beta = 6.0$, provided one calibrates the algorithms according to

$$n_{\rm c} \simeq 0.600\,\alpha\, n_{\rm ape}(\alpha)\,, \quad n_{\rm c} \simeq 0.618\,\alpha\, n_{\rm aus}(\alpha) \quad \text{and} \quad \alpha\, n_{\rm ape}(\alpha) \simeq 1.03\,\alpha'\, n_{\rm aus}(\alpha')\,. \tag{11.2}$$

I have used the topological charge analysis to confirm the action analysis results at $\beta = 6.0$ and to further support the above relations. Comparing the results with a lattice that has a coarser lattice spacing, $\beta = 5.7$, reveals that the evolution curves for the topological charge take different trajectories. At that coarser spacing with an unimproved action different smoothing algorithms will produce smoothed gauge field configurations with similar action, but with different topological properties.

The difficulty with standard Wilson action is that the $\mathcal{O}(a^2)$ errors in the action are large and tend to spoil instantons under the cooling procedure by reducing the action below the one instanton bound. For example, the action decreases monotonically to zero while the topological charge sectors will still transition between integer values as (anti)instantons are destroyed.

This analysis was then repeated for $\mathcal{O}(a^2)$ improved operators, Sec. 7.8. The analysis was done in the same manner as the one with the standard algorithm. To carry out the analysis I have introduced an improved version of the APE smearing algorithm founded on the connection between cooling Eq. (7.24) and the projection of the APE smeared link back to the $SU_c(3)$ gauge group via Eq. (7.38). This algorithm has the advantage of bringing the topological charge to integer value faster than standard APE smearing for $0.1 \le \alpha \le 0.5$. Furthermore, improved smearing preserves instantons inside the lattice for a larger period of time than standard APE smearing. This algorithm remains stable for any smearing fraction α in the range of $0.1 \le \alpha \le 0.5$, which is a slight decrease over the range available with standard APE smearing, $0.1 \le \alpha \le 0.75$. This is due to the extended nature of the "staple" in the improved smearing algorithm. Beyond $\alpha = 0.5$ the improved smearing algorithm becomes unstable.

Using the cooling algorithm described in Sec. 7.3, I have inserted the $\mathcal{O}(a^2)$-improved staple as explained in Sec. 7.4 to create an $\mathcal{O}(a^2)$-improved cooling algorithm. As a final tool I have created an $\mathcal{O}(a^2)$-improved topological charge operator. So, using the cooling algorithms and the algorithms described in Sec. 7.5.1 and in Sec. 7.6.2, I have cross calibrated the speed with which the algorithms remove action from the field configurations. In particular I have cross calibrated the smoothing rates of APE smearing at seven values of the smearing fraction; improved smearing at five values of the smearing fraction; cooling; and improved cooling. I explored smearing fractions in 0.1 intervals starting at $\alpha = 0.1$.

From this extended analysis I found that the results obtained in Sec. 7.7 with unimproved cooling, action and topological charge are consistent with those obtained using an $\mathcal{O}(a^2)$-improved cooling, action and topological charge, Sec. 7.8. I also found that it was possible to make qualitative comparisons between cooling and smearing algorithms. The comparisons were made on lattices as coarse as 0.165(2) fm and on a fine lattice where the lattice spacing is 0.077(1) fm.

I found that the relative smoothing rates can also be described via simple relationships for the $\mathcal{O}(a^2)$ case. I have summarized the results in Tables 7.8 and 7.9 for both the coarse $16^3 \times 32$ and fine $24^3 \times 36$ lattices, respectively. I discovered that a necessary correction to the APE smearing ratio rule, Eq. (11.1), had to be made when improved smearing is considered. These algorithms may be calibrated via

$$\frac{n_{\rm ape}(\alpha')}{n_{\rm ape}(\alpha)} = \frac{\alpha}{\alpha'} \qquad \text{and} \qquad \frac{n_{\rm Iape}(\alpha')}{n_{\rm Iape}(\alpha)} = \left(\frac{\alpha}{\alpha'}\right)^{\delta} , \tag{11.3}$$

for APE smearing and improved smearing respectively. I found $\delta = 0.914(1)$ without a significant dependence on the lattice spacing.

I also found that the calibration was only possible through the action and not through the the topological charge when improved operators were used. This is because the topological charge remains to a fixed integer for hundreds of sweeps while the action is monotonically decreasing, as we saw in Figs. 7.37. This makes the calibration using the topological charge only possible at the very early stages of the smoothing procedure, when the topological fluctuations are large.

However, the $\mathcal{O}(a^2)$-improved topological charge operator gave us some clear evidence that, in order to study the topology of pure gauge, one had to have lattices with a spacing finer than 0.165(2) fm. This argument is based on the fact that various algorithms lead the topological charge to completely different trajectories as reported in Figs. 7.35 and 7.36.

Moreover, as demonstrated in Sec. 7.8.1, small differences in the cooling algorithms can lead to different topological charge determinations, as illustrated in Figs. 7.27 and 7.28. These results indicate that the characteristic size of topological fluctuations in Yang-Mills gauge fields is at the scale of the coarse lattice spacing of 0.165(2) fm.

Now, when we compared the results obtained on the fine $24^3 \times 36$ lattices where $a = 0.077(1)$ fm, Sec. 7.8.3, we saw that on this lattice all algorithms considered were in excellent agreement. In this case it appears that the lattice spacing on these $\mathcal{O}(a^2)$-improved gauge field configurations is finer than the characteristic size of topological fluctuations, such that the gauge fields are already sufficiently smooth to unambiguously extract the topology of the gauge fields.

As a final comparison of the smoothing algorithms, I have provided a visual representation of a gauge field configuration after applying various smoothing algorithms. This demonstrates the quality of the calibration after applying the various smoothing algorithms. For the calibration results for the unimproved and improved case see Fig. 7.58 and in Fig. 7.26 respectively. The parameters used to generate these Figures can be found at the end of Secs. 7.7 and 7.8 respectively.

In Chapter 9 the gluon propagator was calculated on various lattices with $\mathcal{O}(a^2)$ improved gauge action. The infrared behaviour of the gluon propagator in the Landau gauge was clearly established. It was shown that the tree–level correction is a powerful tool that can be used to minimize the ultraviolet lattice artifacts and to reduce rotational symmetry breaking.

It was also shown in Sec. 9.5 that $\mathcal{O}(a^2)$ improved action produces superior results in comparison with those of a standard Wilson gauge action. One of the main improvements was that one is able to go to much coarser lattices while retaining good scaling.

In Fig. 9.10 we saw the comparison between a gluon propagator calculated on a $32^3 \times 64$ with standard gauge fields at $\beta = 6.00$ and an improved gauge field calculated on a $16^3 \times 32$ lattice at $\beta = 4.38$. The comparison showed that the turn–over was consistently reproduced. This was also the case as we made the lattice coarser. This produced robust results that demonstrated that the turn over of the propagator is infrared finite. There is, however, some scaling violation at large momentum for very coarse lattices. The increase of volume for the two largest volumes left the propagator unchanged, indicating that finite volume effects are small. The tree-level corrected results from the $\beta = 3.92$ ($a = 0.353$ fm) $16^3 \times 32$ lattice with a physical volume of $5.65^3 \times 11.30 = 2038$ fm^4 may be regarded as an excellent estimate of the infinite volume limit. An extrapolation of $D(0)$ via a linear ansatz inversely proportional to the physical lattice volume provides a reasonable fit. Moreover results from the largest volume lattice reside very close to the infinite volume limit.

In Chapter 10 I have presented the first study of the quark propagator using an overlap quark action. The study was based on a set of ten different quark masses. In this analysis we can see considerable improvement in the behaviour of the quark propagator when the Dirac Wilson operator is inserted into the overlap fermion formalism which produces an $\mathcal{O}(a)$ improved quark action with an exact chiral symmetry.

The overlap quark propagator did not require any tree–level correction unlike the Wilson fermion quark action or other actions resembling Wilson fermions. The action produced credible results for both functions $Z^{(\mathrm{R})}(q^2)$ and $M(q^2)$.

The quark condensate was then extracted using Eqs. (10.36) and (10.37). At

$\Lambda_{\rm QCD} = 0.234$ GeV with a fitting window of $p = [4,6]$ GeV we obtain at a renormalization point of $\zeta = 1$ GeV, a condensate of $\langle \overline{q}q \rangle_{\zeta=1\,GeV} = -(286\,{\rm MeV})^3$ and $\langle \overline{q}q \rangle_{\zeta=1\,GeV} = -(608\,{\rm MeV})^3$ for $q = [4,6]$ GeV. The discrepancy between the phenomenological value and the one extracted from the overlap simulation is due to the quenched approximation as discussed in Sec. 10.5 and is expected to go away in full QCD simulation.

This calculation was done at a relatively small volume with only one lattice. The objective of the calculation was to get some insight into the structure of the quark propagator in the overlap formalism. Further work is underway which will examine the volume dependence and scaling in this formalism.

Combining these three sections together, we have clear evidence that improved actions are very successful at bringing lattice results closer to the continuum, infinite volume limit while minimizing computational cost.

Appendix A

Gauge Group Representation and Conventions

A.1 $SU_c(2)$

The generators for the $SU_c(2)$ Lie group $t_i = \sigma_i/2$ where σ_i are the Pauli matrices:

$$\sigma_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \sigma_2 = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \sigma_3 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

A.2 $SU_c(3)$

This group is a topological group, and a simple Lie group. This Lie group has eight generators $t_a = \lambda_a/2$ $(a = 1,..,8)$, there is generator one for each gluon fields. These matrices are commonly known as the Gell–Mann matrices:

$$\lambda_1 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \lambda_2 = \begin{pmatrix} 0 & -i & 0 \\ i & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \lambda_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{pmatrix},$$

$$\lambda_4 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \lambda_5 = \begin{pmatrix} 0 & 0 & -i \\ 0 & 0 & 0 \\ i & 0 & 0 \end{pmatrix}, \lambda_6 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix},$$

$$\lambda_7 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -i \\ 0 & i & 0 \end{pmatrix}, \lambda_8 = \frac{1}{\sqrt{3}} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -2 \end{pmatrix}. \tag{A.1}$$

The matrices λ_a satisfy the following commutation relations (the Lie Algebra):

$$[t_a, t_b] = 4i f_{abc} t_c \quad \text{and} \quad \mathrm{Tr}\{\mathrm{t_a}, \mathrm{t_b}\} = \frac{1}{2}\delta_{\mathrm{ab}}, \tag{A.2}$$

where the constants f_{abc} are the structure constants of the Lie Algebra, these are fully antisymmetric with respect to interchange of indices. The independent constants which are different from zero have the following values

$$f_{123} = 1\,, f_{147} = f_{246} = f_{345} = f_{516} = f_{257} = f_{637} = \frac{1}{2}\,, f_{458} = f_{678} = \frac{\sqrt{3}}{2}. \tag{A.3}$$

A.3 The Spinor matrices

The Euclidean hermitian γ matrices satisfy

$$\gamma_\mu = \gamma_\mu^\dagger \quad \text{and} \quad \{\gamma_\mu, \gamma_\nu\} = 2\delta_{\mu\nu}. \tag{A.4}$$

The representation used in this thesis

$$\gamma = \begin{pmatrix} 0 & i\sigma \\ -i\sigma & 0 \end{pmatrix}, \gamma_4 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \gamma_5 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

The Bjorken and Drell Representation:

$$i\gamma_{\mathrm{BD}}^i = \gamma_i \quad , \quad \gamma_{\mathrm{BD}}^0 = \gamma_4 \quad \text{and} \quad \gamma_{\mathrm{BD}}^5 = \gamma_5. \tag{A.5}$$

The matrices γ_1 and γ_3 are purely imaginary and γ_2, γ_4 and γ_5 are real.

Appendix B

Mathematical Background and Definitions

B.1 Mathematical Background

Topology is the mathematical study of the properties of the space and the sets of functions connecting spaces. In general, topology is not only concerned with finite dimensional spaces like subspaces of $\mathbf{R^n}$, but also in infinite dimensional ones such as the spaces that we find in quantum field theory. Topology is concerned with the space of maps between two topological spaces, and how this space of maps is connected.

Two maps $f_0 : X \longrightarrow Y$ and $f_1 : X \longrightarrow Y$ belong to the same space of maps if and only if they can be continuously deformed into one another, that is if there is a continuous family of maps, indexed by a parameter $t \in [0, 1]$ that connects f_0 and f_1. The collection of maps, f_t can be looked at as a single map $F(x, t)$ from $X \in [0, 1]$ into Y such that $f_t(x) = F(x, t)$. The map F must be continuous and its restriction to $X \times \{0\}$ and $X \times \{1\}$ is specified by the condition $F(x, 0) = f_0(x)$ and $F(x, 1) = f_1(x)$. The map F or the family of maps f_t is referred to as a homotopy between f_0 and f_1, and we say that f_0 and f_1 are homotopic if there is a homotopy between them.

In this way, all the maps from X to Y can be grouped into homotopy classes each class consisting of maps homotopic to one another.

A simple example of this would be the mapping of the circle S^1 on a circle S^1 that brings the south pole x_0 of the circle S^1 into a fixed point y_0 of the space S^1. If a point on the circle is specified by the polar angle ϕ then each mapping of the circle on the circle can be described by a function $f(\phi)$ satisfying:

$$f(2\pi) = f(0) + 2\pi n,$$

where n is an integer. This number indicates how many times the circle is wound around itself and is referred to as the winding number. All the mapping with the same winding number can be continuously deformed into one another, hence homotopic.

If X is a single point then the homotopy classes of maps $X \longrightarrow Y$ are in one–to–one correspondence with the connected components of Y. In fact the space of maps between the two topological spaces X and Y is homeomorphic (i.e. topologically equivalent) to Y. On the other hand a map $X \longrightarrow Y$ is called null–homotopic (see Def. B.9), or homotopically trivial, if it is homotopic to a map that takes all of X to a single point of Y (a constant map). If Y is connected, all null–homotopic maps are homotopic to one another, in that case the homotopy class of these maps is called the trivial class or zero class.

One of the most fundamental topological spaces, is the n–sphere, S^n. The n–sphere can be thought of as a topological subspace of $\mathbf{R^{n+1}}$. For example S^1 lives in a two dimensional plane, namely $\mathbf{R^2}$. The map of S^n through the homotopy group onto an arbitrary topological space is one of the deepest concepts in topology. A special case of this, is the event that if every map from a k–sphere into a space X is null homotopic then one says that the topological space X is aspherical (see Def. B.10) in dimension k.

The set of homotopy classes form an Abelian group, some of which are finite and others are infinite. The group is commonly denoted as $\pi_k(X)$. The subscript indicates the dimension of the spatial sphere subjected to the mapping onto which topological space the mapping is performed (indicated in the parenthesis). When $k = 1$, $\pi_1(X)$ is called the fundamental group. We know that a path in a space X, is a map from the interval $[0, 1]$ to X, we can equally think of a path as a continuous parameterized curve in X. The space is connected if any two of its points can be joined by a path. If the path starts and end at the same point $x_0 = x_1$, then the path is closed forming a loop, with x_0 being the basepoint. Such a loop is null–homotopic if it is homotopic to the trivial loop with image x_0. If we now consider all the loops in the space X, whose basepoint is some fixed point x_0, then the set of such loops can be divided into homotopy classes with all loops in the same class homotopic to one another. This partitioning into classes is possible because homotopy is an equivalence relation (i.e. when f_0 is homotopic to f_1 implies that f_1 is homotopic to f_0 which makes it symmetric, it is also transitive because if two loops are homotopic to a third then they are homotopic to one another). The set of homotopy classes of loops in X with basepoint x_0 is the fundamental group.

In general the fundamental group applies to simple topological system like magnetic monopoles in $U(1)$ theories, but in the case of more complicated gauge theories like QCD one has to go into higher dimensional homotopy groups.

Let's now consider a topological space E with a fixed basepoint e_0, then a k–dimensional spheroid of E is a map $S^k \longrightarrow E$ that takes the south pole s of S^k to $e_0 \in E$, the fixed basepoint of E. A null spheroid happens when the map $S^k \longrightarrow E$ is the constant map whose image is the fixed basepoint e_0 and when a spheroid is homotopic to the null spheroid, it is then null homotopic. Furthermore, two spheroids f_0 and f_1 are homotopic if there is a homotopy f_t between the two maps such that $f_t(s) = e_0$, for all t. Note that f_0 and f_1 are homotopic as maps but not as spheroids when the homotopy between f_0 and f_1 cannot be chosen in such a way that s is mapped to e_0 all the time. Homotopic spheroids also form an equivalence relation, which means that in this case too, we get a partition of the space into homotopy classes of spheroids and within each class all spheroids are homotopic to one another. The set $\pi_k(E, e_0)$ can be given a group structure for $k \geq 1$. The resulting group is called the k^{th}–homotopy group of E.

The calculation of these groups is in general non–trivial, the task is actually made easier by creating a fibration of the total topological space. The technique represent a powerful tool for the study of non–trivial topological spaces. A fibration can be thought of as a partitioned domain in the total space and it is defined as follows. Given a map p from a space E onto a space B, we can partition the domain into disjoint sets $F_b = p^{-1}(b)$ for $b \in B$. We then say that p defines a fibration if all sets F_b are homeomorphic (see Def. B.6) to one another. In this case F_b is called the fiber over b. The space B is called the base space of the fibration, E the total space and p is the projection map. The fibers of a fibration are usually contained in a space that we may call the fiber space F. So a fibration with total space E, base space B and fiber

F via a projection p is denoted by (E, B, F).

Let's look at simple examples. The orthogonal projection of a solid cylinder onto the base space defines a fibration. The total space E is the solid cylinder, the base space B is the disk and the fiber space F is homeomorphic to an interval. Another example is a k–dimensional submanifold Ω of $\mathbf{R}^n$. For a point $x \in \Omega$, let $T_x \subset \mathbf{R}^n$ be the space of tangent vectors to Ω at x. Let $T\Omega$ be the space of pairs (x, ζ) where $x \in \Omega$ and $\zeta \in T_x$. Each such pairs can be thought of as a tangent vector based at x. There is a natural projection $p : T\Omega \longrightarrow \Omega$ assigning to (x, ζ) the point $x \in \Omega$. Since $p^{-1}(x)$ can be regarded as a tangent space T_x and since T_x is homeomorphic to $\mathbf{R}^k$, we can see that p gives a fibration with base Ω and fiber $\mathbf{R}^k$. This is called the tangent fibration to Ω. A variation of this construction can be done by replacing T_x by the set of unit tangent vectors at x which gives the unit tangent fibration with fibers homeomorphic to S^{k-1}.

Now consider a topological group (see Def. B.12) G acting on a space E by transformations ϕ_g for $g \in G$. There is then a partition of E into subsets, the orbit (see Def. B.16) of G. If $e \in E$ is a continuous map, α, can be constructed from G onto the orbit of e by setting $\alpha(g) = \phi_g(e)$. This map is one–to–one if and only if the stabilizer (see Def. B.17) H_e is trivial, since H_e is the subgroup of G that takes e to itself. If, in addition G is compact α is a homomorphism (see Def. B.13) since every one–to–one map from a compact space onto a Hausdorff space (see Def. B.11) is a homeomorphism. It can be concluded that when a compact group G acts freely (an action, Def. B.15, is free when all stabilizers are trivial) on a space E, all orbits are homeomorphic to G.

It then follows, the decomposition of E into orbits give rise to a fibration. Fibration that arise in this way are called principal. Thus a principal fibration (E, B, G) is one in which the model fiber G is a topological group and there is a free action of G on E whose orbits are the fibers. Using the Stiefel manifold [43], it can be shown that a principal fibration for $SO(n)$ (the group of orthogonal matrices with unit determinant) is given by:

$$(SO(n),\, S^{n-1},\, SO(n-1)). \tag{B.1}$$

A similar principal fibration can be written down for $SU_c(N)$, so we have

$$(SU_c(N),\, S^{2n-1},\, SU_c(N-1)), \tag{B.2}$$

so when the total space is $SU_c(N)$ the base space is the $(2n-1)$–sphere with fibers in $SU_c(N-1)$.

Furthermore the spaces forming the principal fibration may be related to the homotopy groups when every spheroid f of E can be written in the form (f_1, f_2) where f_1 is a spheroid of the base space, B, and f_2 is a spheroid of the fiber space F. A continuous deformation in f gives continuous deformation in f_1 and f_2. Thus to each element of $\alpha \in \pi_k(E)$ we can associate a pair $(\alpha_1, \alpha_2) \in \pi_k(B) \oplus \pi_k(F)$. Hence, we have the following result, if $E = B \times F$ is a direct product of the base and fiber space then the k^{th}–homotopy of the total space E is isomorphic (see Def. B.14) to the direct sum of the k^{th}–homotopy of the base space $\pi_k(B)$ and the k^{th}–homotopy of the fiber space $\pi_k(F)$, i.e.,

$$\text{if} \quad E = B \times F \quad \Rightarrow \quad \pi_k(E) \simeq \pi_k(B) \oplus \pi_k(F). \tag{B.3}$$

Similarly, if the base space B of a fibration (E, B, F, p) is aspherical in dimension k and $k+1$ then the groups $\pi_k(E)$ and $\pi_k(F)$ are isomorphic, that is,

$$\text{if} \quad \pi_k(B) \simeq \pi_{k+1}(B) \simeq 0 \quad \Rightarrow \quad \pi_k(E) \simeq \pi_k(F). \tag{B.4}$$

If the fiber space F is aspherical in dimension $k-1$ and k then $\pi_k(E)$ and $\pi_k(B)$ are isomorphic. That is, we have the result that

$$\text{if} \quad \pi_k(F) \simeq \pi_{k-1}(F) \simeq 0 \quad \Rightarrow \quad \pi_k(E) \simeq \pi_k(B), \tag{B.5}$$

and

$$\text{if} \quad \pi_{k-1}(E) \simeq \pi_k(E) \simeq 0 \quad \Rightarrow \quad \pi_k(B) \simeq \pi_{k-1}(F). \tag{B.6}$$

Using these relation it is now possible to calculate the homotopy groups for various topological groups that are of physical interest. The classification of the sphere is given in Appendix. B.3. We will use this classification to calculate the homotopy groups of other topological groups.

For example, in the case of $SO(2)$ which is isomorphic to $U(1)$ and homeomorphic to S^1 we get

$$\pi_k(SO(2)) \simeq \pi_k(U(1)) \simeq \pi_k(S^1) \simeq 0 \quad \text{for}\, k \geq 2.$$

Now for $SO(4)$, which has a homomorphic twofold cover, that is $SU_c(2) \times SU_c(2) \longrightarrow SO(4)$, using Eq. (B.3), the homotopy group for $k \geq 2$ becomes:

$$\pi_k(SO(4)) \simeq \pi_k(SU_c(2) \times SU_c(2)) \simeq \pi_k(SU_c(2)) \oplus \pi_k(SU_c(2)),$$

but since $SU_c(2)$ is homeomorphic to S^3 and that $\pi_k(S^n) \simeq 0$ for $k < n$ and $\pi_k(S^n) \simeq \mathbf{Z}$ for $k = n$, we have

$$\pi_1(SO(4)) \simeq \pi_2(SO(4)) \simeq 0 \quad \text{and} \quad \pi_3(SO(4)) \simeq \mathbf{Z} \oplus \mathbf{Z}. \tag{B.7}$$

Using the principal fibration defined in Eq. (B.1) and the fact $\pi_k(S^{n-1}) \simeq 0$ for $k \leq n-2$, we have

$$\pi_k(SO(n-1)) \simeq \pi_k(SO(n)) \quad \text{for } k < n-2.$$

The calculation of the homotopy group for $SU_c(N)$ is done using the principal fibration defined in Eq. (B.2). The base space is then S^{2n-1} and the fiber space is $SU_c(N-1)$. The k^{th} and $(k+1)^{th}$–homotopy group of the base space are aspherical for $k \leq 2n-2$, we therefore have

$$\pi_k(SU_c(N-1)) \simeq \pi_k(SU_c(N)) \quad \text{for } k < 2n-2.$$

In the case of $SU_c(3)$, the fibration becomes $(SU_c(3), S^5, SU_c(2))$. Following the above reasoning, we know that for $k < 4$, S^5 is aspherical in dimension k and $k+1$ i.e.

$$\pi_k(S^5) \simeq \pi_{k+1}(S^5) \simeq 0 \quad \text{for } k < 4.$$

It then implies that the k^{th}–homotopy of the total and fiber spaces are isomorphic and homeomorphic to S^3:

$$\pi_k(SU_c(3)) \simeq \pi_k(SU_c(2)) \simeq \pi_k(S^3) \quad \text{for } k < 4,$$

which gives:

$$\pi_k(SU(3)) \simeq \pi_k(SU(2)) \simeq \pi_k(S^3) \simeq \left\{ \begin{array}{ll} \mathbf{Z} & \text{for } k = 3 \\ 0 & \text{for } k \leq 2. \end{array} \right. \tag{B.8}$$

This shows that the set of homotopy classes of maps $S^3 \longrightarrow SU(3)$ is in one–to–one correspondence with the integers.

$$\{S^3, SU(3)\} \simeq \pi_3(SU(3)) \simeq \mathbf{Z}. \tag{B.9}$$

It is now clear that the number of times the 3–sphere is mapped onto $SU_c(3)$ is an integer and that it is characterised by the topological number or the topological charge. The topological charge is really the degree of the map.

To really see how the true nature of the topological charge arises, one must go into the homology groups of k–dimensional surfaces with vanishing boundaries. These types of surfaces are called k–cycles. The Homology group and Homotopic group are related through a homomorphism of the pull back map, recall that if $f : M \longrightarrow \tilde{M}$ is any smooth map between smooth manifolds and x is a point in M, a k–dimensional tensor with lower indices at the point $f(x) \in \tilde{M}$ can be pulled back to a tensor at the point $x \in M$. This means that the pullback maps under homotopic maps are homologous. A similar statement for cohomology groups can be made when one is looking at differential forms of degree k, or simply k–forms.

The homomorphism in homology is closely connected with the homomorphism in cohomology through Stoke's theorem:

$$\int_\Gamma f^\star w = \int_{f(\Gamma)} w, \tag{B.10}$$

where w is any k–forms on a smooth manifold $\tilde{M}$ and Γ is any k–cycle in M. This connection between homology and homotopy can be used to assign to maps certain homotopy invariants, that is the number defined by Eq. (B.10) does not change under continuous deformation of the map f. The Hurewicz isomorphism theorem [43] tells us that: *if $k \geq 2$ and the total space E is aspherical in dimensions less than k, the Hurewicz homomorphism of the k^{th}–homotopy of the space E, $\pi_k(E)$, and the k^{th}–homology $H_k(E, \mathbf{Z})$ are isomorphic.* Hence we have $\pi_3(SU_c(3)) \simeq H_3(SU_c(3), \mathbf{Z})$. Then Eq. (B.10) is just equal to an integer, the topological number or topological charge.

B.2 Definitions

Definition B.1 (Metric spaces): *A metric space is a set E together with a metric. A metric is a function that assigns to each pair of points (x, y) a distance $\rho(x, y)$, satisfying the following conditions:*

1. *$\rho(x, y) \geq 0$ for $x, y \in E$, with equality iff $x = y$;*
2. *$\rho(x, y) = \rho(y, x)$ for $x, y \in E$; and*
3. *$\rho(x, z) \leq \rho(x, y) + \rho(y, z)$ for $x, y, z \in E$.*

Definition B.2 (A topology): *A topology τ for a space X is a collection of subsets of X that has the following properties:*

1. *$0 \in \tau$ and $X \in \tau$;*
2. *if $U_\alpha \in \tau$ for each $\alpha \in \mathcal{A}$ then $\bigcup_{\alpha \in \mathcal{A}} U_\alpha \in \tau$;*

3. if $\{U_1, .., U_k\} \in \tau$ then $\bigcup_{i=1}^{k} U_i \in \tau$.

Definition B.3 (A topological space): *The pair (X, τ) consisting of a space X and a topology τ for X is a topological space.*

Definition B.4 (Open cover): *An open cover of E is a family $\{U_\alpha\}$ of open subsets of E whose union is all of E. Compactness means that we can choose finitely many indices $\alpha_1, .., \alpha_k$ such that $\{U_{\alpha_1}, .., U_{\alpha_n}\}$ is still a cover for E.*

Definition B.5 (Compactness): *A topological space E is compact if every open cover of E has a finite subcover.*

Definition B.6 (Homeomorphic): *A continuous, one to one map h of X onto Y for which $h^{-1} : Y \longrightarrow X$ is also continuous is called a homeomorphism.*

Definition B.7 (Disconnected): *A space X is said to be disconnected if X can be written as $X = H \cup K$, where H and K are disjoint non–empty open sets in X, the pair $\{H, K\}$ is then called a disconnection of X.*

Definition B.8 (Connected): *if a space X has no disconnection then it is connected.*

Definition B.9 (Null–homotopic): *A map $X \longrightarrow Y$ is called null–homotopic or homotopically trivial if it is homotopic to a map that takes all of X to a single point of Y (i.e. a constant map).*

Definition B.10 (Aspherical in dimension k): *if every map from the k–sphere, S^k, into a space X is null homotopic, then the space X is aspherical in dimension k.*

Definition B.11 (Hausdorff spaces): *A topological space is said to be Hausdorff if whenever x and y are distinct points of x there exists open sets U_x and U_y in X with $x \in U_x$ and $y \in U_y$ and $U_x \cap U_y = \emptyset$ (distinct points can be separated by disjoint open sets).*

Definition B.12 (Topological groups): *A topological group is a Hausdorff topological space G that is also a group in which the operations of multiplication*

$$(x, y) \longrightarrow xy : \quad G \times G \longrightarrow G, \tag{B.11}$$

and inversion

$$x \longrightarrow x^{-1} : \quad G \longrightarrow G \tag{B.12}$$

are continuous. A subgroup of a topological group is also a topological group.

Definition B.13 (Homomorphism): *A map $\phi : G \longrightarrow G'$ from one topological group to another is a homomorphism if it preserves multiplication, i.e.*

$$\text{if} \quad \phi(\text{ab}) = \phi(\text{a})\phi(\text{b}). \tag{B.13}$$

Definition B.14 (Isomorphism): *An isomorphism is a homomorphism that is one–to–one and onto. If the isomorphism of the group is onto itself then it is an automorphism.*

Definition B.15 (A right action): *a right action of a topological group, G, on a topological space, Y, is a continuous map $\sigma : Y \times G \longrightarrow Y$ which satisfies:*

1. $\sigma(y, e) = y \quad \forall\, y \in Y$ *(e is the identity in G)*
2. $\sigma(y, g_1 g_2) = \sigma(\sigma(y, g_1), g_2) \quad \forall\, y \in Y$ *and* $\forall\, g_1, g_2 \in G$.

Definition B.16 (Orbit): *given any point in a topological space, $y \in Y$, the orbit of y under the action σ is the subset $\{y \cdot g : g \in G\}$ of Y.*

Definition B.17 (Stabilizer): *The stabilizer of H_x of a point in $x \in H$ is the set of elements of G that leave x fixed that is $h \in H_x$ if $\phi_h(x) = x$, H_x is a subgroup of G.*

B.3 Classification of the Sphere

$$\begin{array}{rcll}
\pi_1(S^1) & \simeq & \mathbf{Z} & \\
\pi_k(S^1) & \simeq & 0 & \text{for}\, k \geq 2 \\
\pi_1(S^2) & \simeq & 0 & \\
\pi_2(S^2) \simeq \pi_3(S^2) & \simeq & \mathbf{Z} & \\
\pi_4(S^2) \simeq \pi_5(S^2) & \simeq & \mathbf{Z_2} & \\
\pi_6(S^2) & \simeq & \mathbf{Z_{12}} & \\
\pi_1(S^3) \simeq \pi_2(S^3) & \simeq & 0 & \\
\pi_k(S^3) & \simeq & \pi_k(S^2) & \text{for}\, k \geq 3 \\
\pi_k(S^n) & \simeq & 0 & \text{for}\, k < n \\
\pi_n(S^n) & \simeq & \mathbf{Z} & \\
\pi_{n+1}(S^n) \simeq \pi_{n+2}(S^n) & \simeq & \mathbf{Z_2} & \text{for}\, n \geq 3 \\
\pi_{n+3}(S^n) & \simeq & \mathbf{Z_{24}} & \text{for}\, n \geq 5 \\
\pi_{n+4}(S^n) & \simeq & 0 & \text{for}\, n \geq 6 \\
\pi_{n+5}(S^n) & \simeq & 0 & \text{for}\, n \geq 7 \\
\pi_m(S^n) & \simeq & \pi_{m+1}(S^{n+1}) & \text{for}\, m < 2n-1 \\
\pi_{4n-1}(S^{2n}) & \simeq & \mathbf{Z} + \text{finite group.} &
\end{array}$$

All groups $\pi_m(S^n)$, except for $\pi_n(S^n)$ and $\pi_{4n-1}(S^{2n})$, are finite.

Appendix C

Markov Chain Terminology

In this section I explain the terminology used in Section 3.1. These definition may be found for example in Ref. [5].

1. A chain is called **irreducible** if, starting from an arbitrary configuration there exists a finite probability of reaching any other configuration C_j after a finite number of Markov steps. In other words there exist a finite N such that

$$P_{ij}^{(N)} = \sum_{i_k} P_{ii_1} P_{i_1 i_2} \ldots P_{i_{N-1} j} \neq 0. \tag{C.1}$$

2. A Markov chain is called **aperiodic** if $P_{ii}^{N} \neq 0$ for any N.

3. A state is called **positive** if its mean recurrence time is finite. If $P_{ii}^{(n)}$ is the transition probability to get from C_i to C_i in n–steps of the Markov chain, without reaching this configuration at any intermediate step, then the mean recurrence time of C_i is given by

$$\tau_i = \sum_{n=1}^{\infty} n \, P_{ii}^{(n)}. \tag{C.2}$$

Appendix D

Generating Uniformly Distributed Random Numbers

When measuring an observable quantity in Monte Carlo simulations, one must generate a sequence of random numbers which are uniformly distributed on the group manifold. This could easily be done by considering the unit n–sphere, S^n, which is a topological subspace of $\mathbf{R}^{n+1}$, to select points that are inside the n–sphere according to $\sum_{i=1}^{n+1} x_i \leq 1$. In $2D$ the situation is depicted in Fig.(D.1).

This method is easy to implement, but the probability of getting points selected inside S^n with a uniform distribution decreases as the number of dimensions, n, increases. For example, in $2D$ the probability of getting points inside S^1, $P_{\rm in}$, is approximately 75 percent of the time. In $3D$, $P_{\rm in} = 4\pi/24 \approx 50$ percent, and in $4D$ the probability of getting points inside S^3 is roughly $P_{\rm in} = 4\pi^2/32$, about 30 percent. Moreover the points are not uniformly distributed.

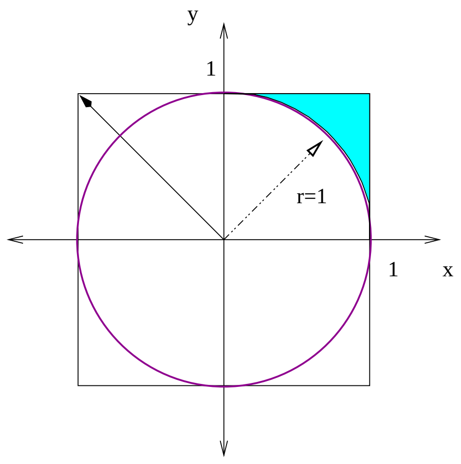


Figure D.1: The unit circle in $2D$, $r = x^2 + y^2 = 1$.

D.1 Random Numbers with a Gaussian Probability Distribution Function

An alternative approach [19] is generating random numbers with a Gaussian probability distribution function. In $\mathbf{R}^1$ such functions take the analytic form of $P(x) = exp(-x^2)$

and in $\mathbf{R}^{n+1}$ we have

$$\prod_{i=1}^{n+1} P(x_i) = \exp(-\sum_{i=1}^{n+1} x_i^2) = \exp(-r^2), \tag{D.1}$$

where $r^2 = 1$, since we are working on the unit sphere. To bring the points on the surface of the unit sphere one just needs to divide by the norm of the vector space, i.e. $(\sum_{i=1}^{n+1} x_i^2)^{1/2}$. These numbers can be used construct a randomly uniformly distributed vector in $\mathbf{R}^2$. Let's consider 1 random number r that is uniformly distributed on $[0,1]$, and another uniformly distributed number $\theta \in [0, 2\pi]$. To construct our uniformly distributed vector of random numbers in $\mathbf{R}^2$ with Gaussian probability distribution, one need to consider

$$r \in [0,1] \implies \ln(r) \in (-\infty, 0] \Longrightarrow \sqrt{-2\ln(r)} \in [0, \infty). \tag{D.2}$$

To make it uniformly distributed onto the surface of the sphere set

$$a_0' = \sqrt{-2\ln(r)} cos(\theta) \qquad \text{and} \qquad a_1' = \sqrt{-2\ln(r)} sin(\theta), \tag{D.3}$$

and divide by its norm. One can now build a vector random vector in $\mathbf{R}^4$ by just considering two sets of the 2 dimensional vectors with $r_1, r_2 \in [0,1]$ and $\theta_1, \theta_2 \in [0, 2\pi]$ and combine them together as such:

$$\begin{aligned} a_\mu \;=\; & \left[\ln \frac{1}{(r_1 r_2)^2}\right]^{\frac{1}{2}} \times \\ & \Big(\sqrt{-2\ln(r_1)} cos\theta_1, \sqrt{-2\ln(r_1)} sin\theta_1, \\ & \sqrt{-2\ln(r_2)} cos\theta_2, \sqrt{-2\ln(r_2)} sin\theta_2\Big). \end{aligned} \tag{D.4}$$

In $\mathbf{R}^{n+1}$, one just needs to consider $n/2$ sets, when $n+1$ is odd just keep $n/2 - 1$ and half of the next set.

Appendix E

Computer Codes to Generate Gauge Fields Configurations

E.1 Masking Routine for the Wilson Action

```
c
c     cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     MaskWilson subroutine establishes the x y z and t masks.
c     This mask is only to be used for the use of ordinary plaquette action
c
c     Author: Frederic Bonnet  fbonnet@physics.adelaide.edu.au
c          Updated 14 Sept. 2000  by DBL
c
      subroutine MaskWilson(mask)

      implicit none
      include 'latticeSize.h'
      integer,parameter                                       :: mu=4
      integer,parameter                                       :: nmask=16
      logical,dimension(nx,ny,nz,nt,mu,nmask)                 :: mask
cmf$  layout mask(:news,:news,:news,:news,:serial,:serial)

c     local variables

      integer                                                 :: ix,iy,iz,it

c     start of the execution commands

      mask = .false.
c
c     Our first idea
c
c      forall( ix=1:nx,iy=1:ny,iz=1:nz,it=1:nt,mod(iy+iz+it,2) .eq. 0 )
c     &     mask(ix,iy,iz,it,1,1)=.true.
c      forall( ix=1:nx,iy=1:ny,iz=1:nz,it=1:nt,mod(ix+iz+it,2) .eq. 0 )
c     &     mask(ix,iy,iz,it,2,1)=.true.
c      forall( ix=1:nx,iy=1:ny,iz=1:nz,it=1:nt,mod(ix+iy+it,2) .eq. 0 )
c     &     mask(ix,iy,iz,it,3,1)=.true.
c      forall( ix=1:nx,iy=1:ny,iz=1:nz,it=1:nt,mod(ix+iy+iz,2) .eq. 0 )
```

```
c     &     mask(ix,iy,iz,it,4,1)=.true.
c
c     Checker Board this time
c
      forall( ix=1:nx,iy=1:ny,iz=1:nz,it=1:nt,mod(ix+iy+iz+it,2) .eq. 0 )
     &     mask(ix,iy,iz,it,1,1)=.true.
      forall( ix=1:nx,iy=1:ny,iz=1:nz,it=1:nt,mod(ix+iy+iz+it,2) .eq. 0 )
     &     mask(ix,iy,iz,it,2,1)=.true.
      forall( ix=1:nx,iy=1:ny,iz=1:nz,it=1:nt,mod(ix+iy+iz+it,2) .eq. 0 )
     &     mask(ix,iy,iz,it,3,1)=.true.
      forall( ix=1:nx,iy=1:ny,iz=1:nz,it=1:nt,mod(ix+iy+iz+it,2) .eq. 0 )
     &     mask(ix,iy,iz,it,4,1)=.true.

      mask(:,:,:,:,:,2) = .not. mask(:,:,:,:,:,1)

      return
      end subroutine MaskWilson
```

E.2 Generating Random $SU_c(2)$ Matrices with a Heat–Bath Method

```
c
c     ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Author: Frederic D.R. Bonnet & D.B. Leinweber; date: 28th of Jully 1998.
c     subroutine that implements the pseudo-heatbath algorithm
c
c
      subroutine pseudoheat(urnewsu2,uinewsu2,phbsr,phbsi,mask,imask,beta,ihat)
      implicit none
      include 'latticeSize.h'

c     global variables

      integer,parameter                                      :: ncsu2=2
      integer,parameter                                      :: nsigma=ncsu2*ncsu2
      integer,parameter                                      :: mu=4
      integer,parameter                                      :: nmask=16

      integer                                                :: ihat,imask
      double precision                                       :: beta

      double precision,dimension(nx,ny,nz,nt,ncsu2,ncsu2)    :: urnewsu2,uinewsu2
      double precision,dimension(nx,ny,nz,nt,ncsu2,ncsu2)    :: phbsr,phbsi
cmf$  layout urnewsu2(:news,:news,:news,:news,:serial,:serial)
cmf$  layout uinewsu2(:news,:news,:news,:news,:serial,:serial)
cmf$  layout phbsr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout phbsi(:news,:news,:news,:news,:serial,:serial)
      logical,dimension(nx,ny,nz,nt,mu,nmask)                :: mask
cmf$  layout mask(:news,:news,:news,:news,:serial,:serial)
```

```
c     local variables

      integer,parameter                                       :: nr=2
      double precision,parameter                              :: pi=3.141592653d0
      double precision,dimension(nx,ny,nz,nt)                 :: k,x,probrjk,random
cmf$  layout k(:news,:news,:news,:news)
cmf$  layout x(:news,:news,:news,:news)
cmf$  layout probrjk(:news,:news,:news,:news)
cmf$  layout random(:news,:news,:news,:news)
      double precision,dimension(nx,ny,nz,nt)                 :: norm
cmf$  layout norm(:news,:news,:news,:news)
      logical,dimension(nx,ny,nz,nt)                          :: update
cmf$  layout update(:news,:news,:news,:news)
      double precision,dimension(nx,ny,nz,nt,nr)              :: r,theta
cmf$  layout r(:news,:news,:news,:news,:serial)
cmf$  layout theta(:news,:news,:news,:news,:serial)
      double precision,dimension(nx,ny,nz,nt,nsigma)          :: a4vector
cmf$  layout a4vector(:news,:news,:news,:news,:serial)
      double precision,dimension(nx,ny,nz,nt,ncsu2,ncsu2)     :: urprmsu2,uiprmsu2
cmf$  layout urprmsu2(:news,:news,:news,:news,:serial,:serial)
cmf$  layout uiprmsu2(:news,:news,:news,:news,:serial,:serial)

      integer                                                 :: imu,isigma,ic,jc,kc
      integer                                                 :: counter

c     start of the execution commands

      CALL CMF_random( r )
      where( r==0.0d0 ) r = 1.0d0
      r = sqrt( -2.0d0 * log(r) )
      CALL CMF_random( theta )
      theta = 2.0d0 * pi * theta

      a4vector(:,:,:,:,1) = r(:,:,:,:,1) * cos( theta(:,:,:,:,1) )
      a4vector(:,:,:,:,2) = r(:,:,:,:,1) * sin( theta(:,:,:,:,1) )
      a4vector(:,:,:,:,3) = r(:,:,:,:,2) * cos( theta(:,:,:,:,2) )
      a4vector(:,:,:,:,4) = 0.0d0

c     calculates sqrt(a1^2+a2^2+a3^2+a4^2)=norm

      norm = sqrt( sum( a4vector**2,dim=5 ) )

c     calculates the determinant k=|\sum_{\alpha=1}^6\widetilde{U}_{\alpha}|^{1/2}
c     where $\widetilde{U}_{\alpha}\equiv$ the six product of the three links
c     variable which interact with the link in question, i.e. stapler and staplei

        k = sqrt( abs( phbsr(:,:,:,:,1,1) * phbsr(:,:,:,:,2,2) -
     &                 phbsr(:,:,:,:,1,2) * phbsr(:,:,:,:,2,1) -
     &                 phbsi(:,:,:,:,1,1) * phbsi(:,:,:,:,2,2) +
     &                 phbsi(:,:,:,:,1,2) * phbsi(:,:,:,:,2,1) ) )

c     the values of update is all true when imask=1 and all false when imask=2
```

```
c     the value of imask is passed in in the subroutine list and defined in the
c     subroutine pseudosweep.

      update = mask(:,:,:,:,ihat,imask)
      counter = 0

c     the do while makes sure that every links are updated on every sweep.
c     an array dimensioned according to the volume of the lattice is reshaped
c     and store in the variable x. x is a random double precision number,
c     uniformely distributed within the region exp(-2*beta*k)<x<1.
c     This generates a4vector distributed with exponential weight
c     exp(beta*k*a4vector).
c     To summarize, the algorithm begins with a trial a4vector=1+ln(x)/(beta*k)
c     where x is a random number uniformely distributed in the region
c     exp(-2*beta*k)<x<1. This generates a4vector distributed with exponential weight
c     exp(beta*k*a4vector). To correct for the factor (1-a4vector**2)^1/2 in
c     P(a4vector)~(1-a4vector**2)^1/2*exp(beta*k*a4vector), reject this a4vector
c     with probability 1.0d0 - sqrt( 1.0d0 - a4vector**2 ) and select a new trial
c     a4vector, repeat this until an a4vector is accepted.

c     update starts to be true if probrjk is false then we have a true with a
c     false which is false then update = ( update .and. probrjk >= random ) becomes
c     true with a true which is true then we go around the while loop again. This
c     time update is still true but probrjk < random is true then the where
c     statement is true and a4vector=x (the trial). This means that
c     update = ( update .and. probrjk >= random )
c     becomes true with false which is false, therefore we exit the while loop.

      do while( any(update) )
         counter = counter + 1
         call CMF_random( x )
         x = ( 1.0d0 - exp( -2.0d0 * beta * k ) ) * x + exp( -2.0d0 * beta * k )
         x = 1.0d0 + log( x ) / ( beta * k )
         probrjk = 1.0d0 - sqrt( 1.0d0 - x**2 )
         call CMF_random( random )
         where( update .and. probrjk < random ) a4vector(:,:,:,:,4) = x(:,:,:,:)
         update = ( update .and. probrjk >= random )
      end do

      do isigma=1,nsigma-1
        a4vector(:,:,:,:,isigma) =
     &   ( a4vector(:,:,:,:,isigma) * sqrt( 1.0d0 - a4vector(:,:,:,:,4)**2 ) )
     &   / norm(:,:,:,:)
      end do

c     this calculates the link variable U=a4vector(4)*I+ia4vector(1to3)*sigma
c     as in su2random, except here we are only interested in one specific
c     direction ihat, the value of ihat is defined according to the loop over
c     the 4 directions imu=1,mu in the subroutine pseudosweep.

c     converting a4vector to an su2 matrix
```

```
      urprmsu2(:,:,:,:,1,1) = a4vector(:,:,:,:,4)
      urprmsu2(:,:,:,:,2,2) = a4vector(:,:,:,:,4)
      urprmsu2(:,:,:,:,1,2) = a4vector(:,:,:,:,2)
      urprmsu2(:,:,:,:,2,1) =-a4vector(:,:,:,:,2)
      uiprmsu2(:,:,:,:,1,1) = a4vector(:,:,:,:,3)
      uiprmsu2(:,:,:,:,2,2) =-a4vector(:,:,:,:,3)
      uiprmsu2(:,:,:,:,1,2) = a4vector(:,:,:,:,1)
      uiprmsu2(:,:,:,:,2,1) = a4vector(:,:,:,:,1)

c     this calculates U --> U'=U * staple^{\dag} / k, the U (full link) coming out
c     of su2random is here replaced by U'(partial link,
c     depends on ihat the direction)

      urnewsu2 = 0.0d0
      uinewsu2 = 0.0d0
      do ic=1,ncsu2
         do jc=1,ncsu2
            do kc=1,ncsu2
                  urnewsu2(:,:,:,:,ic,jc) = urnewsu2(:,:,:,:,ic,jc) +
     &              ( urprmsu2(:,:,:,:,ic,kc) * phbsr(:,:,:,:,jc,kc) +
     &                uiprmsu2(:,:,:,:,ic,kc) * phbsi(:,:,:,:,jc,kc) )
     &              / k(:,:,:,:)
                  uinewsu2(:,:,:,:,ic,jc) = uinewsu2(:,:,:,:,ic,jc) +
     &              ( uiprmsu2(:,:,:,:,ic,kc) * phbsr(:,:,:,:,jc,kc) -
     &                urprmsu2(:,:,:,:,ic,kc) * phbsi(:,:,:,:,jc,kc) )
     &              / k(:,:,:,:)
            end do
         end do
      end do

      return
      end subroutine pseudoheat
```

E.3 Generating Random $SU_c(2)$ Matrices

```
c
c     ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Author: Frederic D.R. Bonnet & Derek B. Leinweber: June 1998.
c     subroutine that calculates the random su2 configuration
c
      subroutine su2random(ursu2,uisu2)
      implicit none
      include 'latticeSize.h'

c     global variables

      integer,parameter                                     :: ncsu2=2
      integer,parameter                                     :: nsigma=ncsu2*ncsu2
      integer,parameter                                     :: mu=4

      double precision,dimension(nx,ny,nz,nt,mu,nsigma)     :: a4vector
```

```
cmf$  layout a4vector(:news,:news,:news,:news,:serial,:serial)
      double precision,dimension(nx,ny,nz,nt,mu,ncsu2,ncsu2)  :: ursu2,uisu2
cmf$  layout ursu2(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout uisu2(:news,:news,:news,:news,:serial,:serial,:serial)

c     local variables

      integer,parameter                                       :: nr=2
      double precision,parameter                              :: pi=3.141592653d0
      double precision,dimension(nx,ny,nz,nt,mu)              :: norm
cmf$  layout norm(:news,:news,:news,:news,:serial)
      double precision,dimension(nx,ny,nz,nt,mu,nr)           :: r,theta
cmf$  layout r(:news,:news,:news,:news,:serial,:serial)
cmf$  layout theta(:news,:news,:news,:news,:serial,:serial)
      double precision,dimension(nx*ny*nz*nt*mu*nr)           :: harvestr
cmf$  layout harvestr(:news)
      integer,dimension(6)                          :: shaper=(/nx,ny,nz,nt,mu,nr/)
cmf$  layout shaper(:news)
      integer                                                 :: ic,jc,isigma

c     call one set of random numbers and put them in a big array called r,
c     it can be splited into 2, accessing both sides of the array using parameter nr.
c     the random number is in (0,1) taking the nat log maps it onto (-infty,0]
c     multiplying it by -2 maps it to [0,infty)

      CALL CMF_random( r )
      where( r==0.0d0 ) r = 1.0d0
c      where( harvestr==0.0d0 ) harvestr = 1.0d0
c      r = reshape( harvestr,shaper )
      r = sqrt( -2.0d0*log(r) )
      CALL CMF_random( theta )
c      theta = reshape( harvestr,shaper )
      theta = 2.0d0 * pi * theta

c     a4vetor 1 and 2 are independent and gaussian distributed with
c     mean 0 and standard deviation 1. repeat process to get a4vec 3 and 4.
c     It is the cos and sine that normally distributed.
c     a4vector becomes normaly distributed on S^4

      a4vector(:,:,:,:,:,1) = r(:,:,:,:,:,1) * cos( theta(:,:,:,:,:,1) )
      a4vector(:,:,:,:,:,2) = r(:,:,:,:,:,1) * sin( theta(:,:,:,:,:,1) )
      a4vector(:,:,:,:,:,3) = r(:,:,:,:,:,2) * cos( theta(:,:,:,:,:,2) )
      a4vector(:,:,:,:,:,4) = r(:,:,:,:,:,2) * sin( theta(:,:,:,:,:,2) )

      norm = sqrt( sum( a4vector**2,dim=6 ) )

c     it when we divide by the norm= r1**2+r2**2 that the points are brought
c     back onto the surface of the sphere

      do isigma = 1, nsigma
         a4vector(:,:,:,:,:,isigma) = a4vector(:,:,:,:,:,isigma) / norm(:,:,:,:,:)
      end do
```

```
c     converting a4vector to an su2 matrix

      ursu2(:,:,:,:,:,1,1) = a4vector(:,:,:,:,:,4)
      ursu2(:,:,:,:,:,2,2) = a4vector(:,:,:,:,:,4)
      ursu2(:,:,:,:,:,1,2) = a4vector(:,:,:,:,:,2)
      ursu2(:,:,:,:,:,2,1) =-a4vector(:,:,:,:,:,2)
      uisu2(:,:,:,:,:,1,1) = a4vector(:,:,:,:,:,3)
      uisu2(:,:,:,:,:,2,2) =-a4vector(:,:,:,:,:,3)
      uisu2(:,:,:,:,:,1,2) = a4vector(:,:,:,:,:,1)
      uisu2(:,:,:,:,:,2,1) = a4vector(:,:,:,:,:,1)

      return
      end subroutine su2random
```

E.4 Generating Random $SU_c(3)$ Matrices

```
c
c     cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Author: Frederic D.R. Bonnet & Derek B. Leinweber: Jully 1998.
c     subroutine su3random calculates the random su3 configuration
c     it gets a complete set of random number for the lattice in the SU(3)
c     group.
c
      subroutine su3random(ur,ui)
      implicit none
      include 'latticeSize.h'

c     global variables

      integer,parameter                                      :: ncsu2=2,nc=3
      integer,parameter                                      :: mu=4

      double precision,dimension(nx,ny,nz,nt,mu,nc,nc)       :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)

c     local variables

      double precision,dimension(nx,ny,nz,nt,mu,nc,nc)       :: urprm,uiprm
cmf$  layout urprm(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout uiprm(:news,:news,:news,:news,:serial,:serial,:serial)
      double precision,dimension(nx,ny,nz,nt,mu,ncsu2,ncsu2)  :: ursu2,uisu2
cmf$  layout ursu2(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout uisu2(:news,:news,:news,:news,:serial,:serial,:serial)

      integer                                                :: ic,jc,kc

      interface
       subroutine su2random(ursu2,uisu2)
        implicit none
```

```
          include 'latticeSize.h'
          integer,parameter                                      :: ncsu2=2
          integer,parameter                                      :: mu=4
          double precision,dimension(nx,ny,nz,nt,mu,ncsu2,ncsu2):: ursu2,uisu2
cmf$  layout ursu2(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout uisu2(:news,:news,:news,:news,:serial,:serial,:serial)
         end subroutine su2random
        end interface

c       start of the executable commands

c       we call su2random to form a matrix called matrixa,matrixb.
c       these two matrix have the form matrixa = [ [SU(2)] 0 ],  matrixb[ 1    0    ].
c                                                [    0    1 ],         [ 0 [SU(2)] ]
c       Where [SU(2)] is hotwired vi ursu2,uisu2.
c       Both matrixa and matrixb have a double precision and imaginary part.
c       matrixar,matrixai,
c       similarly for matrixb

        call su2random(ursu2,uisu2)

c       we next make a product of the matrices in such a way that
c       link Urpm=urprm+iuiprm=( ar + iai ) * ( ur + iui )
c                             =( ar*ur - ai*ui ) + i( ar*ui + ai*ur)
c       by hardwiring the matrix indices for optimization

        do ic=1,nc-1
          do jc=1,nc
            urprm(:,:,:,:,:,ic,jc) =
     &      ( ursu2(:,:,:,:,:,ic,1) * ur(:,:,:,:,:,1,jc) +
     &        ursu2(:,:,:,:,:,ic,2) * ur(:,:,:,:,:,2,jc) -
     &        uisu2(:,:,:,:,:,ic,1) * ui(:,:,:,:,:,1,jc) -
     &        uisu2(:,:,:,:,:,ic,2) * ui(:,:,:,:,:,2,jc) )
            uiprm(:,:,:,:,:,ic,jc) =
     &      ( ursu2(:,:,:,:,:,ic,1) * ui(:,:,:,:,:,1,jc) +
     &        ursu2(:,:,:,:,:,ic,2) * ui(:,:,:,:,:,2,jc) +
     &        uisu2(:,:,:,:,:,ic,1) * ur(:,:,:,:,:,1,jc) +
     &        uisu2(:,:,:,:,:,ic,2) * ur(:,:,:,:,:,2,jc) )
          end do
        end do

        do jc=1,nc
          urprm(:,:,:,:,:,3,jc) = ur(:,:,:,:,:,3,jc)
          uiprm(:,:,:,:,:,3,jc) = ui(:,:,:,:,:,3,jc)
        end do

c       we next make a product of the matrices in such a way that link
c       Udblerpm=urdbleprm+iuidbleprm = ( br + ibi ) * ( urprm + iuiprm )
c                                     = ( br*urprm - bi*uiprm ) + i( br*uiprm + bi*urprm)

        call su2random(ursu2,uisu2)
```

```
      do ic=2,nc
        do jc=1,nc
          ur(:,:,:,:,:,ic,jc) =
     &      ( ursu2(:,:,:,:,:,ic-1,1) * urprm(:,:,:,:,:,2,jc) +
     &        ursu2(:,:,:,:,:,ic-1,2) * urprm(:,:,:,:,:,3,jc) -
     &        uisu2(:,:,:,:,:,ic-1,1) * uiprm(:,:,:,:,:,2,jc) -
     &        uisu2(:,:,:,:,:,ic-1,2) * uiprm(:,:,:,:,:,3,jc) )
          ui(:,:,:,:,:,ic,jc) =
     &      ( ursu2(:,:,:,:,:,ic-1,1) * uiprm(:,:,:,:,:,2,jc) +
     &        ursu2(:,:,:,:,:,ic-1,2) * uiprm(:,:,:,:,:,3,jc) +
     &        uisu2(:,:,:,:,:,ic-1,1) * urprm(:,:,:,:,:,2,jc) +
     &        uisu2(:,:,:,:,:,ic-1,2) * urprm(:,:,:,:,:,3,jc) )
        end do
      end do

      do jc=1,nc
        ur(:,:,:,:,:,1,jc) = urprm(:,:,:,:,:,1,jc)
        ui(:,:,:,:,:,1,jc) = uiprm(:,:,:,:,:,1,jc)
      end do

      return
      end subroutine su3random
```

E.5 Reunitarization of the $SU_c(2)$ Matrices

```
c
c     cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     subroutine that fixes the su2 links. This subroutine needs to
c     called after a certain amount thermalisation has been done, the
c     purpose being to keep the links within the SU(2) algebra. This
c     subroutines forces the condition of unity U*Udag=I
c
      subroutine fixsu2(ur,ui)
      implicit none

c     global variables

      integer,parameter                   :: nx=8,ny=8,nz=8,nt=8 !lattice size
      integer,parameter                   :: nc=2                !color
      integer,parameter                   :: mu=4                !direction

      real,dimension(nx,ny,nz,nt,mu,nc,nc):: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  align ui(a,b,c,d,e,f,g) with ur(a,b,c,d,e,f,g)

c     local variables

      real,dimension(nx,ny,nz,nt,mu)      :: normr,normi
cmf$  align normr(a,b,c,d,e) with ur(a,b,c,d,e,1,1)
cmf$  align normi(a,b,c,d,e) with ui(a,b,c,d,e,1,1)
      integer                             :: jc              !counters
```

```
c
c     first normalise first row
c
      normr = sqrt( ur(:,:,:,:,:,1,1)**2 + ur(:,:,:,:,:,1,2)**2 +
     &              ui(:,:,:,:,:,1,1)**2 + ui(:,:,:,:,:,1,2)**2 )

      do jc=1,nc
         ur(:,:,:,:,:,1,jc) = ur(:,:,:,:,:,1,jc) / normr(:,:,:,:,:)
         ui(:,:,:,:,:,1,jc) = ui(:,:,:,:,:,1,jc) / normr(:,:,:,:,:)
      end do
c
c     now compute row2 - (row2 dot row1)*row1
c
      normr = ur(:,:,:,:,:,2,1) * ur(:,:,:,:,:,1,1) +
     &        ui(:,:,:,:,:,2,1) * ui(:,:,:,:,:,1,1) +
     &        ur(:,:,:,:,:,2,2) * ur(:,:,:,:,:,1,2) +
     &        ui(:,:,:,:,:,2,2) * ui(:,:,:,:,:,1,2)

      normi = ui(:,:,:,:,:,2,1) * ur(:,:,:,:,:,1,1) -
     &        ur(:,:,:,:,:,2,1) * ui(:,:,:,:,:,1,1) +
     &        ui(:,:,:,:,:,2,2) * ur(:,:,:,:,:,1,2) -
     &        ur(:,:,:,:,:,2,2) * ui(:,:,:,:,:,1,2)

      do jc=1,nc
         ur(:,:,:,:,:,2,jc) = ur(:,:,:,:,:,2,jc) -
     &        ( normr * ur(:,:,:,:,:,1,jc) - normi * ui(:,:,:,:,:,1,jc) )
         ui(:,:,:,:,:,2,jc) = ui(:,:,:,:,:,2,jc) -
     &        ( normr * ui(:,:,:,:,:,1,jc) + normi * ur(:,:,:,:,:,1,jc) )
      end do
c
c     Now normalise the second row
c
      normr = sqrt( ur(:,:,:,:,:,2,1)**2 + ui(:,:,:,:,:,2,1)**2 +
     &              ur(:,:,:,:,:,2,2)**2 + ui(:,:,:,:,:,2,2)**2 )

      do jc=1,nc
         ur(:,:,:,:,:,2,jc) = ur(:,:,:,:,:,2,jc) / normr(:,:,:,:,:)
         ui(:,:,:,:,:,2,jc) = ui(:,:,:,:,:,2,jc) / normr(:,:,:,:,:)
      end do

      return
      end subroutine fixsu2
```

E.6 Calculating the Staples

```
c
c     cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Author: Frederic D.R. Bonnet & D. B. Leinweber: October 1998.
c     the subroutine staples combines the wilson staples(squares plaquettes) and
c     the improved staples(the rectangles plaquettes) for one passed in xhat
c     direction. It returns the variables stapler and staplei.
```

```
c
      subroutine staples(ur,ui,stapler,staplei,xhat,local,itype,uzero)
      implicit none
      include 'latticeSize.h'

c     global variables

      integer,parameter                                      :: nc=3
      integer,parameter                                      :: mu=4

      integer                                                :: xhat
      integer                                                :: itype
      double precision                                       :: uzero

      logical                                                :: local

      double precision,dimension(nx,ny,nz,nt,nc,nc)          :: stapler,staplei
cmf$  layout stapler(:news,:news,:news,:news,:serial,:serial)
cmf$  layout staplei(:news,:news,:news,:news,:serial,:serial)
      double precision,dimension(nx,ny,nz,nt,mu,nc,nc)       :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)

c     local variables

      double precision,dimension(nx,ny,nz,nt,nc,nc)          :: rectr,recti
cmf$  layout rectr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout recti(:news,:news,:news,:news,:serial,:serial)

      interface
         SUBROUTINE squares(ur,ui,squarer,squarei,xhat,local)
          IMPLICIT NONE
          include 'latticeSize.h'
          integer,parameter                                  :: nc=3
          integer,parameter                                  :: mu=4
          integer                                            :: xhat
          logical                                            :: local
          double precision,dimension(nx,ny,nz,nt,nc,nc)      :: squarer,squarei
cmf$  layout squarer(:news,:news,:news,:news,:serial,:serial)
cmf$  layout squarei(:news,:news,:news,:news,:serial,:serial)
          double precision,dimension(nx,ny,nz,nt,mu,nc,nc)   :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)
         end SUBROUTINE squares
         SUBROUTINE rectangles(ur,ui,rectr,recti,xhat,local)
          IMPLICIT NONE
          include 'latticeSize.h'
          integer,parameter                                  :: nc=3
          integer,parameter                                  :: mu=4
          integer                                            :: xhat
          logical                                            :: local
          double precision,dimension(nx,ny,nz,nt,nc,nc)      :: rectr,recti
```

```
cmf$  layout rectr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout recti(:news,:news,:news,:news,:serial,:serial)
          double precision,dimension(nx,ny,nz,nt,mu,nc,nc)    :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)
         end SUBROUTINE rectangles
      end interface

c     start of the executions commands

      call squares(ur,ui,stapler,staplei,xhat,local)

      if(itype==1) then
         call rectangles(ur,ui,rectr,recti,xhat,local)
         stapler = ( 5.0d0 / 3.0d0 ) * stapler -
     &             ( 1.0d0 / ( 12.0d0 * uzero**2 ) ) * rectr
         staplei = ( 5.0d0 / 3.0d0 ) * staplei -
     &             ( 1.0d0 / ( 12.0d0 * uzero**2 ) ) * recti
      end if

      return
      end subroutine staples
```

E.7 Calculating the 1×1 Wilson Loop

```
c
c     cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Author: Frederic D.R. Bonnet; date: 28th of Jully 1998.
c     computes the product of links for the action associated with a link in the
c     xhat direction.
c
      subroutine squares(ur,ui,squarer,squarei,xhat,local)
      implicit none
      include 'latticeSize.h'

c     global variable

      integer,parameter                                        :: nc=3
      integer,parameter                                        :: mu=4

      integer                                                  :: xhat
      logical                                                  :: local

      double precision,dimension(nx,ny,nz,nt,nc,nc)            :: squarer,squarei
cmf$  layout squarer(:news,:news,:news,:news,:serial,:serial)
cmf$  layout squarei(:news,:news,:news,:news,:serial,:serial)
      double precision,dimension(nx,ny,nz,nt,mu,nc,nc)         :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)

c     local variables, temporary product variables
```

```
      integer,dimension(3)                                    :: yhat
cmf$  layout yhat(:serial)
      double precision,dimension(nx,ny,nz,nt,nc,nc)           :: tr,ti,tsr,tsi
cmf$  layout tr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout ti(:news,:news,:news,:news,:serial,:serial)
cmf$  layout tsr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout tsi(:news,:news,:news,:news,:serial,:serial)
      double precision,dimension(nx,ny,nz,nt,nc,nc)           :: usqr,usqi
cmf$  layout usqr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout usqi(:news,:news,:news,:news,:serial,:serial)

      integer                                                 :: ic,jc,kc,imu,i

c     starting of the execution commands

c     setting up the yhat array
c     the yhat(1)=yhat,yhat(2)=zhat and yhat(3)=that when xhat eq 1
c     the yhat(1)=zhat,yhat(2)=that and yhat(3)=xhat when xhat eq 2
c     the yhat(1)=that,yhat(2)=xhat and yhat(3)=yhat when xhat eq 3
c     the yhat(1)=xhat,yhat(2)=yhat and yhat(3)=zhat when xhat eq 4

      do imu=1,mu-1
         yhat(imu)    = mod( xhat + imu ,4 )
         if(yhat(imu) .eq. 0) then
            yhat(imu) = 4
         end if
      end do

c     calculation of the link products in the positive plaquette.
c     starting point in the xhat direction with the mask

      squarer = 0.0d0
      squarei = 0.0d0

c     the xy,xz and xt contour when xhat eq 1
c     the yz,yt and yx contour when xhat eq 2
c     the zt,zx and zy contour when xhat eq 3
c     the tx,ty and tz contour when xhat eq 4

      do i=1,mu-1                        !loop over the yhat array
         tr = 0.0d0
         ti = 0.0d0
         tsr = cshift(ur(:,:,:,:,yhat(i),:,:),dim=xhat,shift=1)
         tsi = cshift(ui(:,:,:,:,yhat(i),:,:),dim=xhat,shift=1)
         usqr = cshift(ur(:,:,:,:,xhat,:,:),dim=yhat(i),shift=1)
         usqi = cshift(ui(:,:,:,:,xhat,:,:),dim=yhat(i),shift=1)
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  tr(:,:,:,:,ic,jc) = tr(:,:,:,:,ic,jc)                +
     &                        ( tsr(:,:,:,:,ic,kc) * usqr(:,:,:,:,jc,kc) +
     &                          tsi(:,:,:,:,ic,kc) * usqi(:,:,:,:,jc,kc) )
                  ti(:,:,:,:,ic,jc) = ti(:,:,:,:,ic,jc)                   +
     &                        ( tsi(:,:,:,:,ic,kc) * usqr(:,:,:,:,jc,kc) -
     &                          tsr(:,:,:,:,ic,kc) * usqi(:,:,:,:,jc,kc) )
               end do
            end do
         end do
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  squarer(:,:,:,:,ic,jc) = squarer(:,:,:,:,ic,jc)         +
     &                  ( tr(:,:,:,:,ic,kc) * ur(:,:,:,:,yhat(i),jc,kc)  +
     &                    ti(:,:,:,:,ic,kc) * ui(:,:,:,:,yhat(i),jc,kc)  )
                  squarei(:,:,:,:,ic,jc) = squarei(:,:,:,:,ic,jc)         +
     &                  ( ti(:,:,:,:,ic,kc) * ur(:,:,:,:,yhat(i),jc,kc)  -
     &                    tr(:,:,:,:,ic,kc) * ui(:,:,:,:,yhat(i),jc,kc)  )
               end do
            end do
         end do

c     calculation of the link products in the negative plaquette
c     starting point in the x direction

c     the negative xy,xz and xt contour when xhat eq 1
c     the negative yz,yt and yx contour when xhat eq 2
c     the negative zt,zx and zy contour when xhat eq 3
c     the negative tx,ty and tz contour when xhat eq 4

c     .true. when we calculate the full staple
c     .false. when we calculte plaqbar where
c     only the upper 3 plaquette are needed

         if(local .eq. .true.) then

         tr = 0.0d0
         ti = 0.0d0
         tsr = cshift(cshift(ur(:,:,:,:,yhat(i),:,:),
     &                dim=yhat(i),shift=-1),dim=xhat,shift=1)
         tsi = cshift(cshift(ui(:,:,:,:,yhat(i),:,:),
     &                dim=yhat(i),shift=-1),dim=xhat,shift=1)
         usqr = cshift(ur(:,:,:,:,xhat,:,:),dim=yhat(i),shift=-1)
         usqi = cshift(ui(:,:,:,:,xhat,:,:),dim=yhat(i),shift=-1)
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  tr(:,:,:,:,ic,jc) = tr(:,:,:,:,ic,jc)       +
     &              ( tsr(:,:,:,:,kc,ic) * usqr(:,:,:,:,jc,kc) -
     &                tsi(:,:,:,:,kc,ic) * usqi(:,:,:,:,jc,kc) )
                  ti(:,:,:,:,ic,jc) = ti(:,:,:,:,ic,jc)       +
     &              (-tsr(:,:,:,:,kc,ic) * usqi(:,:,:,:,jc,kc) -
     &                tsi(:,:,:,:,kc,ic) * usqr(:,:,:,:,jc,kc) )
               end do
            end do
         end do

         usqr = cshift(ur(:,:,:,:,yhat(i),:,:),dim=yhat(i),shift=-1)
         usqi = cshift(ui(:,:,:,:,yhat(i),:,:),dim=yhat(i),shift=-1)
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  squarer(:,:,:,:,ic,jc) = squarer(:,:,:,:,ic,jc) +
     &                  ( tr(:,:,:,:,ic,kc) * usqr(:,:,:,:,kc,jc) -
     &                    ti(:,:,:,:,ic,kc) * usqi(:,:,:,:,kc,jc) )
                  squarei(:,:,:,:,ic,jc) = squarei(:,:,:,:,ic,jc) +
     &                  ( tr(:,:,:,:,ic,kc) * usqi(:,:,:,:,kc,jc) +
     &                    ti(:,:,:,:,ic,kc) * usqr(:,:,:,:,kc,jc) )
               end do
            end do
         end do

         end if                                                 !closes the .true. if
      end do                                                    !closes the i=1,3 loop

      return
      end subroutine squares
```

E.8 Calculating the $(1 \times 1), (1 \times 2)$ and (2×1) Wilson Loop

```
c
c     cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Author: Frederic D.R. Bonnet; date: October 1998.
c     computes the product of links for the action associated with a link in the
c     xhat direction. This computes the product of the six rectangular plaquettes
c     associated with the improved action.
c
      subroutine rectangles(ur,ui,rectr,recti,xhat,local)
      implicit none
      include 'latticeSize.h'

c     global variable

      integer,parameter                                        :: nc=3
      integer,parameter                                        :: mu=4

      integer                                                  :: xhat
      logical                                                  :: local

      double precision,dimension(nx,ny,nz,nt,nc,nc)            :: rectr,recti
cmf$  layout rectr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout recti(:news,:news,:news,:news,:serial,:serial)
      double precision,dimension(nx,ny,nz,nt,mu,nc,nc)         :: ur,ui
```

```
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)

c     local variables, temporary product variables

      integer,dimension(3)                                 :: yhat
cmf$  layout yhat(:serial)
      double precision,dimension(nx,ny,nz,nt,nc,nc)        :: t1r,t1i,t2r,t2i
cmf$  layout t1r(:news,:news,:news,:news,:serial,:serial)
cmf$  layout t1i(:news,:news,:news,:news,:serial,:serial)
cmf$  layout t2r(:news,:news,:news,:news,:serial,:serial)
cmf$  layout t2i(:news,:news,:news,:news,:serial,:serial)
      double precision,dimension(nx,ny,nz,nt,nc,nc)        :: usr,usi
cmf$  layout usr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout usi(:news,:news,:news,:news,:serial,:serial)

      integer                                              :: ic,jc,kc,imu,i

c     starting of the execution commands

c     setting up the yhat array
c     the yhat(1)=yhat,yhat(2)=zhat and yhat(3)=that when xhat eq 1
c     the yhat(1)=zhat,yhat(2)=that and yhat(3)=xhat when xhat eq 2
c     the yhat(1)=that,yhat(2)=xhat and yhat(3)=yhat when xhat eq 3
c     the yhat(1)=xhat,yhat(2)=yhat and yhat(3)=zhat when xhat eq 4

      do imu=1,mu-1
         yhat(imu) = 0
      end do

      do imu=1,mu-1
         yhat(imu)    = mod( xhat + imu ,4 )
         if(yhat(imu) .eq. 0) then
            yhat(imu) = 4
         end if
      end do

c     calculation of the link products in the positive plaquette.
c     starting point in the xhat direction with the maskrect

c     first calculate the positive forward 2a x a rectangle

      rectr = 0.0d0
      recti = 0.0d0

      do i=1,mu-1                         !loop over the yhat array
         t1r = 0.0d0
         t1i = 0.0d0
         t2r = cshift(ur(:,:,:,:,xhat,:,:),dim=xhat,shift=1)
         t2i = cshift(ui(:,:,:,:,xhat,:,:),dim=xhat,shift=1)
         usr = cshift(ur(:,:,:,:,yhat(i),:,:),dim=xhat,shift=2)
         usi = cshift(ui(:,:,:,:,yhat(i),:,:),dim=xhat,shift=2)
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)        +
     &                   ( t2r(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) -
     &                     t2i(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) )
                  t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)        +
     &                   ( t2r(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) +
     &                     t2i(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) )
               end do
            end do
         end do

         t2r = 0.0d0
         t2i = 0.0d0
         usr = cshift(cshift(ur(:,:,:,:,xhat,:,:),
     &           dim=yhat(i),shift=1),dim=xhat,shift=1)
         usi = cshift(cshift(ui(:,:,:,:,xhat,:,:),
     &           dim=yhat(i),shift=1),dim=xhat,shift=1)
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  t2r(:,:,:,:,ic,jc) = t2r(:,:,:,:,ic,jc)        +
     &                   ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                     t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                  t2i(:,:,:,:,ic,jc) = t2i(:,:,:,:,ic,jc)        +
     &                   (-t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                     t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
               end do
            end do
         end do

         t1r = 0.0d0
         t1i = 0.0d0
         usr = cshift(ur(:,:,:,:,xhat,:,:),dim=yhat(i),shift=1)
         usi = cshift(ui(:,:,:,:,xhat,:,:),dim=yhat(i),shift=1)
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)     +
     &                ( t2r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                  t2i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                  t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)     +
     &                (-t2r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                  t2i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
               end do
            end do
         end do

         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
```

```
                 rectr(:,:,:,:,ic,jc) = rectr(:,:,:,:,ic,jc)          +
     &                ( t1r(:,:,:,:,ic,kc) * ur(:,:,:,:,yhat(i),jc,kc) +
     &                  t1i(:,:,:,:,ic,kc) * ui(:,:,:,:,yhat(i),jc,kc) )
                 recti(:,:,:,:,ic,jc) = recti(:,:,:,:,ic,jc)          +
     &                (-t1r(:,:,:,:,ic,kc) * ui(:,:,:,:,yhat(i),jc,kc) +
     &                  t1i(:,:,:,:,ic,kc) * ur(:,:,:,:,yhat(i),jc,kc) )
              end do
           end do
        end do

c     now calculating the positive upper rectangles a x 2a

        t1r = 0.0d0
        t1i = 0.0d0
        t2r = cshift(ur(:,:,:,:,yhat(i),:,:),dim=xhat,shift=1)
        t2i = cshift(ui(:,:,:,:,yhat(i),:,:),dim=xhat,shift=1)
        usr = cshift(cshift(ur(:,:,:,:,yhat(i),:,:),
     &               dim=xhat,shift=1),dim=yhat(i),shift=1)
        usi = cshift(cshift(ui(:,:,:,:,yhat(i),:,:),
     &               dim=xhat,shift=1),dim=yhat(i),shift=1)
        do ic=1,nc
           do jc=1,nc
              do kc=1,nc
                 t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)        +
     &                ( t2r(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) -
     &                  t2i(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) )
                 t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)        +
     &                ( t2r(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) +
     &                  t2i(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) )
              end do
           end do
        end do

        t2r = 0.0d0
        t2i = 0.0d0
        usr = cshift(ur(:,:,:,:,xhat,:,:),dim=yhat(i),shift=2)
        usi = cshift(ui(:,:,:,:,xhat,:,:),dim=yhat(i),shift=2)
        do ic=1,nc
           do jc=1,nc
              do kc=1,nc
                 t2r(:,:,:,:,ic,jc) = t2r(:,:,:,:,ic,jc)        +
     &                ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                  t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                 t2i(:,:,:,:,ic,jc) = t2i(:,:,:,:,ic,jc)        +
     &                (-t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                  t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
              end do
           end do
        end do

        t1r = 0.0d0
        t1i = 0.0d0
```

```
         usr = cshift(ur(:,:,:,:,yhat(i),:,:),dim=yhat(i),shift=1)
         usi = cshift(ui(:,:,:,:,yhat(i),:,:),dim=yhat(i),shift=1)
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)       +
     &                 ( t2r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                   t2i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                  t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)       +
     &                 (-t2r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                   t2i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
               end do
            end do
         end do

         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  rectr(:,:,:,:,ic,jc) = rectr(:,:,:,:,ic,jc)          +
     &                 ( t1r(:,:,:,:,ic,kc) * ur(:,:,:,:,yhat(i),jc,kc) +
     &                   t1i(:,:,:,:,ic,kc) * ui(:,:,:,:,yhat(i),jc,kc) )
                  recti(:,:,:,:,ic,jc) = recti(:,:,:,:,ic,jc)          +
     &                 (-t1r(:,:,:,:,ic,kc) * ui(:,:,:,:,yhat(i),jc,kc) +
     &                   t1i(:,:,:,:,ic,kc) * ur(:,:,:,:,yhat(i),jc,kc) )
               end do
            end do
         end do

c     .true. when we calculate the full staple
c     .false. when we calculte impbar where
c     only the upper 3 plaquette are needed

         if(local .eq. .true.) then

c     now calculate the positive backward 2a x a retangle

            t1r = 0.0d0
            t1i = 0.0d0
            t2r = cshift(ur(:,:,:,:,yhat(i),:,:),dim=xhat,shift=1)
            t2i = cshift(ui(:,:,:,:,yhat(i),:,:),dim=xhat,shift=1)
            usr = cshift(ur(:,:,:,:,xhat,:,:),dim=yhat(i),shift=1)
            usi = cshift(ui(:,:,:,:,xhat,:,:),dim=yhat(i),shift=1)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)      +
     &                  ( t2r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                    t2i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                     t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)      +
     &                  (-t2r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                    t2i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
                  end do
               end do
            end do

            t2r = 0.0d0
            t2i = 0.0d0
            usr = cshift(cshift(ur(:,:,:,:,xhat,:,:),
     &                    dim=yhat(i),shift=1),dim=xhat,shift=-1)
            usi = cshift(cshift(ui(:,:,:,:,xhat,:,:),
     &                    dim=yhat(i),shift=1),dim=xhat,shift=-1)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     t2r(:,:,:,:,ic,jc) = t2r(:,:,:,:,ic,jc)      +
     &                    ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                      t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                     t2i(:,:,:,:,ic,jc) = t2i(:,:,:,:,ic,jc)      +
     &                    (-t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                      t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
                  end do
               end do
            end do

            t1r = 0.0d0
            t1i = 0.0d0
            usr = cshift(ur(:,:,:,:,yhat(i),:,:),dim=xhat,shift=-1)
            usi = cshift(ui(:,:,:,:,yhat(i),:,:),dim=xhat,shift=-1)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)      +
     &                    ( t2r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                      t2i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                     t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)      +
     &                    (-t2r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                      t2i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
                  end do
               end do
            end do

            usr = cshift(ur(:,:,:,:,xhat,:,:),dim=xhat,shift=-1)
            usi = cshift(ui(:,:,:,:,xhat,:,:),dim=xhat,shift=-1)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     rectr(:,:,:,:,ic,jc) = rectr(:,:,:,:,ic,jc)    +
     &                      ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) -
     &                        t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) )
                     recti(:,:,:,:,ic,jc) = recti(:,:,:,:,ic,jc)    +
     &                      ( t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) +
     &                        t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) )
                  end do
               end do
            end do

c     calculation of the link products in the negative plaquette
c     starting point in the x direction

c     the negative xy,xz and xt contour when xhat eq 1
c     the negative yz,yt and yx contour when xhat eq 2
c     the negative zt,zx and zy contour when xhat eq 3
c     the negative tx,ty and tz contour when xhat eq 4


c     first calculate the negative forward 2a x a rectangle

            t1r = 0.0d0
            t1i = 0.0d0
            t2r = cshift(ur(:,:,:,:,xhat,:,:),dim=xhat,shift=1)
            t2i = cshift(ui(:,:,:,:,xhat,:,:),dim=xhat,shift=1)
            usr = cshift(cshift(ur(:,:,:,:,yhat(i),:,:),
     &                   dim=xhat,shift=2),dim=yhat(i),shift=-1)
            usi = cshift(cshift(ui(:,:,:,:,yhat(i),:,:),
     &                   dim=xhat,shift=2),dim=yhat(i),shift=-1)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)      +
     &                  ( t2r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                    t2i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                     t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)      +
     &                  (-t2r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                    t2i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
                  end do
               end do
            end do

            t2r = 0.0d0
            t2i = 0.0d0
            usr = cshift(cshift(ur(:,:,:,:,xhat,:,:),
     &                   dim=yhat(i),shift=-1),dim=xhat,shift=1)
            usi = cshift(cshift(ui(:,:,:,:,xhat,:,:),
     &                   dim=yhat(i),shift=-1),dim=xhat,shift=1)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     t2r(:,:,:,:,ic,jc) = t2r(:,:,:,:,ic,jc)      +
     &                  ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                    t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                     t2i(:,:,:,:,ic,jc) = t2i(:,:,:,:,ic,jc)      +
     &                  (-t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                    t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
                  end do
               end do
            end do
```

```
         t1r = 0.0d0
         t1i = 0.0d0
         usr = cshift(ur(:,:,:,:,xhat,:,:),dim=yhat(i),shift=-1)
         usi = cshift(ui(:,:,:,:,xhat,:,:),dim=yhat(i),shift=-1)
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)      +
     &               ( t2r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                 t2i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                  t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)      +
     &               (-t2r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                 t2i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
               end do
            end do
         end do

         usr = cshift(ur(:,:,:,:,yhat(i),:,:),dim=yhat(i),shift=-1)
         usi = cshift(ui(:,:,:,:,yhat(i),:,:),dim=yhat(i),shift=-1)
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  rectr(:,:,:,:,ic,jc) = rectr(:,:,:,:,ic,jc)    +
     &                 ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) -
     &                   t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) )
                  recti(:,:,:,:,ic,jc) = recti(:,:,:,:,ic,jc)    +
     &                 ( t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) +
     &                   t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) )
               end do
            end do
         end do

c     calculate the negative backward 2a x a rectangle

         t1r = 0.0d0
         t1i = 0.0d0
         t2r = cshift(cshift(ur(:,:,:,:,yhat(i),:,:),
     &                dim=yhat(i),shift=-1),dim=xhat,shift=1)
         t2i = cshift(cshift(ui(:,:,:,:,yhat(i),:,:),
     &                dim=yhat(i),shift=-1),dim=xhat,shift=1)
         usr = cshift(ur(:,:,:,:,xhat,:,:),dim=yhat(i),shift=-1)
         usi = cshift(ui(:,:,:,:,xhat,:,:),dim=yhat(i),shift=-1)
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)      +
     &               ( t2r(:,:,:,:,kc,ic) * usr(:,:,:,:,jc,kc) -
     &                 t2i(:,:,:,:,kc,ic) * usi(:,:,:,:,jc,kc) )
                  t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)      +
     &               (-t2r(:,:,:,:,kc,ic) * usi(:,:,:,:,jc,kc) -
     &                 t2i(:,:,:,:,kc,ic) * usr(:,:,:,:,jc,kc) )
```

```
            end do
         end do
      end do

      t2r = 0.0d0
      t2i = 0.0d0
      usr = cshift(cshift(ur(:,:,:,:,xhat,:,:),
&                  dim=yhat(i),shift=-1),dim=xhat,shift=-1)
      usi = cshift(cshift(ui(:,:,:,:,xhat,:,:),
&                  dim=yhat(i),shift=-1),dim=xhat,shift=-1)
      do ic=1,nc
         do jc=1,nc
            do kc=1,nc
               t2r(:,:,:,:,ic,jc) = t2r(:,:,:,:,ic,jc)      +
&                 ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
&                   t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
               t2i(:,:,:,:,ic,jc) = t2i(:,:,:,:,ic,jc)      +
&                 (-t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
&                   t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
            end do
         end do
      end do

      t1r = 0.0d0
      t1i = 0.0d0
      usr = cshift(cshift(ur(:,:,:,:,yhat(i),:,:),
&                  dim=xhat,shift=-1),dim=yhat(i),shift=-1)
      usi = cshift(cshift(ui(:,:,:,:,yhat(i),:,:),
&                  dim=xhat,shift=-1),dim=yhat(i),shift=-1)
      do ic=1,nc
         do jc=1,nc
            do kc=1,nc
               t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)      +
&                 ( t2r(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) -
&                   t2i(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) )
               t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)      +
&                 ( t2r(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) +
&                   t2i(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) )
            end do
         end do
      end do

      usr = cshift(ur(:,:,:,:,xhat,:,:),dim=xhat,shift=-1)
      usi = cshift(ui(:,:,:,:,xhat,:,:),dim=xhat,shift=-1)
      do ic=1,nc
         do jc=1,nc
            do kc=1,nc
               rectr(:,:,:,:,ic,jc) = rectr(:,:,:,:,ic,jc)    +
&                   ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) -
&                     t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) )
               recti(:,:,:,:,ic,jc) = recti(:,:,:,:,ic,jc)    +
&                   ( t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) +
     &                       t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) )
                 end do
              end do
           end do

c     now calculating the negative lower rectangles a x 2a

           t1r = 0.0d0
           t1i = 0.0d0
           t2r = cshift(cshift(ur(:,:,:,:,yhat(i),:,:),
     &                    dim=xhat,shift=1),dim=yhat(i),shift=-1)
           t2i = cshift(cshift(ui(:,:,:,:,yhat(i),:,:),
     &                    dim=xhat,shift=1),dim=yhat(i),shift=-1)
           usr = cshift(cshift(ur(:,:,:,:,yhat(i),:,:),
     &                    dim=xhat,shift=1),dim=yhat(i),shift=-2)
           usi = cshift(cshift(ui(:,:,:,:,yhat(i),:,:),
     &                    dim=xhat,shift=1),dim=yhat(i),shift=-2)
           do ic=1,nc
              do jc=1,nc
                 do kc=1,nc
                    t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)      +
     &                 ( t2r(:,:,:,:,kc,ic) * usr(:,:,:,:,jc,kc) -
     &                   t2i(:,:,:,:,kc,ic) * usi(:,:,:,:,jc,kc) )
                    t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)      +
     &                 (-t2r(:,:,:,:,kc,ic) * usi(:,:,:,:,jc,kc) -
     &                   t2i(:,:,:,:,kc,ic) * usr(:,:,:,:,jc,kc) )
                 end do
              end do
           end do

           t2r = 0.0d0
           t2i = 0.0d0
           usr = cshift(ur(:,:,:,:,xhat,:,:),dim=yhat(i),shift=-2)
           usi = cshift(ui(:,:,:,:,xhat,:,:),dim=yhat(i),shift=-2)
           do ic=1,nc
              do jc=1,nc
                 do kc=1,nc
                    t2r(:,:,:,:,ic,jc) = t2r(:,:,:,:,ic,jc)      +
     &                 ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                   t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                    t2i(:,:,:,:,ic,jc) = t2i(:,:,:,:,ic,jc)      +
     &                 (-t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                   t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
                 end do
              end do
           end do

           t1r = 0.0d0
           t1i = 0.0d0
           usr = cshift(ur(:,:,:,:,yhat(i),:,:),dim=yhat(i),shift=-2)
           usi = cshift(ui(:,:,:,:,yhat(i),:,:),dim=yhat(i),shift=-2)
           do ic=1,nc
```

```
            do jc=1,nc
               do kc=1,nc
                  t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)      +
     &               ( t2r(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) -
     &                 t2i(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) )
                  t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)      +
     &               ( t2r(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) +
     &                 t2i(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) )
               end do
            end do
         end do

         usr = cshift(ur(:,:,:,:,yhat(i),:,:),dim=yhat(i),shift=-1)
         usi = cshift(ui(:,:,:,:,yhat(i),:,:),dim=yhat(i),shift=-1)
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  rectr(:,:,:,:,ic,jc) = rectr(:,:,:,:,ic,jc)    +
     &                 ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) -
     &                   t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) )
                  recti(:,:,:,:,ic,jc) = recti(:,:,:,:,ic,jc)    +
     &                 ( t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) +
     &                   t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) )
               end do
            end do
         end do

      end if
      end do

      return
      end subroutine rectangles
```

E.9 Constructing $SU_c(3)$ Matrices

```
c
c     cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Author: Frederic D.R. Bonnet & D. B. Leinweber: 4th of August 1998.
c     subroutine pseudosweep updates the current link using a psedo-heatbath
c     algorithm. It calls the staples and pseudoheat.
c
      subroutine pseudosweep(ur,ui,mask,umask,beta,itype,nsub,uzero)
      implicit none
      include 'latticeSize.h'

c     global variables

      integer,parameter                                       :: nc=3,ncsu2=2
      integer,parameter                                       :: nsigma=ncsu2*ncsu2
      integer,parameter                                       :: mu=4
      integer,parameter                                       :: nmask=16
      integer                                                 :: itype
      integer                                                 :: nsub
      integer                                                 :: umask

      double precision                                        :: uzero
      double precision                                        :: beta

      double precision,dimension(nx,ny,nz,nt,mu,nc,nc)        :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)
      logical,dimension(nx,ny,nz,nt,mu,nmask)                 :: mask
cmf$  layout mask(:news,:news,:news,:news,:serial,:serial)

c     local variables

      double precision,dimension(nx,ny,nz,nt,nc,nc)           :: stapler,staplei
      double precision,dimension(nx,ny,nz,nt,nc,nc)           :: ltsr,ltsi
      double precision,dimension(nx,ny,nz,nt,nc,nc)           :: urprmsu3,uiprmsu3
cmf$  layout stapler(:news,:news,:news,:news,:serial,:serial)
cmf$  layout staplei(:news,:news,:news,:news,:serial,:serial)
cmf$  layout ltsr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout ltsi(:news,:news,:news,:news,:serial,:serial)
cmf$  layout urprmsu3(:news,:news,:news,:news,:serial,:serial)
cmf$  layout uiprmsu3(:news,:news,:news,:news,:serial,:serial)
      double precision,dimension(nx,ny,nz,nt,ncsu2,ncsu2)     :: phbsr,phbsi
      double precision,dimension(nx,ny,nz,nt,ncsu2,ncsu2)     :: ursu2,uisu2
cmf$  layout phbsr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout phbsi(:news,:news,:news,:news,:serial,:serial)
cmf$  layout ursu2(:news,:news,:news,:news,:serial,:serial)
cmf$  layout uisu2(:news,:news,:news,:news,:serial,:serial)

      double precision                                        :: betanew
      integer                                                 :: of1
      integer                                                :: imask,ihat,ic,jc,kc,ic3
      integer                                                 :: isub

      interface
         SUBROUTINE staples(ur,ui,stapler,staplei,xhat,local,itype,uzero)
          IMPLICIT NONE
          include 'latticeSize.h'
          integer,parameter                                   :: nc=3
          integer,parameter                                   :: mu=4
          integer                                             :: xhat
          integer                                             :: itype
          double precision                                    :: uzero
          logical                                             :: local
          double precision,dimension(nx,ny,nz,nt,nc,nc)       :: stapler,staplei
cmf$  layout stapler(:news,:news,:news,:news,:serial,:serial)
cmf$  layout staplei(:news,:news,:news,:news,:serial,:serial)
          double precision,dimension(nx,ny,nz,nt,mu,nc,nc)    :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
```

```
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)
         end SUBROUTINE staples
         subroutine pseudoheat(urnewsu2,uinewsu2,phbsr,phbsi,mask,imask,beta,ihat)
          implicit none
          include 'latticeSize.h'
          integer,parameter                                  :: ncsu2=2
          integer,parameter                                  :: mu=4
          integer,parameter                                  :: nmask=16
          double precision                                   :: beta
          integer                                            :: ihat,imask
          double precision,dimension(nx,ny,nz,nt,ncsu2,ncsu2) :: phbsr,phbsi
          double precision,dimension(nx,ny,nz,nt,ncsu2,ncsu2) :: urnewsu2,uinewsu2
cmf$  layout phbsr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout phbsi(:news,:news,:news,:news,:serial,:serial)
cmf$  layout urnewsu2(:news,:news,:news,:news,:serial,:serial)
cmf$  layout uinewsu2(:news,:news,:news,:news,:serial,:serial)
          logical,dimension(nx,ny,nz,nt,mu,nmask)            :: mask
cmf$  layout mask(:news,:news,:news,:news,:serial,:serial)
         end subroutine pseudoheat
      end interface

c     starts of the running commands

c     Here we need to multiply Beta by a factor of 2/nc=3 because for the su2 case
c     there is a factor of which cancels the one 1/ncsu2 in the probability
c     distribution function. This beta value is to be passed into pseudoheat

      betanew = ( 2.0d0 / nc ) * beta

c     the ihat do loop, is to loop over all the possible Euclidean directions.
c     The imask do loop, is to ensure that the entire lattice is considered:
c     all true then all false.

      do ihat=1,mu
         do imask=1,umask

c     calculate the staple in the ihat direction

            call staples(ur,ui,stapler,staplei,ihat,.true.,itype,uzero)

            do isub=1,nsub
c
c     case 1. a_1
c
c     The Wilson action can be written as S(U)=Sum_p(Re(U*U_p))+constant
c                                            =Re(Tr(U*R))+constant
c     R is just the sum over the six plaquette namely the staples: stapler,staplei.

c     now we need to calculate the product of the staples time the link variable in
c     each direction U*R=ltsr+iltsi=( ur + iui ) * ( stapler + istaplei )
c                                 =(ur*stapler - ui*staplei)+i(ur*staplei + ui*staplei)
            ltsr = 0.0d0
            ltsi = 0.0d0
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     ltsr(:,:,:,:,ic,jc) = ltsr(:,:,:,:,ic,jc) +
     &                  ( ur(:,:,:,:,ihat,ic,kc) * stapler(:,:,:,:,kc,jc) -
     &                    ui(:,:,:,:,ihat,ic,kc) * staplei(:,:,:,:,kc,jc) )
                     ltsi(:,:,:,:,ic,jc) = ltsi(:,:,:,:,ic,jc) +
     &                  ( ur(:,:,:,:,ihat,ic,kc) * staplei(:,:,:,:,kc,jc) +
     &                    ui(:,:,:,:,ihat,ic,kc) * stapler(:,:,:,:,kc,jc) )
                  end do
               end do
            end do

c     The distribution we are interested in the following
c     dP(a_k) = d^{(k)}a_k * exp[-beta * S(a_k*U^{(k-1)})]/Z_k(U^{(k-1)})
c     where d^{(k)}a_k is the Haar measure on the SU(2)_k and
c     Z_k(U^{(k-1)}) = \int_{SU(2)_k} da exp[-beta * S(aU)]. If a{\in}SU(2)_K then
c     Z_k(aU) = Z_k(U). In order to generate the above distribution for the minimal
c     set F of SU(2) subgroup of SU(N) namely {F:SU(2)_k, k=1,..,nc-1}, let's first
c     note that S(a_k*U)=Re(Tr a_k * U*R) = Re(Tr \alpha_k * r_k)
c                                         + terms independent of \alpha_k.
c     Where r_k is the 2x2 subgroup of the U*R (i.e. link*staples : lts), this matix
c     has the same block structure as the matrix a_k. This means that \alpha_k and
c     r_k = r_0 * I + i \vec{\sigma}\dot\vec{r} are the same block in the ncxnc
c     matrix located at the (k,k+1)th rows and column.

c     For the first call of the pseudo-heatbath we need to match the matrix block,
c     if this is not done wrong information will be sent to the subroutine therefore
c     returning the wrong numbers. To do that we use an offset parameter called of1
c     with this of1 we can acces the correct entries of the ncxnc lts matrix. We then
c     reconstruct a SU(2) matrix that can be passed into pseudoheat which will also
c     return a SU(2) matrix.

            of1 = 0

            phbsr(:,:,:,:,1,1) = (  ltsr(:,:,:,:,1+of1,1+of1) +
     &                              ltsr(:,:,:,:,2+of1,2+of1) ) / 2.0d0
            phbsr(:,:,:,:,1,2) = (  ltsr(:,:,:,:,1+of1,2+of1) -
     &                              ltsr(:,:,:,:,2+of1,1+of1) ) / 2.0d0
            phbsr(:,:,:,:,2,1) = - phbsr(:,:,:,:,1,2)
            phbsr(:,:,:,:,2,2) =   phbsr(:,:,:,:,1,1)

            phbsi(:,:,:,:,1,1) = (  ltsi(:,:,:,:,1+of1,1+of1) -
     &                              ltsi(:,:,:,:,2+of1,2+of1) ) / 2.0d0
            phbsi(:,:,:,:,1,2) = (  ltsi(:,:,:,:,1+of1,2+of1) +
     &                              ltsi(:,:,:,:,2+of1,1+of1) ) / 2.0d0
            phbsi(:,:,:,:,2,1) =   phbsi(:,:,:,:,1,2)
            phbsi(:,:,:,:,2,2) = - phbsi(:,:,:,:,1,1)

c     now calling the pseudo-heatbath algorithm to update, where the
c     mask is .true., the full QCD configuration
```

```
c     we call pseudoheat to form 2 SU(3) matrices.
c     These two matrices have the form matrixa = [ [SU(2)] 0 ],  matrixb[ 1    0    ].
c                                                [    0    1 ],         [ 0 [SU(2)] ]
c     Where [SU(2)] is hotwired via ursu2,uisu2 and ursu2,uisu2.

            call pseudoheat(ursu2,uisu2,phbsr,phbsi,mask,imask,betanew,ihat)

c     We next make a product of the matrices in such a way that
c     link Uprmsu3 = urprmsu3+iuiprmsu3 = ( ar + iai ) * ( ur + iui )
c                              = ( ar*ur - ai*ui ) + i( ar*ui + ai*ur)
c     by hotwiring the matrix indices for optimization.

            do ic=1,nc-1
               do jc=1,nc
                  where( mask(:,:,:,:,ihat,imask) )
              urprmsu3(:,:,:,:,ic,jc) =
     &                 ( ursu2(:,:,:,:,ic,1) * ur(:,:,:,:,ihat,1,jc) +
     &                   ursu2(:,:,:,:,ic,2) * ur(:,:,:,:,ihat,2,jc) -
     &                   uisu2(:,:,:,:,ic,1) * ui(:,:,:,:,ihat,1,jc) -
     &                   uisu2(:,:,:,:,ic,2) * ui(:,:,:,:,ihat,2,jc) )
              uiprmsu3(:,:,:,:,ic,jc) =
     &                 ( ursu2(:,:,:,:,ic,1) * ui(:,:,:,:,ihat,1,jc) +
     &                   ursu2(:,:,:,:,ic,2) * ui(:,:,:,:,ihat,2,jc) +
     &                   uisu2(:,:,:,:,ic,1) * ur(:,:,:,:,ihat,1,jc) +
     &                   uisu2(:,:,:,:,ic,2) * ur(:,:,:,:,ihat,2,jc) )
                  elsewhere
                     urprmsu3(:,:,:,:,ic,jc) = ur(:,:,:,:,ihat,ic,jc)
                     uiprmsu3(:,:,:,:,ic,jc) = ui(:,:,:,:,ihat,ic,jc)
                  end where
               end do
            end do

            do jc=1,nc
               urprmsu3(:,:,:,:,3,jc) = ur(:,:,:,:,ihat,3,jc)
               uiprmsu3(:,:,:,:,3,jc) = ui(:,:,:,:,ihat,3,jc)
            end do
c
c     case 2. a_2
c
c     here we calculate the next bit of the product, link*staples.
c     We use the old staples to multiply it with Uprmsu3 = a_1 * U
c     lts = matrixa = [ [SU(2)] 0 ]*lts[ ele in SU(3)    0    ].
c                     [    0    1 ]    [      0          0    ]

            ltsr = 0.0d0
            ltsi = 0.0d0
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     ltsr(:,:,:,:,ic,jc) = ltsr(:,:,:,:,ic,jc) +
     &                   ( urprmsu3(:,:,:,:,ic,kc) * stapler(:,:,:,:,kc,jc) -
     &                     uiprmsu3(:,:,:,:,ic,kc) * staplei(:,:,:,:,kc,jc) )
                     ltsi(:,:,:,:,ic,jc) = ltsi(:,:,:,:,ic,jc) +
     &                  ( urprmsu3(:,:,:,:,ic,kc) * staplei(:,:,:,:,kc,jc) +
     &                     uiprmsu3(:,:,:,:,ic,kc) * stapler(:,:,:,:,kc,jc) )
                  end do
               end do
            end do

c     Here again after having calculated the new ltsr and ltsi ( Uprmsu3*R ) with
c     the old staples R, Uprmsu3 = a_1 * U. We use an offset of1=1 to access the
c     lower block of the ncxnc lts matrix. Then once again we reshape these
c     entries by hardwiring them into a SU(2) matrix called pseudo-heatbath
c     staples namely phbsr and phbsi. The resulting SU(2) matrix is then
c     passed into pseudoheat.

            of1 = 1

            phbsr(:,:,:,:,1,1) = (  ltsr(:,:,:,:,1+of1,1+of1) +
     &                              ltsr(:,:,:,:,2+of1,2+of1) ) / 2.0d0
            phbsr(:,:,:,:,1,2) = (  ltsr(:,:,:,:,1+of1,2+of1) -
     &                              ltsr(:,:,:,:,2+of1,1+of1) ) / 2.0d0
            phbsr(:,:,:,:,2,1) = - phbsr(:,:,:,:,1,2)
            phbsr(:,:,:,:,2,2) =   phbsr(:,:,:,:,1,1)


            phbsi(:,:,:,:,1,1) = (  ltsi(:,:,:,:,1+of1,1+of1) -
     &                              ltsi(:,:,:,:,2+of1,2+of1) ) / 2.0d0
            phbsi(:,:,:,:,1,2) = (  ltsi(:,:,:,:,1+of1,2+of1) +
     &                              ltsi(:,:,:,:,2+of1,1+of1) ) / 2.0d0
            phbsi(:,:,:,:,2,1) =   phbsi(:,:,:,:,1,2)
            phbsi(:,:,:,:,2,2) = - phbsi(:,:,:,:,1,1)

c     we next make a product of the matrices in such a way that link
c     Udblerpm=urdbleprmsu3+iuidbleprmsu3
c             =( br + ibi ) * ( urprmsu3 + iuiprmsu3 )
c             =( br*urprmsu3 - bi*uiprmsu3 ) + i( br*uiprmsu3 + bi*urprmsu3)
c             = a_2 * Uprmsu3
c     Where the a_1 and a_2 have the form of matrixa and matrixb mentioned above.

c     Calling pseudoheat to obtain two new SU(2) matrices with the linkprmsu3*staples
c     calculated above.
c     Allocating by hardwiring to the urnew and uinew which are the return
c     variables of these products.

            call pseudoheat(ursu2,uisu2,phbsr,phbsi,mask,imask,betanew,ihat)

c     Now mapping these variables to the full QCD link, ur and ui. For each
c     direction mu, the main loop imu.

            do ic=2,nc
               do jc=1,nc
                  where( mask(:,:,:,:,ihat,imask) )
```

```
              ur(:,:,:,:,ihat,ic,jc) =
     &          ( ursu2(:,:,:,:,ic-1,1) * urprmsu3(:,:,:,:,2,jc) +
     &            ursu2(:,:,:,:,ic-1,2) * urprmsu3(:,:,:,:,3,jc) -
     &            uisu2(:,:,:,:,ic-1,1) * uiprmsu3(:,:,:,:,2,jc) -
     &            uisu2(:,:,:,:,ic-1,2) * uiprmsu3(:,:,:,:,3,jc) )
              ui(:,:,:,:,ihat,ic,jc) =
     &          ( ursu2(:,:,:,:,ic-1,1) * uiprmsu3(:,:,:,:,2,jc) +
     &            ursu2(:,:,:,:,ic-1,2) * uiprmsu3(:,:,:,:,3,jc) +
     &            uisu2(:,:,:,:,ic-1,1) * urprmsu3(:,:,:,:,2,jc) +
     &            uisu2(:,:,:,:,ic-1,2) * urprmsu3(:,:,:,:,3,jc) )
                elsewhere
                   ur(:,:,:,:,ihat,ic,jc) = urprmsu3(:,:,:,:,ic,jc)
                   ui(:,:,:,:,ihat,ic,jc) = uiprmsu3(:,:,:,:,ic,jc)
                end where
             end do
          end do

          do jc=1,nc
             ur(:,:,:,:,ihat,1,jc) = urprmsu3(:,:,:,:,1,jc)
             ui(:,:,:,:,ihat,1,jc) = uiprmsu3(:,:,:,:,1,jc)
          end do
c
c     case 3. a_3
c
c     forming another SU(2) subgroup of the form
c     These two matrices have the form matrixa = [ x  0   x ]
c                                                [ 0  1   0 ]
c                                                [ x  0   x ]
          ltsr = 0.0d0
          ltsi = 0.0d0
          do ic=1,nc
             do jc=1,nc
                do kc=1,nc
                   ltsr(:,:,:,:,ic,jc) = ltsr(:,:,:,:,ic,jc) +
     &                 ( ur(:,:,:,:,ihat,ic,kc) * stapler(:,:,:,:,kc,jc) -
     &                   ui(:,:,:,:,ihat,ic,kc) * staplei(:,:,:,:,kc,jc) )
                   ltsi(:,:,:,:,ic,jc) = ltsi(:,:,:,:,ic,jc) +
     &                 ( ur(:,:,:,:,ihat,ic,kc) * staplei(:,:,:,:,kc,jc) +
     &                   ui(:,:,:,:,ihat,ic,kc) * stapler(:,:,:,:,kc,jc) )
                end do
             end do
          end do


          phbsr(:,:,:,:,1,1) = (  ltsr(:,:,:,:,1,1) + ltsr(:,:,:,:,3,3) ) / 2.0d0
          phbsr(:,:,:,:,1,2) = (  ltsr(:,:,:,:,1,3) - ltsr(:,:,:,:,3,1) ) / 2.0d0
          phbsr(:,:,:,:,2,1) = - phbsr(:,:,:,:,1,2)
          phbsr(:,:,:,:,2,2) =   phbsr(:,:,:,:,1,1)

          phbsi(:,:,:,:,1,1) = (  ltsi(:,:,:,:,1,1) - ltsi(:,:,:,:,3,3) ) / 2.0d0
          phbsi(:,:,:,:,1,2) = (  ltsi(:,:,:,:,1,3) + ltsi(:,:,:,:,3,1) ) / 2.0d0
          phbsi(:,:,:,:,2,1) =   phbsi(:,:,:,:,1,2)
```

```
            phbsi(:,:,:,:,2,2) = - phbsi(:,:,:,:,1,1)

c     now calling the cooling algorithm.

            call pseudoheat(ursu2,uisu2,phbsr,phbsi,mask,imask,betanew,ihat)

c     We next make a product of the matrices in such a way that
c     link Uprmsu3 = urprmsu3+iuiprmsu3 = ( ar + iai ) * ( ur + iui )
c                                = ( ar*ur - ai*ui ) + i( ar*ui + ai*ur)
c     by hotwiring the matrix indices for optimization.

            do ic=1,nc-1
               do jc=1,nc
                  ic3 = ic
                  if(ic3 == 2 ) ic3 = 3
                  where( mask(:,:,:,:,ihat,imask) )
                     urprmsu3(:,:,:,:,ic3,jc) =
     &                       ( ursu2(:,:,:,:,ic,1) * ur(:,:,:,:,ihat,1,jc) +
     &                         ursu2(:,:,:,:,ic,2) * ur(:,:,:,:,ihat,3,jc) -
     &                         uisu2(:,:,:,:,ic,1) * ui(:,:,:,:,ihat,1,jc) -
     &                         uisu2(:,:,:,:,ic,2) * ui(:,:,:,:,ihat,3,jc) )
                     uiprmsu3(:,:,:,:,ic3,jc) =
     &                       ( ursu2(:,:,:,:,ic,1) * ui(:,:,:,:,ihat,1,jc) +
     &                         ursu2(:,:,:,:,ic,2) * ui(:,:,:,:,ihat,3,jc) +
     &                         uisu2(:,:,:,:,ic,1) * ur(:,:,:,:,ihat,1,jc) +
     &                         uisu2(:,:,:,:,ic,2) * ur(:,:,:,:,ihat,3,jc) )
                  elsewhere
                     urprmsu3(:,:,:,:,ic3,jc) = ur(:,:,:,:,ihat,ic3,jc)
                     uiprmsu3(:,:,:,:,ic3,jc) = ui(:,:,:,:,ihat,ic3,jc)
                  end where
               end do
            end do

            do jc=1,nc
               ur(:,:,:,:,ihat,1,jc) = urprmsu3(:,:,:,:,1,jc)
               ui(:,:,:,:,ihat,1,jc) = uiprmsu3(:,:,:,:,1,jc)
               ur(:,:,:,:,ihat,3,jc) = urprmsu3(:,:,:,:,3,jc)
               ui(:,:,:,:,ihat,3,jc) = uiprmsu3(:,:,:,:,3,jc)
            end do

            end do

         end do
      end do

      return
      end subroutine pseudosweep
```

E.10 Front End of the Gauge Field Generator for Anisotropic Lattices

```
c
c     A program to generate some random configuration in
c     SU(N) non-abelian group here N=3 on an anisotropic lattice.
c     This program contains the timers, the two files for data measurements
c     It fixes every five sweeps, has while loop and fixes every
c     partial sweep in the subroutine pseudosweepani (optional).
c     ---------------------------------------------------------------------
c     SU(3) Gauge configuration maker
c     ---------------------------------------------------------------------
c     Author: F.D.R. Bonnet & D.B. Leinweber
c     supervised by: D.B. Leinweber and A.G. Williams
c
c     starting condition
c     istart = -1    cold start (ordered start, calling initialsu2)
c     istart =  0    hot start  (random start, calling su2random)
c     istart =  1    read in an old configuration
c
c     Output:
c     for the configuration
c     unit 3,unformatted:  nfig,beta,nx,ny,nz,nt,ur,ui
c     for the report
c     unit 1: su2conf.log
c
c     To compile
c
c     cmf -cm5 -vu -nopadding -extend_source -f90syntax aniimpsu3confNTimp.fcm -o
c                           outputfile -lcmsslcm5vu
c     --------------------------------------------------------------------------

      PROGRAM aniimpsu3conf
      IMPLICIT NONE
      include '../GeneralSupport/latticeSize.h'

c     global variables

      integer,parameter                                        :: nc=3     !color
      integer,parameter                                        :: mu=4     !direction
      integer,parameter                                        :: nmask=16 !# of masks

      double precision                                         :: beta
      double precision                                         :: xi0

      double precision,dimension(nx,ny,nz,nt,mu,nc,nc)         :: ur,ui    !link variable
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)
      double precision,dimension(nx,ny,nz,nt)                  :: action
cmf$  layout action(:news,:news,:news,:news)
      logical,dimension(nx,ny,nz,nt,mu,nmask)                  :: mask
```

```
cmf$  layout mask(:news,:news,:news,:news,:serial,:serial)
      logical,dimension(nx,ny,nz,nt,mu,nmask)                 :: maskrect !sweep maskrect
cmf$  layout maskrect(:news,:news,:news,:news,:serial,:serial)

c     local variables

      double precision                                        :: lastPlaq
      double precision     :: uzeroprm=1.0d0, uzeroprmupdate=1.0d0, uzeroprmavg
      double precision     :: uzeros  =1.0d0, uzerosupdate  =1.0d0, uzerosavg
      double precision     :: uzerot  =1.0d0, uzerotupdate  =1.0d0, uzerotavg
      double precision     :: uzero   =1.0d0, uzeroupdate   =0.0d0, uzeroavg
      double precision     :: ut_on_us=1.0d0                          !ratio ut/us
      double precision                                        :: plaqbar,plaqbarAvg
      integer                                                 :: ireport,isweep
      integer                                                 :: istart,itype,iseed
      integer                                                 :: nsweep
      integer                :: nfig,nreport,nfix,u0switch,tmswitch=1,u0tswitch=1
      integer                                                 :: umask
      integer                                                 :: nsub  !# su2sub=nsub*3
      character(len=1)                                        :: nsubfile
      character(len=4)                                        :: confnum
      character(len=80)                                       :: newconfig,lastconfig
      character(len=80)                                       :: datafile

      integer,parameter                                       :: naver=20
      double precision,parameter                              :: naver2=20.0d0
      double precision,dimension(naver)                       :: plqbrvec=0.0d0
cmf$  layout plqbrvec(:serial)
      double precision                                        :: deltaplqbr,stdplqbrvec
      integer                                                 :: iaver
      integer                                                 :: sweep     !sweep counter

      integer                                                 :: strlen

      integer                                                 :: nxold,nyold,nzold,ntold
      double precision                                        :: betaold,xi0old

      INTERFACE
       subroutine ReadLinksani(filename,ur,ui,nfig,beta,nxf,nyf,nzf,ntf,
     &                        lastPlaq,plaqbarAvg,itype,tmswitch,uzeros,uzerot,xi0)
        implicit none
        include '../GeneralSupport/latticeSize.h'
        integer,parameter                                     :: nc=3
        integer,parameter                                     :: mu=4
        double precision,dimension(nx,ny,nz,nt,mu,nc,nc)      :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)
        character(len=80)                                     :: filename
        integer                                               :: nfig,nxf,nyf,nzf,ntf
        integer                                               :: itype,tmswitch
        double precision                                      :: beta,xi0
        double precision                  :: lastPlaq,plaqbarAvg,uzeros,uzerot
       end subroutine ReadLinksani
       subroutine WriteLinksani(filename,ur,ui,nfig,beta,nxf,nyf,nzf,ntf,
     &     lastPlaq,plaqbarAvg,itype,tmswitch,uzeros,uzerot,xi0)
        implicit none
        include '../GeneralSupport/latticeSize.h'
        integer,parameter                                    :: nc=3
        integer,parameter                                    :: mu=4
        character(len=80)                                    :: filename
        integer                                              :: itype,tmswitch
        integer                                              :: nfig,nxf,nyf,nzf,ntf
        double precision                                     :: beta,xi0
        double precision                        :: lastPlaq,plaqbarAvg,uzeros,uzerot
        double precision,dimension(nx,ny,nz,nt,mu,nc,nc)     :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)
       end subroutine WriteLinksani
       subroutine initialu(ur,ui)
        implicit none
        include '../GeneralSupport/latticeSize.h'
        integer,parameter                                    :: nc=3
        integer,parameter                                    :: mu=4
        double precision,dimension(nx,ny,nz,nt,mu,nc,nc)     :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)
       end subroutine initialu
       subroutine su3random(ur,ui)
        implicit none
        include '../GeneralSupport/latticeSize.h'
        integer,parameter                                    :: nc=3
        integer,parameter                                    :: mu=4
        double precision,dimension(nx,ny,nz,nt,mu,nc,nc)     :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)
       end subroutine su3random
       subroutine SetMask(mask, umask, itype)
        implicit none
        include '../GeneralSupport/latticeSize.h'
        integer,parameter                                    :: mu=4
        integer,parameter                                    :: nmask=16
        integer                                              :: umask, itype
        logical,dimension(nx,ny,nz,nt,mu,nmask)              :: mask
cmf$  layout mask(:news,:news,:news,:news,:serial,:serial)
       end subroutine SetMask
       subroutine GetActionani(ur,ui,action,plaqbar,itype,tmswitch,uzeros,uzerot,xi0)
        implicit none
        include '../GeneralSupport/latticeSize.h'
        integer,parameter                                    :: nc=3
        integer,parameter                                    :: mu=4
        integer                                              :: itype,tmswitch
        double precision                                     :: xi0
        double precision                                     :: uzeros,uzerot
        double precision                                     :: plaqbar
        double precision,dimension(nx,ny,nz,nt,mu,nc,nc)      :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)
        double precision,dimension(nx,ny,nz,nt)               :: action
cmf$  layout action(:news,:news,:news,:news)
       end subroutine GetActionani
       subroutine tadpoleimpani(ur,ui,uzero)
        implicit none
        include '../GeneralSupport/latticeSize.h'
        integer,parameter                                     :: nc=3      !sigma,color
        integer,parameter                                     :: mu=4      !direction
        double precision                                      :: uzero
        double precision,dimension(nx,ny,nz,nt,mu,nc,nc)      :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)
       end subroutine tadpoleimpani
       subroutine tadpoleimpspace(ur,ui,uzeros,xi0)
        implicit none
        include '../GeneralSupport/latticeSize.h'
        integer,parameter                                     :: nc=3       !sigma,color
        integer,parameter                                     :: mu=4       !direction
        double precision                                      :: xi0
        double precision                                      :: uzeros
        double precision,dimension(nx,ny,nz,nt,mu,nc,nc)      :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)
       end subroutine tadpoleimpspace
       subroutine tadpoleimptime(ur,ui,uzeros,uzerot,xi0)
        implicit none
        include '../GeneralSupport/latticeSize.h'
        integer,parameter                                     :: nc=3       !sigma,color
        integer,parameter                                     :: mu=4       !direction
        double precision                                      :: xi0
        double precision                                      :: uzeros,uzerot
        double precision,dimension(nx,ny,nz,nt,mu,nc,nc)      :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)
       end subroutine tadpoleimptime
       subroutine fixsu3(ur,ui)
        implicit none
        include '../GeneralSupport/latticeSize.h'
        integer,parameter                                     :: nc=3
        integer,parameter                                     :: mu=4
        double precision,dimension(nx,ny,nz,nt,mu,nc,nc)      :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)
       end subroutine fixsu3
       subroutine pseudosweepani(ur,ui,mask,umask,beta,itype,
     &                          nsub,tmswitch,uzeros,uzerot,xi0)
        implicit none
        include '../GeneralSupport/latticeSize.h'
        integer,parameter                                     :: nc=3
        integer,parameter                                     :: mu=4
        integer,parameter                                     :: nmask=16
        integer                                               :: itype,tmswitch
        integer                                               :: umask
        integer                                               :: nsub
        double precision                                      :: uzeros,uzerot
        double precision                                      :: beta,xi0
        double precision,dimension(nx,ny,nz,nt,mu,nc,nc)      :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)
        logical,dimension(nx,ny,nz,nt,mu,nmask)               :: mask
cmf$  layout mask(:news,:news,:news,:news,:serial,:serial)
       end subroutine pseudosweepani
      END INTERFACE

      write(*,*)
      write(*,*)'Please enter a beta value.'
      read(*,*) beta

      write(*,*) beta

      write(*,*)'Enter the configuration number: '
      read(*,'(a4)') confnum
      write(*,*)

       write(*,*) confnum

      write(*,*)
      write(*,*)'How many times that you would like to wrappe around the'
      write(*,*)'3 su2 subgroups. nsub*3su2=su3'
      read(*,*) nsub

write(*,*) nsub

      write(*,*)
      write(*,*)'please enter the same number as previous entry, for file purposes'
      write(*,*)'3 su2 subgroups. nsub*3su2=su3, '
      read(*,'(a1)') nsubfile

write(*,*) nsubfile

      write(*,*)
      write(*,*)'Please enter a value for xi_0=a_s/a_t, the bare anisotropy.'
      read(*,*) xi0

      write(*,*) xi0

      write(*,*)
      write(*,*)'which Action should we use this time'
      write(*,*)'          0: Pseudo-heatbath algorithm with standard Wilson Action'
      write(*,*)'          1: Pseudo-heatbath algorithm with improved Wilson Action'
      read(*,*) itype
      write(*,*) itype

c       if(itype==1) then
          write(*,*)
          write(*,*)'Would you like to improve in
     &              the time direction as well?. 0=no, 1=yes'
          read(*,*) tmswitch

            write(*,*) tmswitch
c       end if

      write(*,*)
      write(*,*)'Would you like to have u0 updated at report time?. 0=no, 1=yes'
      read(*,*) u0switch

       write(*,*) u0switch

      write(*,*)
      write(*,*)'Would you like to calculate time mean field
     &           improvement factor u0_t?. 0=no, 1=yes'
      read(*,*) u0tswitch

       write(*,*) u0tswitch

      write(*,*)
      write(*,*)'How many sweeps would you like to be done between reports?'
      read(*,*) nreport

       write(*,*) nreport

      write(*,*)
      write(*,*)'How often would you like to reuniterize the group matrix'
      read (*,*) nfix

      write(*,*) nfix

      write(*,*)
      write(*,*)'How many sweeps would you like to be done?,'
      write(*,*)'must be a multiple of the sweep report number.'
      read(*,*) nsweep

       write(*,*) nsweep

      write(*,*)
      write(*,*)'How shall we run this time?'
      write(*,*)'          -1: cold start'
      write(*,*)'           0: hot start'
      write(*,*)'           1: read in from an old configuration'
      read(*,*) istart

      write(*,*) istart
```

```
if (istart .eq. 1) then
   write(*,*)
   write(*,*)'Which file are we reading from?'
   read (*,'(a80)') lastConfig
   write(*,'(a)') lastConfig

endif

write(*,*)
write(*,*)'Please enter a seed number,'
write(*,*)'The Wolfram random number generator cannot be seeded with -1 or 0.'
read(*,*) iseed

 write(*,*) iseed

write(*,*)
write(*,*)'Enter a filename for this new configuration, output file.'
read(*,'(a80)') newconfig

write(*,'(a)') newconfig

call CMF_randomize(iseed)

if( itype == 0 ) then
   open(1,file='aniSu3conf.DoNloop.3su2.log',status='unknown',position='append')
   datafile = 'aniSu3confphb.Do'
   datafile = datafile(1:strlen(datafile))//nsubfile//'loop.3su2.dat.c'//confnum
elseif( itype == 1 ) then
   open(1,file='aniImpSu3conf.Doloop.3su2.log',status='unknown',position='append')
   datafile = 'aniImpSu3confphb.Do'
   datafile = datafile(1:strlen(datafile))//nsubfile//'loop.3su2.dat.c'//confnum
end if

write(1,*)
write(1,*)'=================================================================='
write(1,*)'    SU(3) Gauge Configuration Maker on an Anisotropic Lattice    '
write(1,*)'=================================================================='

write(1,*)
write(1,*)'******************************************************************'
if(itype==0) then
   write(1,*)'   Using a pseudo-heatbath algorithm with standard Wilson Action   '
elseif(itype==1) then
   write(1,*)'   Using a pseudo-heatbath algorithm with improved Wilson Aciton   '
end if
write(1,*)'******************************************************************'
write(1,*)
write(1,'(a,i3,a,i3,a,i3,a,i3)')'lattice size = ',nx,'x',ny,'x',nz,'x',nt
write(1,'(a,i3)')'PHB, number of types -1:cold start, 0:hot start.',istart
write(1,'(a,i4)')'time improvement:  tmswitch,0=off,1=on,=',tmswitch
write(1,'(a,i4)')'u_0  calculation:  u0switch,0=off,1=on,=',u0switch
```

```
      write(1,'(a,i4)')'u_0t calculation: u0tswitch,0=off,1=on,=',u0tswitch
      write(1,'(a,i4)')'no of sweeps :',nsweep
      write(1,'(a,i4)')'no of sweeps between reports:',nreport
      write(1,'(a,i10)')'The starting seed was:',iseed
      write(1,'(a,i4)')'no of times the group matrix are reuniterized:',nfix
      if(istart .eq. -1) then
         nfig = 0
         write(1,*)'starting with a cold start'
         call initialu(ur,ui)
         call GetActionani(ur,ui,action,plaqbar,itype,tmswitch,uzeros,uzerot,xi0)
      elseif(istart .eq. 0) then
         nfig = 0
         write(1,*)'starting with a hot start'
         call initialu(ur,ui)
         call su3random(ur,ui)
         call GetActionani(ur,ui,action,plaqbar,itype,tmswitch,uzeros,uzerot,xi0)
      elseif(istart .eq. 1) then

         call ReadLinksani(lastconfig,ur,ui,nfig,betaold,nxold,
     &        nyold,nzold,ntold,lastPlaq,plaqbarAvg,
     &        itype,tmswitch,uzeros,uzerot,xi0old)

         write(1,'(a,i3,2x,a,2x,a)')'for configuration number',
     &                              nfig,'the input file =',lastconfig

         nfig = nfig + 1

         if (beta.ne.betaold) pause 'mismatch in beta'
         if (xi0.ne.xi0old) pause 'mismatch in xi0'
         if (nx.ne.nxold) pause 'mismatch in nx'
         if (ny.ne.nyold) pause 'mismatch in ny'
         if (nz.ne.nzold) pause 'mismatch in nz'
         if (nt.ne.ntold) pause 'mismatch in nt'

      end if

      write(1,*)'printing the average plaquette from Pseudo-heatbath algo routine'
      write(1,'(a,f4.2,2x,a,f6.4)')'for beta = ',beta,
     &                             'and with bare anysotropy xi_0 = ',xi0
      write(1,'(3x,a,f12.8)')'calling GetActionani, untouched action, plaqbar = ',plaqbar
      write(1,'(a,f12.8)')'starting moving avg. mean link          (uzero)  = ',uzero
      write(1,'(a,f12.8)')'starting moving avg. mean link spatial  (uzeros) = ',uzeros
      write(1,'(a,f12.8)')'starting moving avg. mean link temporal (uzerot) = ',uzerot
      write(1,'(a,f12.8)')'starting moving avg. mean link ratio   (u_t/u_s) = ',ut_on_us
      write(1,'(2x,a,4x,a,9x,a,4x,a,6x,a,8x,a,8x,a,8x,a,8x,a)')
     &          'sweep','plaqbar','plaqbaravg','deltaplqbr',
     &     'uzeros','uzerot','uzeroprm','uzero','ut/us'
      write(*,'(2x,a,4x,a,9x,a,4x,a,6x,a,8x,a,8x,a,8x,a,8x,a)')
     &     'sweep','plaqbar','plaqbaravg','deltaplqbr',
     &     'uzeros','uzerot','uzeroprm','uzero','ut/us'

c     calling the mask subroutine, to be passed into metropolis,
c     pseudoheat as well as pseudosweepani

      call SetMask(mask, umask, itype)

      open(4,file='phbAniimpsu3confNTimp.dat',status='unknown',position='append')
      open(5,file='su3avebeta.dat',status='unknown',position='append')

c     calling the subroutine pseudoseep,n times, nthermphb
c     these are the thermalisation sweeps using a pseudo-heatbath algo

      sweep = 0
      do isweep=1,nsweep/nreport
         uzerosavg = 0.0d0
         uzerotavg = 0.0d0
         uzeroavg  = 0.0d0
         uzeroprmavg = 0.0d0
         do ireport=1,nreport
            sweep = sweep + 1
            call pseudosweepani(ur,ui,mask,umask,beta,itype,
     &                          nsub,tmswitch,uzeros,uzerot,xi0)
            if(mod(sweep,nfix).eq.0) call fixsu3(ur,ui)
            if(u0switch==1) then
               call tadpoleimpspace(ur,ui,uzerosupdate,xi0)
               if( u0tswitch == 1 ) then
                  call tadpoleimptime(ur,ui,uzerosupdate,uzerotupdate,xi0)
               end if
               if( xi0 == 1.0d0 ) then
                  call tadpoleimpani(ur,ui,uzeroupdate)
                  uzeroprmavg  = uzeroprmavg +
     &                 ( ( uzerosupdate**4 + (uzerosupdate**2) * (uzerotupdate**2) )
     &                     / 2.0d0 ) ** (0.25d0)
                  write(*,*)'uzerosupdate,uzerotupdate,uzeroprm,uzeroupdate'
                  write(*,'(4f20.15)') uzerosupdate,uzerotupdate,
     &                 ( ( uzerosupdate**4 + (uzerosupdate**2) * (uzerotupdate**2) )
     &                     / 2.0d0 ) ** (0.25d0),uzeroupdate
                  uzerosupdate = uzeroupdate
                  uzerotupdate = uzeroupdate
               end if
               uzerosavg = uzerosavg + uzerosupdate
               uzerotavg = uzerotavg + uzerotupdate
               uzeroavg  =  uzeroavg +  uzeroupdate
            end if
         end do
         if(u0switch==1) then
            uzeros = uzerosavg / nreport
            uzerot = uzerotavg / nreport
            uzero  =  uzeroavg / nreport
            uzeroprm  =  uzeroprmavg / nreport
         end if
         ut_on_us = uzerot / uzeros
         call GetActionani(ur,ui,action,plaqbar,itype,tmswitch,uzeros,uzerot,xi0)
         do iaver = 1, naver-1
```

```
           plqbrvec(iaver) = plqbrvec(iaver + 1)
         end do
         plqbrvec(naver) = plaqbar
         plaqbarAvg = sum(plqbrvec) / naver2
         stdplqbrvec = sqrt( sum( (plqbrvec - plaqbarAvg)**2 ) / (naver2 - 1.0d0) )
         deltaplqbr  = stdplqbrvec / sqrt(naver2)
         write(*,'(2x,i5,2x,f12.8,4x,f12.8,6(2x,f12.8))') sweep,plaqbar,
     &        plaqbarAvg,deltaplqbr,uzeros,uzerot,uzeroprm,uzero,ut_on_us
         write(1,'(2x,i5,2x,f12.8,4x,f12.8,6(2x,f12.8))') sweep,plaqbar,
     &        plaqbarAvg,deltaplqbr,uzeros,uzerot,uzeroprm,uzero,ut_on_us
         write(4,'(2x,i5,2x,f12.8,4x,f12.8,2x,f12.8)')sweep,plaqbar,plaqbarAvg,deltaplqbr
      end do

      write(*,'(f12.8)') plqbrvec
      write(1,'(a,i3,a,i2,a,f12.8)')'after:',nsweep,
     &                             ' sweeps the average over the last ',naver,
     &     ' ending average plaquette is = ',plaqbarAvg
      write(1,'(a,f12.8)')'with statistical error of ',deltaplqbr
      write(5,'(2x,f5.2,3(2x,f25.8))') beta,plaqbarAvg,stdplqbrvec,deltaplqbr

      call WriteLinksani(newconfig,ur,ui,nfig,beta,nx,ny,nz,nt,lastPlaq,plaqbarAvg,
     &                   itype,tmswitch,uzeros,uzerot,xi0)

      write(1,*)
      write(1,'(a,i3,2x,a,2x,a)')'for configuration number',nfig,
     &                           'the output file =',newconfig
      write(1,'(a,f12.8)')'Ending average plaquette is = ',plaqbar
      write(1,'(a,f12.8)')'Ending moving avg. plaq. is = ',plaqbarAvg
      write(1,'(a,f12.8)')'Ending moving avg. mean link         (uzero) = ',uzero
      write(1,'(a,f12.8)')'Ending moving avg. mean link spatial (uzeros)= ',uzeros
      write(1,'(a,f12.8)')'Ending moving avg. mean link temporal(uzerot)= ',uzerot

      close(1)
      close(4)
      close(5)

      end program aniimpsu3conf
```

E.11 Anisotropic Version for Constructing $SU_c(3)$ Matrices

```
c
c     cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Author: Frederic D.R. Bonnet & D. B. Leinweber: 12th of August 1999.
c     subroutine pseudosweepani updates the current link using a psedo-heatbath
c     algorithm. It calls the staplesani and pseudoheat.
c
      subroutine pseudosweepani(ur,ui,mask,umask,beta,itype,
     &                          nsub,tmswitch,uzeros,uzerot,xi0)
      implicit none
      include '../GeneralSupport/latticeSize.h'

c     global variables

      integer,parameter                                     :: nc=3,ncsu2=2
      integer,parameter                                     :: nsigma=ncsu2*ncsu2
      integer,parameter                                     :: mu=4
      integer,parameter                                     :: nmask=16

      integer                                               :: itype,tmswitch
      integer                                               :: umask
      integer                                               :: nsub

      double precision                                      :: uzeros,uzerot
      double precision                                      :: beta,xi0

      double precision,dimension(nx,ny,nz,nt,mu,nc,nc)      :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)
      logical,dimension(nx,ny,nz,nt,mu,nmask)               :: mask
cmf$  layout mask(:news,:news,:news,:news,:serial,:serial)

c     local variables

      double precision,dimension(nx,ny,nz,nt,nc,nc)         :: stapler,staplei
      double precision,dimension(nx,ny,nz,nt,nc,nc)         :: ltsr,ltsi
      double precision,dimension(nx,ny,nz,nt,nc,nc)         :: urprmsu3,uiprmsu3
cmf$  layout stapler(:news,:news,:news,:news,:serial,:serial)
cmf$  layout staplei(:news,:news,:news,:news,:serial,:serial)
cmf$  layout ltsr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout ltsi(:news,:news,:news,:news,:serial,:serial)
cmf$  layout urprmsu3(:news,:news,:news,:news,:serial,:serial)
cmf$  layout uiprmsu3(:news,:news,:news,:news,:serial,:serial)
      double precision,dimension(nx,ny,nz,nt,ncsu2,ncsu2)   :: phbsr,phbsi
      double precision,dimension(nx,ny,nz,nt,ncsu2,ncsu2)   :: ursu2,uisu2
cmf$  layout phbsr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout phbsi(:news,:news,:news,:news,:serial,:serial)
cmf$  layout ursu2(:news,:news,:news,:news,:serial,:serial)
cmf$  layout uisu2(:news,:news,:news,:news,:serial,:serial)

      double precision                                      :: betanew
      integer                                               :: of1
      integer                                               :: imask,ihat,ic,jc,kc,ic3
      integer                                               :: isub

      interface
         SUBROUTINE staplesani(ur,ui,stapler,staplei,xhat,local,
     &                     itype,tmswitch,uzeros,uzerot,xi0)
           IMPLICIT NONE
           include '../GeneralSupport/latticeSize.h'
           integer,parameter                                :: nc=3
           integer,parameter                                :: mu=4
```

```
          integer                                           :: xhat
          integer                                           :: itype,tmswitch
          double precision                                  :: xi0
          double precision                                  :: uzeros,uzerot
          logical                                           :: local
          double precision,dimension(nx,ny,nz,nt,nc,nc)     :: stapler,staplei
cmf$  layout stapler(:news,:news,:news,:news,:serial,:serial)
cmf$  layout staplei(:news,:news,:news,:news,:serial,:serial)
          double precision,dimension(nx,ny,nz,nt,mu,nc,nc)  :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)
         end SUBROUTINE staplesani
         subroutine pseudoheat(urnewsu2,uinewsu2,phbsr,phbsi,mask,imask,beta,ihat)
          implicit none
          include '../GeneralSupport/latticeSize.h'
          integer,parameter                                 :: ncsu2=2
          integer,parameter                                 :: nc=3
          integer,parameter                                 :: mu=4
          integer,parameter                                 :: nmask=16
          double precision                                  :: beta
          integer                                           :: ihat,imask
          double precision,dimension(nx,ny,nz,nt,ncsu2,ncsu2) :: phbsr,phbsi
          double precision,dimension(nx,ny,nz,nt,ncsu2,ncsu2) :: urnewsu2,uinewsu2
cmf$  layout phbsr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout phbsi(:news,:news,:news,:news,:serial,:serial)
cmf$  layout urnewsu2(:news,:news,:news,:news,:serial,:serial)
cmf$  layout uinewsu2(:news,:news,:news,:news,:serial,:serial)
          logical,dimension(nx,ny,nz,nt,mu,nmask)           :: mask
cmf$  layout mask(:news,:news,:news,:news,:serial,:serial)
         end subroutine pseudoheat
      end interface

c     starts of the running commands

c     Here we need to multiply Beta by a factor of 2/nc=3 because for the su2 case
c     there is a factor of which cancels the one 1/ncsu2 in the probability
c     distribution function. This beta value is to be passed into pseudoheat

      betanew = ( 2.0d0 / nc ) * beta
c      betanew = ( 2.0d0 / nc ) * beta * uzeros**4

c     the ihat do loop, is to loop over all the possible Euclidean directions.
c     The imask do loop, is to ensure that the entire lattice is considered:
c     all true then all false.

      do ihat=1,mu
         do imask=1,umask

c     calculate the staple in the ihat direction

            call staplesani(ur,ui,stapler,staplei,ihat,.true.,
     &                      itype,tmswitch,uzeros,uzerot,xi0)
```

```
            do isub=1,nsub
c
c     case 1. a_1
c
c     The Wilson action can be written as S(U)=Sum_p(Re(U*U_p))+constant
c                                              =Re(Tr(U*R))+constant
c     R is just the sum over the six plaquette namely the staplesani: stapler,staplei.

c     now we need to calculate the product of the staplesani time the link variable in
c     each direction U*R=ltsr+iltsi=( ur + iui ) * ( stapler + istaplei )
c                                 =(ur*stapler - ui*staplei)+i(ur*staplei + ui*staplei)
            ltsr = 0.0d0
            ltsi = 0.0d0
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     ltsr(:,:,:,:,ic,jc) = ltsr(:,:,:,:,ic,jc) +
     &                   ( ur(:,:,:,:,ihat,ic,kc) * stapler(:,:,:,:,kc,jc) -
     &                     ui(:,:,:,:,ihat,ic,kc) * staplei(:,:,:,:,kc,jc) )
                     ltsi(:,:,:,:,ic,jc) = ltsi(:,:,:,:,ic,jc) +
     &                   ( ur(:,:,:,:,ihat,ic,kc) * staplei(:,:,:,:,kc,jc) +
     &                     ui(:,:,:,:,ihat,ic,kc) * stapler(:,:,:,:,kc,jc) )
                  end do
               end do
            end do

c     The distribution we are interested in the following
c     dP(a_k) = d^{(k)}a_k * exp[-beta * S(a_k*U^{(k-1)})]/Z_k(U^{(k-1)})
c     where d^{(k)}a_k is the Haar measure on the SU(2)_k and
c     Z_k(U^{(k-1)}) = \int_{SU(2)_k} da exp[-beta * S(aU)]. If a{\in}SU(2)_K then
c     Z_k(aU) = Z_k(U). In order to generate the above distribution for the minimal
c     set F of SU(2) subgroup of SU(N) namely {F:SU(2)_k, k=1,..,nc-1}, let's first
c     note that S(a_k*U)=Re(Tr a_k * U*R) = Re(Tr \alpha_k * r_k)
c                                        + terms independent of \alpha_k.
c     Where r_k is the 2x2 subgroup of the U*R (i.e. link*staplesani : lts), this matix
c     has the same block structure as the matrix a_k. This means that \alpha_k and
c     r_k = r_0 * I + i \vec{\sigma}\dot\vec{r} are the same block in the ncxnc
c     matrix located at the (k,k+1)th rows and column.

c     For the first call of the pseudo-heatbath we need to match the matrix block,
c     if this is not done wrong information will be sent to the subroutine therefore
c     returning the wrong numbers. To do that we use an offset parameter called of1
c     with this of1 we can acces the correct entries of the ncxnc lts matrix. We then
c     reconstruct a SU(2) matrix that can be passed into pseudoheat which will also
c     return a SU(2) matrix.

            of1 = 0

            phbsr(:,:,:,:,1,1) =
     &           ( ltsr(:,:,:,:,1+of1,1+of1) + ltsr(:,:,:,:,2+of1,2+of1) ) / 2.0d0
            phbsr(:,:,:,:,2,2) =   phbsr(:,:,:,:,1,1)
            phbsr(:,:,:,:,1,2) =
     &           ( ltsr(:,:,:,:,1+of1,2+of1) - ltsr(:,:,:,:,2+of1,1+of1) ) / 2.0d0
            phbsr(:,:,:,:,2,1) = - phbsr(:,:,:,:,1,2)

            phbsi(:,:,:,:,1,1) =
     &           ( ltsi(:,:,:,:,1+of1,1+of1) - ltsi(:,:,:,:,2+of1,2+of1) ) / 2.0d0
            phbsi(:,:,:,:,1,2) =
     &           ( ltsi(:,:,:,:,1+of1,2+of1) + ltsi(:,:,:,:,2+of1,1+of1) ) / 2.0d0
            phbsi(:,:,:,:,2,1) =   phbsi(:,:,:,:,1,2)
            phbsi(:,:,:,:,2,2) = - phbsi(:,:,:,:,1,1)

c     now calling the pseudo-heatbath algorithm to update, where the
c     mask is .true., the full QCD configuration

c     we call pseudoheat to form 2 SU(3) matrices.
c     These two matrices have the form matrixa = [ [SU(2)] 0 ],  matrixb[ 1    0    ].
c                                                [    0    1 ],         [ 0 [SU(2)] ]
c     Where [SU(2)] is hotwired via ursu2,uisu2 and ursu2,uisu2.

            call pseudoheat(ursu2,uisu2,phbsr,phbsi,mask,imask,betanew,ihat)

c     We next make a product of the matrices in such a way that
c     link Uprmsu3 = urprmsu3+iuiprmsu3 = ( ar + iai ) * ( ur + iui )
c                                 = ( ar*ur - ai*ui ) + i( ar*ui + ai*ur)
c     by hotwiring the matrix indices for optimization.

            do ic=1,nc-1
               do jc=1,nc
                  where( mask(:,:,:,:,ihat,imask) )
              urprmsu3(:,:,:,:,ic,jc) = ( ursu2(:,:,:,:,ic,1) * ur(:,:,:,:,ihat,1,jc) +
     &                                    ursu2(:,:,:,:,ic,2) * ur(:,:,:,:,ihat,2,jc) -
     &                                    uisu2(:,:,:,:,ic,1) * ui(:,:,:,:,ihat,1,jc) -
     &                                    uisu2(:,:,:,:,ic,2) * ui(:,:,:,:,ihat,2,jc) )
              uiprmsu3(:,:,:,:,ic,jc) = ( ursu2(:,:,:,:,ic,1) * ui(:,:,:,:,ihat,1,jc) +
     &                                    ursu2(:,:,:,:,ic,2) * ui(:,:,:,:,ihat,2,jc) +
     &                                    uisu2(:,:,:,:,ic,1) * ur(:,:,:,:,ihat,1,jc) +
     &                                    uisu2(:,:,:,:,ic,2) * ur(:,:,:,:,ihat,2,jc) )
                  elsewhere
                     urprmsu3(:,:,:,:,ic,jc) = ur(:,:,:,:,ihat,ic,jc)
                     uiprmsu3(:,:,:,:,ic,jc) = ui(:,:,:,:,ihat,ic,jc)
                  end where
               end do
            end do

            do jc=1,nc
               urprmsu3(:,:,:,:,3,jc) = ur(:,:,:,:,ihat,3,jc)
               uiprmsu3(:,:,:,:,3,jc) = ui(:,:,:,:,ihat,3,jc)
            end do
c
c     case 2. a_2
c
c     here we calculate the next bit of the product, link*staplesani.
```

```
c     We use the old staplesani to multiply it with Uprmsu3 = a_1 * U
c     lts = matrixa = [ [SU(2)] 0 ]*lts[ ele in SU(3)    0    ].
c                     [    0    1 ]    [      0          0    ]

            ltsr = 0.0d0
            ltsi = 0.0d0
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     ltsr(:,:,:,:,ic,jc) = ltsr(:,:,:,:,ic,jc) +
     &                  ( urprmsu3(:,:,:,:,ic,kc) * stapler(:,:,:,:,kc,jc) -
     &                    uiprmsu3(:,:,:,:,ic,kc) * staplei(:,:,:,:,kc,jc) )
                     ltsi(:,:,:,:,ic,jc) = ltsi(:,:,:,:,ic,jc) +
     &                  ( urprmsu3(:,:,:,:,ic,kc) * staplei(:,:,:,:,kc,jc) +
     &                    uiprmsu3(:,:,:,:,ic,kc) * stapler(:,:,:,:,kc,jc) )
                  end do
               end do
            end do

c     Here again after having calculated the new ltsr and ltsi ( Uprmsu3*R ) with
c     the old staplesani R, Uprmsu3 = a_1 * U. We use an offset of1=1 to access the
c     lower block of the ncxnc lts matrix. Then once again we reshape these
c     entries by hardwiring them into a SU(2) matrix called pseudo-heatbath
c     staplesani namely phbsr and phbsi. The resulting SU(2) matrix is then
c     passed into pseudoheat.

            of1 = 1

            phbsr(:,:,:,:,1,1) =
     &          ( ltsr(:,:,:,:,1+of1,1+of1) + ltsr(:,:,:,:,2+of1,2+of1) ) / 2.0d0
            phbsr(:,:,:,:,2,2) =   phbsr(:,:,:,:,1,1)
            phbsr(:,:,:,:,1,2) =
     &          ( ltsr(:,:,:,:,1+of1,2+of1) - ltsr(:,:,:,:,2+of1,1+of1) ) / 2.0d0
            phbsr(:,:,:,:,2,1) = - phbsr(:,:,:,:,1,2)


            phbsi(:,:,:,:,1,1) =
     &          ( ltsi(:,:,:,:,1+of1,1+of1) - ltsi(:,:,:,:,2+of1,2+of1) ) / 2.0d0
            phbsi(:,:,:,:,1,2) =
     &          ( ltsi(:,:,:,:,1+of1,2+of1) + ltsi(:,:,:,:,2+of1,1+of1) ) / 2.0d0
            phbsi(:,:,:,:,2,1) =   phbsi(:,:,:,:,1,2)
            phbsi(:,:,:,:,2,2) = - phbsi(:,:,:,:,1,1)

c     we next make a product of the matrices in such a way that link
c     Udblerpm=urdbleprmsu3+iuidbleprmsu3
c             =( br + ibi ) * ( urprmsu3 + iuiprmsu3 )
c             =( br*urprmsu3 - bi*uiprmsu3 ) + i( br*uiprmsu3 + bi*urprmsu3)
c             = a_2 * Uprmsu3
c     Where the a_1 and a_2 have the form of matrixa and matrixb mentioned above.

c     Calling pseudoheat to obtain two new SU(2) matrices with the linkprmsu3*staplesani
c     calculated above.
```

```
c     Allocating by hardwiring to the urnew and uinew which are the return
c     variables of these products.

            call pseudoheat(ursu2,uisu2,phbsr,phbsi,mask,imask,betanew,ihat)

c     Now mapping these variables to the full QCD link, ur and ui. For each
c     direction mu, the main loop imu.

            do ic=2,nc
               do jc=1,nc
                  where( mask(:,:,:,:,ihat,imask) )
              ur(:,:,:,:,ihat,ic,jc) = ( ursu2(:,:,:,:,ic-1,1) * urprmsu3(:,:,:,:,2,jc) +
     &                                   ursu2(:,:,:,:,ic-1,2) * urprmsu3(:,:,:,:,3,jc) -
     &                                   uisu2(:,:,:,:,ic-1,1) * uiprmsu3(:,:,:,:,2,jc) -
     &                                   uisu2(:,:,:,:,ic-1,2) * uiprmsu3(:,:,:,:,3,jc) )
              ui(:,:,:,:,ihat,ic,jc) = ( ursu2(:,:,:,:,ic-1,1) * uiprmsu3(:,:,:,:,2,jc) +
     &                                   ursu2(:,:,:,:,ic-1,2) * uiprmsu3(:,:,:,:,3,jc) +
     &                                   uisu2(:,:,:,:,ic-1,1) * urprmsu3(:,:,:,:,2,jc) +
     &                                   uisu2(:,:,:,:,ic-1,2) * urprmsu3(:,:,:,:,3,jc) )
                  elsewhere
                     ur(:,:,:,:,ihat,ic,jc) = urprmsu3(:,:,:,:,ic,jc)
                     ui(:,:,:,:,ihat,ic,jc) = uiprmsu3(:,:,:,:,ic,jc)
                  end where
               end do
            end do

            do jc=1,nc
               ur(:,:,:,:,ihat,1,jc) = urprmsu3(:,:,:,:,1,jc)
               ui(:,:,:,:,ihat,1,jc) = uiprmsu3(:,:,:,:,1,jc)
            end do
c
c     case 3. a_3
c
c     forming another SU(2) subgroup of the form
c     These two matrices have the form matrixa = [ x  0   x ]
c                                                [ 0  1   0 ]
c                                                [ x  0   x ]
            ltsr = 0.0d0
            ltsi = 0.0d0
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     ltsr(:,:,:,:,ic,jc) = ltsr(:,:,:,:,ic,jc) +
     &                   ( ur(:,:,:,:,ihat,ic,kc) * stapler(:,:,:,:,kc,jc) -
     &                     ui(:,:,:,:,ihat,ic,kc) * staplei(:,:,:,:,kc,jc) )
                     ltsi(:,:,:,:,ic,jc) = ltsi(:,:,:,:,ic,jc) +
     &                   ( ur(:,:,:,:,ihat,ic,kc) * staplei(:,:,:,:,kc,jc) +
     &                     ui(:,:,:,:,ihat,ic,kc) * stapler(:,:,:,:,kc,jc) )
                  end do
               end do
            end do
```

```
         phbsr(:,:,:,:,1,1) =
     &           ( ltsr(:,:,:,:,1,1) + ltsr(:,:,:,:,3,3) ) / 2.0d0
         phbsr(:,:,:,:,2,2) =   phbsr(:,:,:,:,1,1)
         phbsr(:,:,:,:,1,2) =
     &           ( ltsr(:,:,:,:,1,3) - ltsr(:,:,:,:,3,1) ) / 2.0d0
         phbsr(:,:,:,:,2,1) = - phbsr(:,:,:,:,1,2)

         phbsi(:,:,:,:,1,1) =
     &           ( ltsi(:,:,:,:,1,1) - ltsi(:,:,:,:,3,3) ) / 2.0d0
         phbsi(:,:,:,:,1,2) =
     &           ( ltsi(:,:,:,:,1,3) + ltsi(:,:,:,:,3,1) ) / 2.0d0
         phbsi(:,:,:,:,2,1) =   phbsi(:,:,:,:,1,2)
         phbsi(:,:,:,:,2,2) = - phbsi(:,:,:,:,1,1)

c     now calling the cooling algorithm.

         call pseudoheat(ursu2,uisu2,phbsr,phbsi,mask,imask,betanew,ihat)

c     We next make a product of the matrices in such a way that
c     link Uprmsu3 = urprmsu3+iuiprmsu3 = ( ar + iai ) * ( ur + iui )
c                                 = ( ar*ur - ai*ui ) + i( ar*ui + ai*ur)
c     by hotwiring the matrix indices for optimization.

         do ic=1,nc-1
            do jc=1,nc
               ic3 = ic
               if(ic3 == 2 ) ic3 = 3
               where( mask(:,:,:,:,ihat,imask) )
                  urprmsu3(:,:,:,:,ic3,jc) =
     &                  ( ursu2(:,:,:,:,ic,1) * ur(:,:,:,:,ihat,1,jc) +
     &                    ursu2(:,:,:,:,ic,2) * ur(:,:,:,:,ihat,3,jc) -
     &                    uisu2(:,:,:,:,ic,1) * ui(:,:,:,:,ihat,1,jc) -
     &                    uisu2(:,:,:,:,ic,2) * ui(:,:,:,:,ihat,3,jc) )
                  uiprmsu3(:,:,:,:,ic3,jc) =
     &                  ( ursu2(:,:,:,:,ic,1) * ui(:,:,:,:,ihat,1,jc) +
     &                    ursu2(:,:,:,:,ic,2) * ui(:,:,:,:,ihat,3,jc) +
     &                    uisu2(:,:,:,:,ic,1) * ur(:,:,:,:,ihat,1,jc) +
     &                    uisu2(:,:,:,:,ic,2) * ur(:,:,:,:,ihat,3,jc) )
               elsewhere
                  urprmsu3(:,:,:,:,ic3,jc) = ur(:,:,:,:,ihat,ic3,jc)
                  uiprmsu3(:,:,:,:,ic3,jc) = ui(:,:,:,:,ihat,ic3,jc)
               end where
            end do
         end do

         do jc=1,nc
            ur(:,:,:,:,ihat,1,jc) = urprmsu3(:,:,:,:,1,jc)
            ui(:,:,:,:,ihat,1,jc) = uiprmsu3(:,:,:,:,1,jc)
            ur(:,:,:,:,ihat,3,jc) = urprmsu3(:,:,:,:,3,jc)
            ui(:,:,:,:,ihat,3,jc) = uiprmsu3(:,:,:,:,3,jc)
         end do
```

```
          end do

       end do
    end do

    return
    end subroutine pseudosweepani
```

E.12 The Anisotropic Version for the Squares Routine

```
c
c     cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Author: Frederic D.R. Bonnet & Derek B. Leinweber date: Ausgust 1999.
c     computes the product of links for the action associated with a link in the
c     xhat direction.
c
      subroutine squaresani(ur,ui,squarer,squarei,xhat,local,itype,
     &                       tmswitch,uzeros,uzerot,xi0)
      implicit none
      include '../../GeneralSupport/latticeSize.h'

c     global variable

      integer,parameter                                        :: nc=3
      integer,parameter                                        :: mu=4

      integer                                                  :: xhat,itype,tmswitch
      logical                                                  :: local
      double precision                                         :: uzeros,uzerot,xi0

      double precision,dimension(nx,ny,nz,nt,nc,nc)            :: squarer,squarei
cmf$  layout squarer(:news,:news,:news,:news,:serial,:serial)
cmf$  layout squarei(:news,:news,:news,:news,:serial,:serial)
      double precision,dimension(nx,ny,nz,nt,mu,nc,nc)         :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)

c     local variables, temporary product variables

      double precision                                         :: stfact,factorsq
      integer,dimension(3)                                     :: yhat
cmf$  layout yhat(:serial)
      double precision,dimension(nx,ny,nz,nt,nc,nc)            :: tr,ti,tsr,tsi
cmf$  layout tr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout ti(:news,:news,:news,:news,:serial,:serial)
cmf$  layout tsr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout tsi(:news,:news,:news,:news,:serial,:serial)
      double precision,dimension(nx,ny,nz,nt,nc,nc)            :: usqr,usqi
```

```
cmf$  layout usqr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout usqi(:news,:news,:news,:news,:serial,:serial)

      integer                                                  :: muloop
      integer                                                  :: ic,jc,kc,imu,i

c     starting of the execution commands

c     setting up the yhat array
c     the yhat(1)=yhat,yhat(2)=zhat and yhat(3)=that when xhat eq 1
c     the yhat(1)=zhat,yhat(2)=that and yhat(3)=xhat when xhat eq 2
c     the yhat(1)=that,yhat(2)=xhat and yhat(3)=yhat when xhat eq 3
c     the yhat(1)=xhat,yhat(2)=yhat and yhat(3)=zhat when xhat eq 4

      if( xhat == 1 ) then
         yhat(1) = 2
         yhat(2) = 3
         yhat(3) = 4
      elseif( xhat == 2 ) then
         yhat(1) = 1
         yhat(2) = 3
         yhat(3) = 4
      elseif( xhat == 3 ) then
         yhat(1) = 1
         yhat(2) = 2
         yhat(3) = 4
      elseif( xhat == 4 ) then
         yhat(1) = 1
         yhat(2) = 2
         yhat(3) = 3
      end if

      if( xi0 < 0.0d0 ) then
         xi0 = 1.0d0
         muloop = mu - 2
      else
         muloop = mu - 1
      end if

      if( itype == 1 ) then
         if( tmswitch == 0 ) then
            stfact = 4.0d0 / 3.0d0
         elseif( tmswitch == 1 ) then
            stfact = 5.0d0 / 3.0d0
         end if
      elseif( itype == 0 ) then
         stfact = 5.0d0 / 3.0d0
      end if

      squarer = 0.0d0
      squarei = 0.0d0
```

```
c     calculation of the link products in the positive plaquette, POSITIVE a x a.

      do i=1,muloop                        !loop over the yhat array

c     putting the factors for the anisotropy lattices. xi0 = a_s/a_t

         if( xhat /= 4 ) then
            if( yhat(i) /= 4 ) then
               factorsq = ( 1.0d0 / ( xi0 * uzeros**4 ) ) * ( 5.0d0 / 3.0d0 )
            elseif( yhat(i) == 4 ) then
               factorsq = ( xi0 / ( ( uzeros**2 ) * ( uzerot**2 ) ) ) * stfact
            end if
         elseif( xhat == 4 ) then
            factorsq = ( xi0 / ( ( uzeros**2 ) * ( uzerot**2 ) ) ) * stfact
         end if

         tr = 0.0d0
         ti = 0.0d0
         tsr = cshift(ur(:,:,:,:,yhat(i),:,:),dim=xhat,shift=1)
         tsi = cshift(ui(:,:,:,:,yhat(i),:,:),dim=xhat,shift=1)
         usqr = cshift(ur(:,:,:,:,xhat,:,:),dim=yhat(i),shift=1)
         usqi = cshift(ui(:,:,:,:,xhat,:,:),dim=yhat(i),shift=1)
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  tr(:,:,:,:,ic,jc) = tr(:,:,:,:,ic,jc)                +
     &              ( tsr(:,:,:,:,ic,kc) * usqr(:,:,:,:,jc,kc) +
     &                tsi(:,:,:,:,ic,kc) * usqi(:,:,:,:,jc,kc) )
                  ti(:,:,:,:,ic,jc) = ti(:,:,:,:,ic,jc)                +
     &              ( tsi(:,:,:,:,ic,kc) * usqr(:,:,:,:,jc,kc) -
     &                tsr(:,:,:,:,ic,kc) * usqi(:,:,:,:,jc,kc) )
               end do
            end do
         end do
         tr = factorsq * tr
         ti = factorsq * ti
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  squarer(:,:,:,:,ic,jc) = squarer(:,:,:,:,ic,jc)        +
     &                   ( tr(:,:,:,:,ic,kc) * ur(:,:,:,:,yhat(i),jc,kc)  +
     &                     ti(:,:,:,:,ic,kc) * ui(:,:,:,:,yhat(i),jc,kc)  )
                  squarei(:,:,:,:,ic,jc) = squarei(:,:,:,:,ic,jc)        +
     &                   ( ti(:,:,:,:,ic,kc) * ur(:,:,:,:,yhat(i),jc,kc)  -
     &                     tr(:,:,:,:,ic,kc) * ui(:,:,:,:,yhat(i),jc,kc)  )
               end do
            end do
         end do

c     calculation of the link products in the negative plaquette, NEGATIVE a x a.

c     .true.  when we calculate the full staple
```

```
c     .false. when we calculate half of the staple for plaqbar where only the
c      upper 3 plaquettes are needed

         if(local) then

         tr = 0.0d0
         ti = 0.0d0
         tsr = cshift(cshift(ur(:,:,:,:,yhat(i),:,:),
     &                dim=yhat(i),shift=-1),dim=xhat,shift=1)
         tsi = cshift(cshift(ui(:,:,:,:,yhat(i),:,:),
     &                dim=yhat(i),shift=-1),dim=xhat,shift=1)
         usqr = cshift(ur(:,:,:,:,xhat,:,:),dim=yhat(i),shift=-1)
         usqi = cshift(ui(:,:,:,:,xhat,:,:),dim=yhat(i),shift=-1)
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  tr(:,:,:,:,ic,jc) = tr(:,:,:,:,ic,jc)       +
     &              ( tsr(:,:,:,:,kc,ic) * usqr(:,:,:,:,jc,kc) -
     &                tsi(:,:,:,:,kc,ic) * usqi(:,:,:,:,jc,kc) )
                  ti(:,:,:,:,ic,jc) = ti(:,:,:,:,ic,jc)       +
     &              (-tsr(:,:,:,:,kc,ic) * usqi(:,:,:,:,jc,kc) -
     &                tsi(:,:,:,:,kc,ic) * usqr(:,:,:,:,jc,kc) )
               end do
            end do
         end do

         tr = factorsq * tr
         ti = factorsq * ti

         usqr = cshift(ur(:,:,:,:,yhat(i),:,:),dim=yhat(i),shift=-1)
         usqi = cshift(ui(:,:,:,:,yhat(i),:,:),dim=yhat(i),shift=-1)
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  squarer(:,:,:,:,ic,jc) = squarer(:,:,:,:,ic,jc) +
     &                  ( tr(:,:,:,:,ic,kc) * usqr(:,:,:,:,kc,jc) -
     &                    ti(:,:,:,:,ic,kc) * usqi(:,:,:,:,kc,jc) )
                  squarei(:,:,:,:,ic,jc) = squarei(:,:,:,:,ic,jc) +
     &                  ( tr(:,:,:,:,ic,kc) * usqi(:,:,:,:,kc,jc) +
     &                    ti(:,:,:,:,ic,kc) * usqr(:,:,:,:,kc,jc) )
               end do
            end do
         end do

         end if                    !closes the .true. if
      end do                                                 !closes the i=1,3 loop

      if(itype == 0 ) then
         squarer = ( 3.0d0 / 5.0d0 ) * squarer
         squarei = ( 3.0d0 / 5.0d0 ) * squarei
      end if
      return
      end subroutine squaresani
```

E.13 The Anisotropic Version for the Rectangle Routine

```
c
c     cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Author: Frederic D.R. Bonnet & D. B. Leinweber: August 1999.
c     computes the product of links for the action associated with a link in the
c     xhat direction. This computes the product of the six rectangular plaquettes
c     associated with the improved action.
c
      subroutine rectanglesani(ur,ui,rectr,recti,xhat,local,tmswitch,uzeros,uzerot,xi0)
      implicit none
      include '../../GeneralSupport/latticeSize.h'

c     global variable

      integer,parameter                                       :: nc=3   !sigma,color
      integer,parameter                                       :: mu=4   !direction

      integer                                                 :: xhat,tmswitch
      logical                                                 :: local
      double precision                                        :: uzeros,uzerot,xi0

      double precision,dimension(nx,ny,nz,nt,nc,nc)           :: rectr,recti
cmf$  layout rectr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout recti(:news,:news,:news,:news,:serial,:serial)
      double precision,dimension(nx,ny,nz,nt,mu,nc,nc)        :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)

c     local variables, temporary product variables

      double precision                                        :: factorrect
      integer,dimension(3)                                    :: yhat
cmf$  layout yhat(:serial)
      double precision,dimension(nx,ny,nz,nt,nc,nc)           :: t1r,t1i,t2r,t2i
cmf$  layout t1r(:news,:news,:news,:news,:serial,:serial)
cmf$  layout t1i(:news,:news,:news,:news,:serial,:serial)
cmf$  layout t2r(:news,:news,:news,:news,:serial,:serial)
cmf$  layout t2i(:news,:news,:news,:news,:serial,:serial)
      double precision,dimension(nx,ny,nz,nt,nc,nc)           :: usr,usi
cmf$  layout usr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout usi(:news,:news,:news,:news,:serial,:serial)

      integer                                                 :: muprm
      integer                                                 :: ic,jc,kc,imu,i
```

```
c     starting of the execution commands

c     setting up the yhat array
c     the yhat(1)=yhat,yhat(2)=zhat and yhat(3)=that when xhat eq 1
c     the yhat(1)=zhat,yhat(2)=that and yhat(3)=xhat when xhat eq 2
c     the yhat(1)=that,yhat(2)=xhat and yhat(3)=yhat when xhat eq 3
c     the yhat(1)=xhat,yhat(2)=yhat and yhat(3)=zhat when xhat eq 4

c      do imu=1,mu-1
c         yhat(imu) = 0
c      end do

c      do imu=1,mu-1
c         yhat(imu)    = mod( xhat + imu , 4 )
c         if( yhat(imu) == 0 ) yhat(imu) = 4
c      end do

      if( xhat == 1 ) then
         yhat(1) = 2
         yhat(2) = 3
         yhat(3) = 4
      elseif( xhat == 2 ) then
         yhat(1) = 1
         yhat(2) = 3
         yhat(3) = 4
      elseif( xhat == 3 ) then
         yhat(1) = 1
         yhat(2) = 2
         yhat(3) = 4
      elseif( xhat == 4 ) then
         yhat(1) = 1
         yhat(2) = 2
         yhat(3) = 3
      end if

c     calculation of the link products in the positive plaquette.
c     starting point in the xhat direction with the maskrect

c     first calculate the positive forward 2a x a rectangle

      rectr = 0.0d0
      recti = 0.0d0

      do i=1,mu-1                      !loop over the yhat array

c     putting the factors for the anisotropy lattices. xi0 = a_s/a_t

         if( xhat /= 4 ) then
            if( yhat(i) /= 4 ) then
               factorrect = 1.0d0 / ( xi0 * uzeros**6 )
            elseif( yhat(i) == 4 ) then
               factorrect = xi0 / ( ( uzeros**4 ) * ( uzerot**2 ) )
            end if
         elseif( xhat == 4 ) then
            if( tmswitch == 1 ) then
               factorrect = xi0 / ( ( uzeros**2 ) * ( uzerot**4 ) )
            elseif( tmswitch == 0 ) then
               factorrect = 0.0d0
            end if
         end if

         t1r = 0.0d0
         t1i = 0.0d0
         t2r = cshift(ur(:,:,:,:,xhat,:,:),dim=xhat,shift=1)
         t2i = cshift(ui(:,:,:,:,xhat,:,:),dim=xhat,shift=1)
         usr = cshift(ur(:,:,:,:,yhat(i),:,:),dim=xhat,shift=2)
         usi = cshift(ui(:,:,:,:,yhat(i),:,:),dim=xhat,shift=2)
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)       +
     &               ( t2r(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) -
     &                 t2i(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) )
                  t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)       +
     &               ( t2r(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) +
     &                 t2i(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) )
               end do
            end do
         end do

         t1r = factorrect * t1r
         t1i = factorrect * t1i

         t2r = 0.0d0
         t2i = 0.0d0
         usr = cshift(cshift(ur(:,:,:,:,xhat,:,:),dim=yhat(i),shift=1),
     &              dim=xhat,shift=1)
         usi = cshift(cshift(ui(:,:,:,:,xhat,:,:),dim=yhat(i),shift=1),
     &              dim=xhat,shift=1)
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  t2r(:,:,:,:,ic,jc) = t2r(:,:,:,:,ic,jc)       +
     &               ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                 t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                  t2i(:,:,:,:,ic,jc) = t2i(:,:,:,:,ic,jc)       +
     &               (-t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                 t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
               end do
            end do
         end do

         t1r = 0.0d0
         t1i = 0.0d0
```

```
         usr = cshift(ur(:,:,:,:,xhat,:,:),dim=yhat(i),shift=1)
         usi = cshift(ui(:,:,:,:,xhat,:,:),dim=yhat(i),shift=1)
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)     +
     &              ( t2r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                t2i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                  t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)     +
     &              (-t2r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                t2i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
               end do
            end do
         end do
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  rectr(:,:,:,:,ic,jc) = rectr(:,:,:,:,ic,jc)           +
     &                 ( t1r(:,:,:,:,ic,kc) * ur(:,:,:,:,yhat(i),jc,kc) +
     &                   t1i(:,:,:,:,ic,kc) * ui(:,:,:,:,yhat(i),jc,kc) )
                  recti(:,:,:,:,ic,jc) = recti(:,:,:,:,ic,jc)           +
     &                 (-t1r(:,:,:,:,ic,kc) * ui(:,:,:,:,yhat(i),jc,kc) +
     &                   t1i(:,:,:,:,ic,kc) * ur(:,:,:,:,yhat(i),jc,kc) )
               end do
            end do
         end do

c     now calculating the positive upper rectanglesani a x 2a

         if( xhat /= 4 ) then
            if( yhat(i) /= 4 ) then
               factorrect = 1.0d0 / ( xi0 * uzeros**6 )
            elseif( yhat(i) == 4 ) then
               if( tmswitch == 1 ) then
                  factorrect = xi0 / ( ( uzeros**2 ) * ( uzerot**4 ) )
               elseif( tmswitch == 0 ) then
                  factorrect = 0.0d0
               end if
            end if
         elseif( xhat == 4 ) then
            factorrect = xi0 / ( ( uzeros**4 ) * ( uzerot**2 ) )
         end if

         t1r = 0.0d0
         t1i = 0.0d0
         t2r = cshift(ur(:,:,:,:,yhat(i),:,:),dim=xhat,shift=1)
         t2i = cshift(ui(:,:,:,:,yhat(i),:,:),dim=xhat,shift=1)
         usr = cshift(cshift(ur(:,:,:,:,yhat(i),:,:),
     &                dim=xhat,shift=1),dim=yhat(i),shift=1)
         usi = cshift(cshift(ui(:,:,:,:,yhat(i),:,:),
     &                dim=xhat,shift=1),dim=yhat(i),shift=1)
         do ic=1,nc
```

```
            do jc=1,nc
               do kc=1,nc
                  t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)        +
     &                 ( t2r(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) -
     &                   t2i(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) )
                  t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)        +
     &                 ( t2r(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) +
     &                   t2i(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) )
               end do
            end do
         end do

         t1r = factorrect * t1r
         t1i = factorrect * t1i

         t2r = 0.0d0
         t2i = 0.0d0
         usr = cshift(ur(:,:,:,:,xhat,:,:),dim=yhat(i),shift=2)
         usi = cshift(ui(:,:,:,:,xhat,:,:),dim=yhat(i),shift=2)
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  t2r(:,:,:,:,ic,jc) = t2r(:,:,:,:,ic,jc)        +
     &                 ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                   t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                  t2i(:,:,:,:,ic,jc) = t2i(:,:,:,:,ic,jc)        +
     &                 (-t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                   t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
               end do
            end do
         end do

         t1r = 0.0d0
         t1i = 0.0d0
         usr = cshift(ur(:,:,:,:,yhat(i),:,:),dim=yhat(i),shift=1)
         usi = cshift(ui(:,:,:,:,yhat(i),:,:),dim=yhat(i),shift=1)
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)        +
     &                 ( t2r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                   t2i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                  t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)        +
     &                 (-t2r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                   t2i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
               end do
            end do
         end do

         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
```

```
                  rectr(:,:,:,:,ic,jc) = rectr(:,:,:,:,ic,jc)           +
     &                 ( t1r(:,:,:,:,ic,kc) * ur(:,:,:,:,yhat(i),jc,kc) +
     &                   t1i(:,:,:,:,ic,kc) * ui(:,:,:,:,yhat(i),jc,kc) )
                  recti(:,:,:,:,ic,jc) = recti(:,:,:,:,ic,jc)           +
     &                 (-t1r(:,:,:,:,ic,kc) * ui(:,:,:,:,yhat(i),jc,kc) +
     &                   t1i(:,:,:,:,ic,kc) * ur(:,:,:,:,yhat(i),jc,kc) )
               end do
            end do
         end do

c     .true. when we calculate the full staple
c     .false. when we calculte impbar where
c     only the upper 3 plaquette are needed

         if(local) then

c     now calculate the positive backward 2a x a retangle

            if( xhat /= 4 ) then
               if( yhat(i) /= 4 ) then
                  factorrect = 1.0d0 / ( xi0 * uzeros**6 )
               elseif( yhat(i) == 4 ) then
                  factorrect = xi0 / ( ( uzeros**4 ) * ( uzerot**2 ) )
               end if
            elseif( xhat == 4 ) then
               if( tmswitch == 1 ) then
                  factorrect = xi0 / ( ( uzeros**2 ) * ( uzerot**4 ) )
               elseif( tmswitch == 0 ) then
                  factorrect = 0.0d0
               end if
            end if

            t1r = 0.0d0
            t1i = 0.0d0
            t2r = cshift(ur(:,:,:,:,yhat(i),:,:),dim=xhat,shift=1)
            t2i = cshift(ui(:,:,:,:,yhat(i),:,:),dim=xhat,shift=1)
            usr = cshift(ur(:,:,:,:,xhat,:,:),dim=yhat(i),shift=1)
            usi = cshift(ui(:,:,:,:,xhat,:,:),dim=yhat(i),shift=1)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)      +
     &                  ( t2r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                    t2i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                     t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)      +
     &                  (-t2r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                    t2i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
                  end do
               end do
            end do

            t1r = factorrect * t1r
```

```
       t1i = factorrect * t1i

       t2r = 0.0d0
       t2i = 0.0d0
       usr = cshift(cshift(ur(:,:,:,:,xhat,:,:),
&                   dim=yhat(i),shift=1),dim=xhat,shift=-1)
       usi = cshift(cshift(ui(:,:,:,:,xhat,:,:),
&                   dim=yhat(i),shift=1),dim=xhat,shift=-1)
       do ic=1,nc
          do jc=1,nc
             do kc=1,nc
                t2r(:,:,:,:,ic,jc) = t2r(:,:,:,:,ic,jc)      +
&                  ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
&                    t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                t2i(:,:,:,:,ic,jc) = t2i(:,:,:,:,ic,jc)      +
&                  (-t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
&                    t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
             end do
          end do
       end do

       t1r = 0.0d0
       t1i = 0.0d0
       usr = cshift(ur(:,:,:,:,yhat(i),:,:),dim=xhat,shift=-1)
       usi = cshift(ui(:,:,:,:,yhat(i),:,:),dim=xhat,shift=-1)
       do ic=1,nc
          do jc=1,nc
             do kc=1,nc
                t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)      +
&                  ( t2r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
&                    t2i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)      +
&                  (-t2r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
&                    t2i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
             end do
          end do
       end do

       usr = cshift(ur(:,:,:,:,xhat,:,:),dim=xhat,shift=-1)
       usi = cshift(ui(:,:,:,:,xhat,:,:),dim=xhat,shift=-1)
       do ic=1,nc
          do jc=1,nc
             do kc=1,nc
                rectr(:,:,:,:,ic,jc) = rectr(:,:,:,:,ic,jc)    +
&                    ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) -
&                      t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) )
                recti(:,:,:,:,ic,jc) = recti(:,:,:,:,ic,jc)    +
&                    ( t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) +
&                      t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) )
             end do
          end do
       end do
```

```
c     calculation of the link products in the negative plaquette
c     starting point in the x direction

c     the negative xy,xz and xt contour when xhat eq 1
c     the negative yz,yt and yx contour when xhat eq 2
c     the negative zt,zx and zy contour when xhat eq 3
c     the negative tx,ty and tz contour when xhat eq 4


c     first calculate the negative forward 2a x a rectangle

        t1r = 0.0d0
        t1i = 0.0d0
        t2r = cshift(ur(:,:,:,:,xhat,:,:),dim=xhat,shift=1)
        t2i = cshift(ui(:,:,:,:,xhat,:,:),dim=xhat,shift=1)
        usr = cshift(cshift(ur(:,:,:,:,yhat(i),:,:),
     &                 dim=xhat,shift=2),dim=yhat(i),shift=-1)
        usi = cshift(cshift(ui(:,:,:,:,yhat(i),:,:),
     &                 dim=xhat,shift=2),dim=yhat(i),shift=-1)
        do ic=1,nc
           do jc=1,nc
              do kc=1,nc
                 t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)      +
     &              ( t2r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                t2i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                 t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)      +
     &              (-t2r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                t2i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
              end do
           end do
        end do

        t1r = factorrect * t1r
        t1i = factorrect * t1i

        t2r = 0.0d0
        t2i = 0.0d0
        usr = cshift(cshift(ur(:,:,:,:,xhat,:,:),
     &                 dim=yhat(i),shift=-1),dim=xhat,shift=1)
        usi = cshift(cshift(ui(:,:,:,:,xhat,:,:),
     &                 dim=yhat(i),shift=-1),dim=xhat,shift=1)
        do ic=1,nc
           do jc=1,nc
              do kc=1,nc
                 t2r(:,:,:,:,ic,jc) = t2r(:,:,:,:,ic,jc)      +
     &              ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                 t2i(:,:,:,:,ic,jc) = t2i(:,:,:,:,ic,jc)      +
     &              (-t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
              end do
```

```
            end do
         end do

         t1r = 0.0d0
         t1i = 0.0d0
         usr = cshift(ur(:,:,:,:,xhat,:,:),dim=yhat(i),shift=-1)
         usi = cshift(ui(:,:,:,:,xhat,:,:),dim=yhat(i),shift=-1)
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)      +
     &               ( t2r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                 t2i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                  t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)      +
     &               (-t2r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                 t2i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
               end do
            end do
         end do

         usr = cshift(ur(:,:,:,:,yhat(i),:,:),dim=yhat(i),shift=-1)
         usi = cshift(ui(:,:,:,:,yhat(i),:,:),dim=yhat(i),shift=-1)
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  rectr(:,:,:,:,ic,jc) = rectr(:,:,:,:,ic,jc)    +
     &                 ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) -
     &                   t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) )
                  recti(:,:,:,:,ic,jc) = recti(:,:,:,:,ic,jc)    +
     &                 ( t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) +
     &                   t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) )
               end do
            end do
         end do

c     calculate the negative backward 2a x a rectangle

         t1r = 0.0d0
         t1i = 0.0d0
         t2r = cshift(cshift(ur(:,:,:,:,yhat(i),:,:),
     &                dim=yhat(i),shift=-1),dim=xhat,shift=1)
         t2i = cshift(cshift(ui(:,:,:,:,yhat(i),:,:),
     &                dim=yhat(i),shift=-1),dim=xhat,shift=1)
         usr = cshift(ur(:,:,:,:,xhat,:,:),dim=yhat(i),shift=-1)
         usi = cshift(ui(:,:,:,:,xhat,:,:),dim=yhat(i),shift=-1)
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)      +
     &               ( t2r(:,:,:,:,kc,ic) * usr(:,:,:,:,jc,kc) -
     &                 t2i(:,:,:,:,kc,ic) * usi(:,:,:,:,jc,kc) )
                  t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)      +
     &                 (-t2r(:,:,:,:,kc,ic) * usi(:,:,:,:,jc,kc) -
     &                   t2i(:,:,:,:,kc,ic) * usr(:,:,:,:,jc,kc) )
               end do
            end do
         end do

         t1r = factorrect * t1r
         t1i = factorrect * t1i

         t2r = 0.0d0
         t2i = 0.0d0
         usr = cshift(cshift(ur(:,:,:,:,xhat,:,:),
     &                  dim=yhat(i),shift=-1),dim=xhat,shift=-1)
         usi = cshift(cshift(ui(:,:,:,:,xhat,:,:),
     &                  dim=yhat(i),shift=-1),dim=xhat,shift=-1)
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  t2r(:,:,:,:,ic,jc) = t2r(:,:,:,:,ic,jc)      +
     &                 ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                   t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                  t2i(:,:,:,:,ic,jc) = t2i(:,:,:,:,ic,jc)      +
     &                 (-t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                   t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
               end do
            end do
         end do

         t1r = 0.0d0
         t1i = 0.0d0
         usr = cshift(cshift(ur(:,:,:,:,yhat(i),:,:),
     &                  dim=xhat,shift=-1),dim=yhat(i),shift=-1)
         usi = cshift(cshift(ui(:,:,:,:,yhat(i),:,:),
     &                  dim=xhat,shift=-1),dim=yhat(i),shift=-1)
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)      +
     &                 ( t2r(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) -
     &                   t2i(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) )
                  t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)      +
     &                 ( t2r(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) +
     &                   t2i(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) )
               end do
            end do
         end do

         usr = cshift(ur(:,:,:,:,xhat,:,:),dim=xhat,shift=-1)
         usi = cshift(ui(:,:,:,:,xhat,:,:),dim=xhat,shift=-1)
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
```

```
                  rectr(:,:,:,:,ic,jc) = rectr(:,:,:,:,ic,jc)    +
     &                     ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) -
     &                       t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) )
                  recti(:,:,:,:,ic,jc) = recti(:,:,:,:,ic,jc)    +
     &                     ( t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) +
     &                       t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) )
               end do
            end do
         end do

c     now calculating the negative lower rectanglesani a x 2a

         if( xhat /= 4 ) then
            if( yhat(i) /= 4 ) then
               factorrect = 1.0d0 / ( xi0 * uzeros**6 )
            elseif( yhat(i) == 4 ) then
               if( tmswitch == 1 ) then
                  factorrect = xi0 / ( ( uzeros**2 ) * ( uzerot**4 ) )
               elseif( tmswitch == 0 ) then
                  factorrect = 0.0d0
               end if
            end if
         elseif( xhat == 4 ) then
            factorrect = xi0 / ( ( uzeros**4 ) * ( uzerot**2 ) )
         end if

         t1r = 0.0d0
         t1i = 0.0d0
         t2r = cshift(cshift(ur(:,:,:,:,yhat(i),:,:),
     &                dim=xhat,shift=1),dim=yhat(i),shift=-1)
         t2i = cshift(cshift(ui(:,:,:,:,yhat(i),:,:),
     &                dim=xhat,shift=1),dim=yhat(i),shift=-1)
         usr = cshift(cshift(ur(:,:,:,:,yhat(i),:,:),
     &                dim=xhat,shift=1),dim=yhat(i),shift=-2)
         usi = cshift(cshift(ui(:,:,:,:,yhat(i),:,:),
     &                dim=xhat,shift=1),dim=yhat(i),shift=-2)
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)      +
     &               ( t2r(:,:,:,:,kc,ic) * usr(:,:,:,:,jc,kc) -
     &                 t2i(:,:,:,:,kc,ic) * usi(:,:,:,:,jc,kc) )
                  t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)      +
     &               (-t2r(:,:,:,:,kc,ic) * usi(:,:,:,:,jc,kc) -
     &                 t2i(:,:,:,:,kc,ic) * usr(:,:,:,:,jc,kc) )
               end do
            end do
         end do

         t1r = factorrect * t1r
         t1i = factorrect * t1i
         t2r = 0.0d0
         t2i = 0.0d0
         usr = cshift(ur(:,:,:,:,xhat,:,:),dim=yhat(i),shift=-2)
         usi = cshift(ui(:,:,:,:,xhat,:,:),dim=yhat(i),shift=-2)
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  t2r(:,:,:,:,ic,jc) = t2r(:,:,:,:,ic,jc)      +
     &               ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                 t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                  t2i(:,:,:,:,ic,jc) = t2i(:,:,:,:,ic,jc)      +
     &               (-t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                 t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
               end do
            end do
         end do

         t1r = 0.0d0
         t1i = 0.0d0
         usr = cshift(ur(:,:,:,:,yhat(i),:,:),dim=yhat(i),shift=-2)
         usi = cshift(ui(:,:,:,:,yhat(i),:,:),dim=yhat(i),shift=-2)
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)      +
     &               ( t2r(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) -
     &                 t2i(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) )
                  t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)      +
     &               ( t2r(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) +
     &                 t2i(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) )
               end do
            end do
         end do

         usr = cshift(ur(:,:,:,:,yhat(i),:,:),dim=yhat(i),shift=-1)
         usi = cshift(ui(:,:,:,:,yhat(i),:,:),dim=yhat(i),shift=-1)
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  rectr(:,:,:,:,ic,jc) = rectr(:,:,:,:,ic,jc)    +
     &                 ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) -
     &                   t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) )
                  recti(:,:,:,:,ic,jc) = recti(:,:,:,:,ic,jc)    +
     &                 ( t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) +
     &                   t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) )
               end do
            end do
         end do

      end if
   end do
```

```
      rectr = - ( 1.0d0 / 12.0d0 ) * rectr
      recti = - ( 1.0d0 / 12.0d0 ) * recti

      return
      end subroutine rectanglesani
```

E.14 Calculating the Gauge Action

```
c
c     ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Author: Frederic D.R. Bonnet & D.B. Leinweber: date: March 1999.
c     GetAction subroutine will calculate the total action, which is a sum
c     over all elementary action in the lattice.
c
      subroutine GetAction(ur,ui,action,plaqbar,itype,uzero)
      implicit none
      include 'latticeSize.h'

c     global variables

      integer,parameter                                       :: nc=3
      integer,parameter                                       :: mu=4

      integer                                                 :: itype
      double precision                                        :: uzero
      double precision                                        :: plaqbar

      double precision,dimension(nx,ny,nz,nt,mu,nc,nc)        :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)
      double precision,dimension(nx,ny,nz,nt)                 :: action
cmf$  layout action(:news,:news,:news,:news)

c     local variables

      double precision                                        :: factor

      double precision,dimension(nx,ny,nz,nt,nc,nc)           :: stapler,staplei
cmf$  layout stapler(:news,:news,:news,:news,:serial,:serial)
cmf$  layout staplei(:news,:news,:news,:news,:serial,:serial)

      integer                                                 :: ihat,ic,kc

      INTERFACE
         SUBROUTINE staples(ur,ui,stapler,staplei,xhat,local,itype,uzero)
          IMPLICIT NONE
          include 'latticeSize.h'
          integer,parameter                                   :: nc=3
          integer,parameter                                   :: mu=4
          integer                                             :: xhat
          integer                                             :: itype
          double precision                                   :: uzero
          logical                                            :: local
          double precision,dimension(nx,ny,nz,nt,nc,nc)      :: stapler,staplei
cmf$  layout stapler(:news,:news,:news,:news,:serial,:serial)
cmf$  layout staplei(:news,:news,:news,:news,:serial,:serial)
          double precision,dimension(nx,ny,nz,nt,mu,nc,nc)   :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)
         end SUBROUTINE staples
      end interface

c     In the following do loop we are calculating the average plaquette.
c     when calling staples only three plaquette are calculated.
c     The ones in the positive direction ( not in the negative direction
c     because of the local=.false. logical condition).
c     at an arbitrary point we have
c     at ihat=1, plaquette 12,13,14. At ihat=2, plaquette 21,23,24
c     at ihat=3, plaquette 31,32,34. At ihat=4, plaquette 41,42,43

c     Initialize action to zero here.

      if( itype == 0 ) factor = 1.0d0
      if( itype == 1 ) factor = 5.0d0 / 3.0d0 - 1.0d0 / ( 6.0d0 * uzero**2 )

      action  = 0.0d0
      plaqbar = 0.0d0

      do ihat=1,mu
         call staples(ur,ui,stapler,staplei,ihat,.false.,itype,uzero)
         do ic=1,nc
            do kc=1,nc
               action(:,:,:,:)    = action(:,:,:,:)    +
     &              ( ur(:,:,:,:,ihat,ic,kc) * stapler(:,:,:,:,kc,ic) -
     &                ui(:,:,:,:,ihat,ic,kc) * staplei(:,:,:,:,kc,ic) )
            end do
         end do
      end do

c     each subroutine call to staples calculates 3 plaquettes (mu-1)
c     when called with .false., and there are 4 calls, one for each direction (mu).

      action = factor - action / ( nc * mu*(mu-1) )

      plaqbar = sum(action) / ( nx*ny*nz*nt )

      return
      end subroutine GetAction
```

E.15 Calculating the Tadpole Factor, u_0

```
c
```

```
c     cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Author: Frederic D.R. Bonnet: August 1998.
c     tadpoleimp subroutine will calculate the u_0 which is just the fourth
c     root of the average square plaquette. This uzero is uzero for the tadpole
c     improvement.
c
      subroutine tadpoleimp(ur,ui,uzero)
      implicit none
      include 'latticeSize.h'

c     global variables

      integer,parameter                                     :: nc=3   !sigma,color
      integer,parameter                                     :: mu=4   !direction

      double precision                                      :: uzero

      double precision,dimension(nx,ny,nz,nt,mu,nc,nc)      :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)

c     local variables

      double precision,dimension(nx,ny,nz,nt)               :: action
cmf$  layout action(:news,:news,:news,:news)
      double precision,dimension(nx,ny,nz,nt,nc,nc)         :: squarer,squarei
cmf$  layout squarer(:news,:news,:news,:news,:serial,:serial)
cmf$  layout squarei(:news,:news,:news,:news,:serial,:serial)

      integer                                               :: ihat,ic,kc

      interface
       SUBROUTINE squares(ur,ui,squarer,squarei,xhat,local)
        IMPLICIT NONE
        include 'latticeSize.h'
        integer,parameter                                   :: nc=3   !sigma,color
        integer,parameter                                   :: mu=4   !direction
        integer                                             :: xhat
        logical                                             :: local
        double precision,dimension(nx,ny,nz,nt,nc,nc)       :: squarer,squarei
cmf$  layout squarer(:news,:news,:news,:news,:serial,:serial)
cmf$  layout squarei(:news,:news,:news,:news,:serial,:serial)
        double precision,dimension(nx,ny,nz,nt,mu,nc,nc)    :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)
       end SUBROUTINE squares
      end interface

c     In the following do loop we are calculating the average plaquette.
c     when calling staples only three plaquette are calculated.
c     The ones in the positive direction ( not in the negative direction
c     because of the local=.false. logical condition).
```

```
c     at an arbitrary point we have
c     at ihat=1, plaquette 12,13,14. At ihat=2, plaquette 21,23,24
c     at ihat=3, plaquette 31,32,34. At ihat=4, plaquette 41,42,43

      action = 0.0d0
      uzero = 0.0d0

      do ihat=1,mu
         call squares(ur,ui,squarer,squarei,ihat,.false.)
         do ic=1,nc
            do kc=1,nc
               action(:,:,:,:) = action(:,:,:,:) +
     &              ( ur(:,:,:,:,ihat,ic,kc) * squarer(:,:,:,:,kc,ic) -
     &                ui(:,:,:,:,ihat,ic,kc) * squarei(:,:,:,:,kc,ic) )
            end do
         end do
      end do

c     now calculating the the fourth root of the plaquette. This uzero
c     makes up the tadpole improvement 1/3ReTr(square) consists of only
c     tadpole contributions. A large contribution of the tadpole contribution
c     can be eliminated by dividing every link operator by u_0

      uzero = ( sum(action) / ( nx*ny*nz*nt*nc*mu*(mu-1) ) ) ** ( 0.25d0 )

      return
      end subroutine tadpoleimp
```

E.16 Constructing the Topological Charge Operator on the Lattice

```
c
c     ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Author: Frederic D.R. Bonnet; date: 1st of March 2001.
c     subtoutine to calculate the topological charge Q.
c
      subroutine Qtop(Fr,Fi,QTopDensity,Q)
      implicit none
      include 'latticeSize.h'

c     global variables

      integer,parameter                                      :: nc=3    !no of colours
      integer,parameter                                      :: mu=4    !directions
      integer,parameter                                      :: nf=6    !# of Fmunu

      double precision                                       :: Q !topological charge

      double precision,dimension(nx,ny,nz,nt)                :: QTopDensity
cmf$  layout QTopDensity(:news,:news,:news,:news)
```

```
      double precision,dimension(nx,ny,nz,nt,nc,nc,nf)        :: Fr,Fi   !Fmunu
cmf$  layout Fr(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout Fi(:news,:news,:news,:news,:serial,:serial,:serial)

c     local variables

      double precision                                        :: pi=0.0d0

      integer,dimension(2:mu,1:mu-1)                          :: Findex
cmf$  layout Findex(:serial,:serial)

      integer                                                 :: i,j,ic,jc,kc

c     start of the execution commands.

      pi = 4.0d0 * atan(1.0d0)

c
c     Set up the Findex
c
      do i=2,mu
         do j=1,i-1
            Findex(i,j) = (i+j) - 3 * ( (i-j) / 3 ) - 1
         end do
      end do
c
c     Calculation of topological charge density
c
c     QTopDensity(x,y,z,t) is proportional to
c     colourtrace{levicevita(mu,nu,rho,sigma) *
c                       F(x,y,z,t,a,b,mu,nu)F(x,y,z,t,a,b,rho,sigma)]}

      QTopDensity = 0.0d0
      do ic=1,nc
         do kc=1,nc
            QTopDensity(:,:,:,:) = QTopDensity(:,:,:,:) +
     &           ( Fr(:,:,:,:,ic,kc,Findex(2,1)) * Fr(:,:,:,:,kc,ic,Findex(4,3)) -
     &         Fr(:,:,:,:,ic,kc,Findex(3,1)) * Fr(:,:,:,:,kc,ic,Findex(4,2)) +
     &         Fr(:,:,:,:,ic,kc,Findex(4,1)) * Fr(:,:,:,:,kc,ic,Findex(3,2)) -
     &         Fi(:,:,:,:,ic,kc,Findex(2,1)) * Fi(:,:,:,:,kc,ic,Findex(4,3)) +
     &         Fi(:,:,:,:,ic,kc,Findex(3,1)) * Fi(:,:,:,:,kc,ic,Findex(4,2)) -
     &         Fi(:,:,:,:,ic,kc,Findex(4,1)) * Fi(:,:,:,:,kc,ic,Findex(3,2)) ) * 8.0d0
         end do
      end do
c
c     Calculation and display of topological charge
c
      QTopDensity = QTopDensity / ( 32.0d0 * ( pi**2 ) )
      Q = sum(QTopDensity)

      return
      end subroutine Qtop
```

E.17 Constructing The non–Abelian field strength Tensor

```
c
c     cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     author: Frederic D.R. Bonnet, January 2000.
c     subroutine will calculate the topological charge using the field,
c     strength tensor Fmunu. It is a sum over all elementary 4 plaquettes.
c     It combines the standard term (the square contribution) and
c     the rectangular part of the topological charge tensor.
c
      subroutine Fmunu(ur,ui,Fr,Fi,uzero,itopo)
      implicit none
      include 'latticeSize.h'

c     global variables

      integer,parameter                                      :: nc=3
      integer,parameter                                      :: mu=4

      integer,parameter                                      :: nf=6

      integer                                                :: itopo !which Q.
      double precision                                       :: uzero

      double precision,dimension(nx,ny,nz,nt,nc,nc,nf)       :: Fr,Fi     !Fmunu
cmf$  layout Fr(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout Fi(:news,:news,:news,:news,:serial,:serial,:serial)
      double precision,dimension(nx,ny,nz,nt,mu,nc,nc)       :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)

c     local variables

      double precision,dimension(nx,ny,nz,nt)                :: TrFr  !Trace Variable
cmf$  layout TrFr(:news,:news,:news,:news)
      double precision,dimension(nx,ny,nz,nt,nc,nc)          :: Clr,Cli
cmf$  layout Clr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout Cli(:news,:news,:news,:news,:serial,:serial)
      double precision,dimension(nx,ny,nz,nt,nc,nc)          :: rectClr,rectCli
cmf$  layout rectClr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout rectCli(:news,:news,:news,:news,:serial,:serial)

      integer                                                :: xhat,yhat
      integer                                                :: index
      integer                                                :: ic,jc,kc

      interface
         subroutine clover(ur,ui,sqCr,sqCi,muhat,nuhat)
          implicit none
          include 'latticeSize.h'
          integer,parameter                              :: nc=3
          integer,parameter                              :: nd=4
          integer,parameter                              :: nf=6   !# of F stored
          integer                                      :: muhat,nuhat
          double precision,dimension(nx,ny,nz,nt,nc,nc)       :: sqCr,sqCi!Clover Term
cmf$  layout sqCr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout sqCi(:news,:news,:news,:news,:serial,:serial)
          double precision,dimension(nx,ny,nz,nt,nd,nc,nc)    :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)
         end subroutine clover
         subroutine impclover(ur,ui,rectCr,rectCi,xhat,yhat)
          implicit none
          include 'latticeSize.h'
          integer,parameter                              :: nc=3
          integer,parameter                              :: nd=4
          integer,parameter                              :: nf=6
          integer                                            :: xhat,yhat
          double precision,dimension(nx,ny,nz,nt,nc,nc)       :: rectCr,rectCi
cmf$  layout rectCr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout rectCi(:news,:news,:news,:news,:serial,:serial)
          double precision,dimension(nx,ny,nz,nt,nd,nc,nc)    :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)
         end subroutine impclover
      end interface

c     start of the execution commands.

c
c     Now lets implement the Hermitian and traceless aspects of
c     the Gell-Mann Matrices by subtracting the
c     Hermitian conjugate of C which gives us a clover term of  C = 2i*g*F_mu_nu.
c     Hence multiply C by -i to get g*F_mu_nu.
c

      Fr = 0.0d0
      Fi = 0.0d0

      do xhat=2,mu
         do yhat=1,xhat-1

            index = ( xhat + yhat ) - 3 * ( ( xhat-yhat ) / 3 ) - 1

            call clover(ur,ui,Clr,Cli,xhat,yhat)

            if(itopo==1) then
               call impclover(ur,ui,rectClr,rectCli,xhat,yhat)
               Clr = ( 5.0d0 / 3.0d0 ) * Clr -
     &               ( 1.0d0 / ( 6.0d0 * (uzero**2) ) ) * rectClr
               Cli = ( 5.0d0 / 3.0d0 ) * Cli -
     &               ( 1.0d0 / ( 6.0d0 * (uzero**2) ) ) * rectCli
```

```
            end if

            do ic=1,nc
               do jc=1,nc
                  Fr(:,:,:,:,ic,jc,index) =
     &               ( Cli(:,:,:,:,ic,jc) + Cli(:,:,:,:,jc,ic) ) / 2.0d0
                  Fi(:,:,:,:,ic,jc,index) =
     &            - ( Clr(:,:,:,:,ic,jc) - Clr(:,:,:,:,jc,ic) ) / 2.0d0
               end do
            end do
c
c     Fi is traceless, Fr is not, but the Gell-Mann Matrices are.
c     Therefore subtract 1/3 the trace from each diagonal element of Fr.
c
            TrFr = 0.0d0
            do ic=1,nc
               TrFr = TrFr + Fr(:,:,:,:,ic,ic,index)
            end do
            do ic=1,nc
               Fr(:,:,:,:,ic,ic,index) = Fr(:,:,:,:,ic,ic,index) - ( TrFr ) / 3.0d0
            end do

         end do
      end do

      return
      end subroutine Fmunu
c
c     cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     author: Frederic D.R. Bonnet, Patrick Fitzhenry, D.B. Leinweber: February 1999.
c     clover subroutine will calculate the topological charge using the field,
c     strength tensor Fmunu. It is a sum over all elementary 4 plaquettes.
c
      subroutine clover(ur,ui,sqCr,sqCi,muhat,nuhat)
      implicit none
      include 'latticeSize.h'

c     global variables

      integer,parameter                                       :: nc=3
      integer,parameter                                       :: mu=4

      integer,parameter                                       :: nf=6

      double precision,dimension(nx,ny,nz,nt,nc,nc)          :: sqCr,sqCi
cmf$  layout sqCr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout sqCi(:news,:news,:news,:news,:serial,:serial)
      double precision,dimension(nx,ny,nz,nt,mu,nc,nc)       :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)

c     local variables
```

```
      double precision,dimension(nx,ny,nz,nt,nc,nc)           :: tr1,ti1,tr2,ti2
cmf$  layout tr1(:news,:news,:news,:news,:serial,:serial)
cmf$  layout ti1(:news,:news,:news,:news,:serial,:serial)
cmf$  layout tr2(:news,:news,:news,:news,:serial,:serial)
cmf$  layout ti2(:news,:news,:news,:news,:serial,:serial)

      double precision,dimension(nx,ny,nz,nt,nc,nc)           :: fsr,fsi
cmf$  layout fsr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout fsi(:news,:news,:news,:news,:serial,:serial)

      integer                                                :: ic,jc,kc,muhat,nuhat
      integer                                                :: index

c     start of the execution commands.
c
c     Calculation of F_mu_nu for a given muhat and xhat.
c


c
c     ReZero the clover term
c
            sqCr = 0.0d0
            sqCi = 0.0d0

c     Calculate the index that represents the unique combination of
c     muhat and nuhat.
c     This is done to save storage.
c     F(x,y,z,t,a,b,mu,nu) contains 16 elements but we can exploit the
c     traceless antisymmetric properties of F and only store 6
c     elements for each F.
c     ie
c     F(x,y,z,t,a,b,2,1)->F(x,a,b,2)
c     F(x,y,z,t,a,b,3,1)->F(x,a,b,3)
c     F(x,y,z,t,a,b,3,2)->F(x,a,b,4)
c     F(x,y,z,t,a,b,4,1)->F(x,a,b,1)
c     F(x,y,z,t,a,b,4,2)->F(x,a,b,5)
c     F(x,y,z,t,a,b,4,3)->F(x,a,b,6)
c     The unique muhat/nuhat index combination is rolled into the
c     single index by the following relation

            index = ( muhat + nuhat ) - 3 * ( ( muhat-nuhat ) / 3 ) - 1

c     Calculation of link products for upper-right plaquette, #1

c     t1 = U(x,mu)*U(x+mu,nu)

            tr1 = 0.0d0
            ti1 = 0.0d0
            tr2 = cshift(ur(:,:,:,:,nuhat,:,:),dim=muhat,shift=1)
            ti2 = cshift(ui(:,:,:,:,nuhat,:,:),dim=muhat,shift=1)
            do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  tr1(:,:,:,:,ic,jc) = tr1(:,:,:,:,ic,jc)           +
     &               ( ur(:,:,:,:,muhat,ic,kc) * tr2(:,:,:,:,kc,jc) -
     &                 ui(:,:,:,:,muhat,ic,kc) * ti2(:,:,:,:,kc,jc) )
                  ti1(:,:,:,:,ic,jc) = ti1(:,:,:,:,ic,jc)           +
     &               ( ui(:,:,:,:,muhat,ic,kc) * tr2(:,:,:,:,kc,jc) +
     &                 ur(:,:,:,:,muhat,ic,kc) * ti2(:,:,:,:,kc,jc) )
               end do
            end do
         end do
c
c     t2 = t1*Udagger(x+nu,mu)
c
         tr2 = 0.0d0
         ti2 = 0.0d0
         fsr = cshift(ur(:,:,:,:,muhat,:,:),dim=nuhat,shift=1)
         fsi = cshift(ui(:,:,:,:,muhat,:,:),dim=nuhat,shift=1)
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  tr2(:,:,:,:,ic,jc) = tr2(:,:,:,:,ic,jc)      +
     &               ( tr1(:,:,:,:,ic,kc) * fsr(:,:,:,:,jc,kc) +
     &                 ti1(:,:,:,:,ic,kc) * fsi(:,:,:,:,jc,kc) )
                  ti2(:,:,:,:,ic,jc) = ti2(:,:,:,:,ic,jc)      +
     &               ( ti1(:,:,:,:,ic,kc) * fsr(:,:,:,:,jc,kc) -
     &                 tr1(:,:,:,:,ic,kc) * fsi(:,:,:,:,jc,kc) )
               end do
            end do
         end do
c
c     Fnew = Fold + Im{t2*Udagger(x,nu)}/4
c
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  sqCr(:,:,:,:,ic,jc) = sqCr(:,:,:,:,ic,jc)     +
     &                ( tr2(:,:,:,:,ic,kc) * ur(:,:,:,:,nuhat,jc,kc) +
     &                  ti2(:,:,:,:,ic,kc) * ui(:,:,:,:,nuhat,jc,kc) ) / 4.0d0
                  sqCi(:,:,:,:,ic,jc) = sqCi(:,:,:,:,ic,jc)     +
     &                ( ti2(:,:,:,:,ic,kc) * ur(:,:,:,:,nuhat,jc,kc) -
     &                  tr2(:,:,:,:,ic,kc) * ui(:,:,:,:,nuhat,jc,kc) ) / 4.0d0
               end do
            end do
         end do
c
c     Calculation of link products for upper-left plaquette, #2
c
c     t1 = U(x,nu)*Udagger(x-mu+nu,mu)
c
         tr1 = 0.0d0
         ti1 = 0.0d0
```

```
           tr2 = cshift(cshift(ur(:,:,:,:,muhat,:,:),
     &                   dim=muhat,shift=-1),dim=nuhat,shift=1)
           ti2 = cshift(cshift(ui(:,:,:,:,muhat,:,:),
     &                   dim=muhat,shift=-1),dim=nuhat,shift=1)
           do ic=1,nc
              do jc=1,nc
                 do kc=1,nc
                    tr1(:,:,:,:,ic,jc) = tr1(:,:,:,:,ic,jc)            +
     &                   ( ur(:,:,:,:,nuhat,ic,kc) * tr2(:,:,:,:,jc,kc) +
     &                     ui(:,:,:,:,nuhat,ic,kc) * ti2(:,:,:,:,jc,kc) )
                    ti1(:,:,:,:,ic,jc) = ti1(:,:,:,:,ic,jc)            +
     &                   ( ui(:,:,:,:,nuhat,ic,kc) * tr2(:,:,:,:,jc,kc) -
     &                     ur(:,:,:,:,nuhat,ic,kc) * ti2(:,:,:,:,jc,kc) )
                 end do
              end do
           end do
c
c     t2 = t1*Udagger(x-mu,nu)
c
           tr2 = 0.0d0
           ti2 = 0.0d0
           fsr = cshift(ur(:,:,:,:,nuhat,:,:),dim=muhat,shift=-1)
           fsi = cshift(ui(:,:,:,:,nuhat,:,:),dim=muhat,shift=-1)
           do ic=1,nc
              do jc=1,nc
                 do kc=1,nc
                    tr2(:,:,:,:,ic,jc) = tr2(:,:,:,:,ic,jc)      +
     &                 ( tr1(:,:,:,:,ic,kc) * fsr(:,:,:,:,jc,kc) +
     &                   ti1(:,:,:,:,ic,kc) * fsi(:,:,:,:,jc,kc) )
                    ti2(:,:,:,:,ic,jc) = ti2(:,:,:,:,ic,jc)      +
     &                 ( ti1(:,:,:,:,ic,kc) * fsr(:,:,:,:,jc,kc) -
     &                   tr1(:,:,:,:,ic,kc) * fsi(:,:,:,:,jc,kc) )
                 end do
              end do
           end do
c
c     Fnew = Fold + Im{t2*U(x-mu,mu)}/4
c
           fsr = cshift(ur(:,:,:,:,muhat,:,:),dim=muhat,shift=-1)
           fsi = cshift(ui(:,:,:,:,muhat,:,:),dim=muhat,shift=-1)
           do ic=1,nc
              do jc=1,nc
                 do kc=1,nc
                    sqCr(:,:,:,:,ic,jc) = sqCr(:,:,:,:,ic,jc) +
     &                   ( tr2(:,:,:,:,ic,kc) * fsr(:,:,:,:,kc,jc)  -
     &                     ti2(:,:,:,:,ic,kc) * fsi(:,:,:,:,kc,jc)  ) / 4.0d0
                    sqCi(:,:,:,:,ic,jc) = sqCi(:,:,:,:,ic,jc) +
     &                   ( ti2(:,:,:,:,ic,kc) * fsr(:,:,:,:,kc,jc)  +
     &                     tr2(:,:,:,:,ic,kc) * fsi(:,:,:,:,kc,jc)  ) / 4.0d0
                 end do
              end do
           end do
```

```
c
c     Calculation of link products for lower-left plaquette, #3
c
c
c     t1 = Udagger(x-mu,mu)*Udagger(x-mu-nu,nu)
c
            tr1 = 0.0d0
            ti1 = 0.0d0
            tr2 = cshift(ur(:,:,:,:,muhat,:,:),dim=muhat,shift=-1)
            ti2 = cshift(ui(:,:,:,:,muhat,:,:),dim=muhat,shift=-1)
            fsr = cshift(cshift(ur(:,:,:,:,nuhat,:,:),
     &                  dim=muhat,shift=-1),dim=nuhat,shift=-1)
            fsi = cshift(cshift(ui(:,:,:,:,nuhat,:,:),
     &                  dim=muhat,shift=-1),dim=nuhat,shift=-1)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     tr1(:,:,:,:,ic,jc) = tr1(:,:,:,:,ic,jc)      +
     &                  ( tr2(:,:,:,:,kc,ic) * fsr(:,:,:,:,jc,kc) -
     &                    ti2(:,:,:,:,kc,ic) * fsi(:,:,:,:,jc,kc) )
                     ti1(:,:,:,:,ic,jc) = ti1(:,:,:,:,ic,jc)      +
     &                  (-ti2(:,:,:,:,kc,ic) * fsr(:,:,:,:,jc,kc) -
     &                    tr2(:,:,:,:,kc,ic) * fsi(:,:,:,:,jc,kc) )
                  end do
               end do
            end do
c
c     t2 = t1*U(x-mu-nu,mu)
c
            tr2 = 0.0d0
            ti2 = 0.0d0
            fsr = cshift(cshift(ur(:,:,:,:,muhat,:,:),
     &                  dim=muhat,shift=-1),dim=nuhat,shift=-1)
            fsi = cshift(cshift(ui(:,:,:,:,muhat,:,:),
     &                  dim=muhat,shift=-1),dim=nuhat,shift=-1)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     tr2(:,:,:,:,ic,jc) = tr2(:,:,:,:,ic,jc)      +
     &                  ( tr1(:,:,:,:,ic,kc) * fsr(:,:,:,:,kc,jc) -
     &                    ti1(:,:,:,:,ic,kc) * fsi(:,:,:,:,kc,jc) )
                     ti2(:,:,:,:,ic,jc) = ti2(:,:,:,:,ic,jc)      +
     &                  ( ti1(:,:,:,:,ic,kc) * fsr(:,:,:,:,kc,jc) +
     &                    tr1(:,:,:,:,ic,kc) * fsi(:,:,:,:,kc,jc) )
                  end do
               end do
            end do
c
c     Fnew = Fold + Im{t2*U(x-nu,nu)}/4
c
            fsr = cshift(ur(:,:,:,:,nuhat,:,:),dim=nuhat,shift=-1)
            fsi = cshift(ui(:,:,:,:,nuhat,:,:),dim=nuhat,shift=-1)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     sqCr(:,:,:,:,ic,jc) = sqCr(:,:,:,:,ic,jc) +
     &                    ( tr2(:,:,:,:,ic,kc) * fsr(:,:,:,:,kc,jc)  -
     &                      ti2(:,:,:,:,ic,kc) * fsi(:,:,:,:,kc,jc)  ) / 4.0d0
                     sqCi(:,:,:,:,ic,jc) = sqCi(:,:,:,:,ic,jc) +
     &                    ( ti2(:,:,:,:,ic,kc) * fsr(:,:,:,:,kc,jc)  +
     &                      tr2(:,:,:,:,ic,kc) * fsi(:,:,:,:,kc,jc)  ) / 4.0d0
                  end do
               end do
            end do
c
c     Calculation of link products for lower-right plaquette, #4
c
c     t1 = Udagger(x-nu,nu)*U(x-nu,mu)
c
            tr1 = 0.0d0
            ti1 = 0.0d0
            tr2 = cshift(ur(:,:,:,:,nuhat,:,:),dim=nuhat,shift=-1)
            ti2 = cshift(ui(:,:,:,:,nuhat,:,:),dim=nuhat,shift=-1)
            fsr = cshift(ur(:,:,:,:,muhat,:,:),dim=nuhat,shift=-1)
            fsi = cshift(ui(:,:,:,:,muhat,:,:),dim=nuhat,shift=-1)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     tr1(:,:,:,:,ic,jc) = tr1(:,:,:,:,ic,jc)      +
     &                  ( tr2(:,:,:,:,kc,ic) * fsr(:,:,:,:,kc,jc) +
     &                    ti2(:,:,:,:,kc,ic) * fsi(:,:,:,:,kc,jc) )
                     ti1(:,:,:,:,ic,jc) = ti1(:,:,:,:,ic,jc)      +
     &                  (-ti2(:,:,:,:,kc,ic) * fsr(:,:,:,:,kc,jc) +
     &                    tr2(:,:,:,:,kc,ic) * fsi(:,:,:,:,kc,jc) )
                  end do
               end do
            end do
c
c     t2 = t1*U(x+mu-nu,nu)
c
            tr2 = 0.0d0
            ti2 = 0.0d0
            fsr = cshift(cshift(ur(:,:,:,:,nuhat,:,:),
     &                   dim=muhat,shift=1),dim=nuhat,shift=-1)
            fsi = cshift(cshift(ui(:,:,:,:,nuhat,:,:),
     &                   dim=muhat,shift=1),dim=nuhat,shift=-1)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     tr2(:,:,:,:,ic,jc) = tr2(:,:,:,:,ic,jc)      +
     &                  ( tr1(:,:,:,:,ic,kc) * fsr(:,:,:,:,kc,jc) -
     &                    ti1(:,:,:,:,ic,kc) * fsi(:,:,:,:,kc,jc) )
                     ti2(:,:,:,:,ic,jc) = ti2(:,:,:,:,ic,jc)      +
     &                  ( ti1(:,:,:,:,ic,kc) * fsr(:,:,:,:,kc,jc) +
     &                    tr1(:,:,:,:,ic,kc) * fsi(:,:,:,:,kc,jc) )
                    end do
                 end do
              end do
c
c     Fnew = Fold + Im{t2*Udagger(x,mu)}/4
c
              do ic=1,nc
                 do jc=1,nc
                    do kc=1,nc
                       sqCr(:,:,:,:,ic,jc) = sqCr(:,:,:,:,ic,jc)     +
     &                    ( tr2(:,:,:,:,ic,kc) * ur(:,:,:,:,muhat,jc,kc) +
     &                      ti2(:,:,:,:,ic,kc) * ui(:,:,:,:,muhat,jc,kc) ) / 4.0d0
                       sqCi(:,:,:,:,ic,jc) = sqCi(:,:,:,:,ic,jc)     +
     &                    ( ti2(:,:,:,:,ic,kc) * ur(:,:,:,:,muhat,jc,kc) -
     &                      tr2(:,:,:,:,ic,kc) * ui(:,:,:,:,muhat,jc,kc) ) / 4.0d0
                    end do
                 end do
              end do

      return
      end subroutine clover
c
c     cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     author: Frederic D.R. Bonnet, January 2000.
c     Subroutine to calculate an improved topological charge operator.
c     This operator will contain the 1x1 standard Wilson loop plus
c     1x2 and 2x1 Wilson loops.
c
      subroutine impclover(ur,ui,rectCr,rectCi,xhat,yhat)
      implicit none
      include 'latticeSize.h'

c     Global variables.

      integer,parameter                                    :: nc=3  !no. colours
      integer,parameter                                    :: mu=4  !no. dimensions

      integer,parameter                                    :: nf=6

      double precision,dimension(nx,ny,nz,nt,mu,nc,nc)     :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)

      double precision,dimension(nx,ny,nz,nt,nc,nc,nf)     :: rectFr,rectFi
cmf$  layout rectFr(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout rectFi(:news,:news,:news,:news,:serial,:serial,:serial)

c     Local variables.

      double precision,dimension(nx,ny,nz,nt)              :: impTrFr
cmf$  layout impTrFr(:news,:news,:news,:news)
```

```
      double precision,dimension(nx,ny,nz,nt,nc,nc)           :: t1r,t1i,t2r,t2i
cmf$  layout t1r(:news,:news,:news,:news,:serial,:serial)
cmf$  layout t1i(:news,:news,:news,:news,:serial,:serial)
cmf$  layout t2r(:news,:news,:news,:news,:serial,:serial)
cmf$  layout t2i(:news,:news,:news,:news,:serial,:serial)
      double precision,dimension(nx,ny,nz,nt,nc,nc)           :: usr,usi
cmf$  layout usr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout usi(:news,:news,:news,:news,:serial,:serial)
      double precision,dimension(nx,ny,nz,nt,nc,nc)           :: rectCr,rectCi
cmf$  layout rectCr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout rectCi(:news,:news,:news,:news,:serial,:serial)

      integer                                                 :: xhat,yhat
      integer                                               :: index
      integer                                                 :: ic,jc,kc,imu,i

c     start of the execution commands

c     calculation of the first set 2x1 upper forward + 1x2 upper forward.

            rectCr = 0.0d0
            rectCi = 0.0d0

c     2x1 upper forward.

            t1r = 0.0d0
            t1i = 0.0d0
            usr = cshift(ur(:,:,:,:,xhat,:,:),dim=xhat,shift=1)
            usi = cshift(ui(:,:,:,:,xhat,:,:),dim=xhat,shift=1)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)          +
     &                    ( ur(:,:,:,:,xhat,ic,kc) * usr(:,:,:,:,kc,jc) -
     &                      ui(:,:,:,:,xhat,ic,kc) * usi(:,:,:,:,kc,jc) )
                     t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)          +
     &                    ( ur(:,:,:,:,xhat,ic,kc) * usi(:,:,:,:,kc,jc) +
     &                      ui(:,:,:,:,xhat,ic,kc) * usr(:,:,:,:,kc,jc) )
                  end do
               end do
            end do

            t2r = 0.0d0
            t2i = 0.0d0
            usr = cshift(ur(:,:,:,:,yhat,:,:),dim=xhat,shift=2)
            usi = cshift(ui(:,:,:,:,yhat,:,:),dim=xhat,shift=2)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     t2r(:,:,:,:,ic,jc) = t2r(:,:,:,:,ic,jc)      +
     &                    ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) -
     &                      t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) )
```

```
                 t2i(:,:,:,:,ic,jc) = t2i(:,:,:,:,ic,jc)        +
     &                ( t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) +
     &                  t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) )
              end do
           end do
        end do

        t1r = 0.0d0
        t1i = 0.0d0
        usr = cshift(cshift(ur(:,:,:,:,xhat,:,:),dim=xhat,shift=1),
     &               dim=yhat,shift=1)
        usi = cshift(cshift(ui(:,:,:,:,xhat,:,:),dim=xhat,shift=1),
     &               dim=yhat,shift=1)
        do ic=1,nc
           do jc=1,nc
              do kc=1,nc
                 t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)        +
     &                ( t2r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                  t2i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                 t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)        +
     &                (-t2r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                  t2i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
              end do
           end do
        end do

        t2r = 0.0d0
        t2i = 0.0d0
        usr = cshift(ur(:,:,:,:,xhat,:,:),dim=yhat,shift=1)
        usi = cshift(ui(:,:,:,:,xhat,:,:),dim=yhat,shift=1)
        do ic=1,nc
           do jc=1,nc
              do kc=1,nc
                 t2r(:,:,:,:,ic,jc) = t2r(:,:,:,:,ic,jc)        +
     &                ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                  t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                 t2i(:,:,:,:,ic,jc) = t2i(:,:,:,:,ic,jc)        +
     &                (-t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                  t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
              end do
           end do
        end do
c
c     Fnew = Fold + Im{t2*Udagger(x,nu)}/4
c
        do ic=1,nc
           do jc=1,nc
              do kc=1,nc
                 rectCr(:,:,:,:,ic,jc) = rectCr(:,:,:,:,ic,jc) +
     &                ( t2r(:,:,:,:,ic,kc) * ur(:,:,:,:,yhat,jc,kc)  +
     &                  t2i(:,:,:,:,ic,kc) * ui(:,:,:,:,yhat,jc,kc)  ) / 4.0d0
                 rectCi(:,:,:,:,ic,jc) = rectCi(:,:,:,:,ic,jc) +
     &                     (-t2r(:,:,:,:,ic,kc) * ui(:,:,:,:,yhat,jc,kc)  +
     &                       t2i(:,:,:,:,ic,kc) * ur(:,:,:,:,yhat,jc,kc)  ) / 4.0d0
              end do
           end do
        end do

c     1x2 upper forward.

        t1r = 0.0d0
        t1i = 0.0d0
        usr = cshift(ur(:,:,:,:,yhat,:,:),dim=xhat,shift=1)
        usi = cshift(ui(:,:,:,:,yhat,:,:),dim=xhat,shift=1)
        do ic=1,nc
           do jc=1,nc
              do kc=1,nc
                 t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)        +
     &                    ( ur(:,:,:,:,xhat,ic,kc) * usr(:,:,:,:,kc,jc) -
     &                      ui(:,:,:,:,xhat,ic,kc) * usi(:,:,:,:,kc,jc) )
                 t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)        +
     &                    ( ur(:,:,:,:,xhat,ic,kc) * usi(:,:,:,:,kc,jc) +
     &                      ui(:,:,:,:,xhat,ic,kc) * usr(:,:,:,:,kc,jc) )
              end do
           end do
        end do

        t2r = 0.0d0
        t2i = 0.0d0
        usr = cshift(cshift(ur(:,:,:,:,yhat,:,:),
     &                 dim=xhat,shift=1),dim=yhat,shift=1)
        usi = cshift(cshift(ui(:,:,:,:,yhat,:,:),
     &                 dim=xhat,shift=1),dim=yhat,shift=1)
        do ic=1,nc
           do jc=1,nc
              do kc=1,nc
                 t2r(:,:,:,:,ic,jc) = t2r(:,:,:,:,ic,jc)        +
     &                    ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) -
     &                      t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) )
                 t2i(:,:,:,:,ic,jc) = t2i(:,:,:,:,ic,jc)        +
     &                    ( t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) +
     &                      t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) )
              end do
           end do
        end do

        t1r = 0.0d0
        t1i = 0.0d0
        usr = cshift(ur(:,:,:,:,xhat,:,:),dim=yhat,shift=2)
        usi = cshift(ui(:,:,:,:,xhat,:,:),dim=yhat,shift=2)
        do ic=1,nc
           do jc=1,nc
              do kc=1,nc
                 t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)        +
     &                    ( t2r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                      t2i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                    t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)        +
     &                    (-t2r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                      t2i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
                 end do
              end do
           end do

           t2r = 0.0d0
           t2i = 0.0d0
           usr = cshift(ur(:,:,:,:,yhat,:,:),dim=yhat,shift=1)
           usi = cshift(ui(:,:,:,:,yhat,:,:),dim=yhat,shift=1)
           do ic=1,nc
              do jc=1,nc
                 do kc=1,nc
                    t2r(:,:,:,:,ic,jc) = t2r(:,:,:,:,ic,jc)        +
     &                    ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                      t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc)  )
                    t2i(:,:,:,:,ic,jc) = t2i(:,:,:,:,ic,jc)        +
     &                    (-t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                      1i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc)  )
                 end do
              end do
           end do
c
c     Fnew = Fold + Im{t2*Udagger(x,yhat)}/4
c
           do ic=1,nc
              do jc=1,nc
                 do kc=1,nc
                    rectCr(:,:,:,:,ic,jc) = rectCr(:,:,:,:,ic,jc) +
     &                    ( t2r(:,:,:,:,ic,kc) * ur(:,:,:,:,yhat,jc,kc)  +
     &                      t2i(:,:,:,:,ic,kc) * ui(:,:,:,:,yhat,jc,kc)  ) / 4.0d0
                    rectCi(:,:,:,:,ic,jc) = rectCi(:,:,:,:,ic,jc) +
     &                    (-t2r(:,:,:,:,ic,kc) * ui(:,:,:,:,yhat,jc,kc)  +
     &                      t2i(:,:,:,:,ic,kc) * ur(:,:,:,:,yhat,jc,kc)  ) / 4.0d0
                 end do
              end do
           end do

c     calculation of the second set 2x1 upper backward + 1x2 upper backward.

c     2x1 upper backward

           t1r = 0.0d0
           t1i = 0.0d0
           usr = cshift(cshift(ur(:,:,:,:,xhat,:,:),dim=xhat,shift=-1),
     &                  dim=yhat,shift=1)
           usi = cshift(cshift(ui(:,:,:,:,xhat,:,:),dim=xhat,shift=-1),
     &                  dim=yhat,shift=1)
           do ic=1,nc
```

```
            do jc=1,nc
               do kc=1,nc
                  t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)            +
     &                 ( ur(:,:,:,:,yhat,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                   ui(:,:,:,:,yhat,ic,kc) * usi(:,:,:,:,jc,kc) )
                  t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)            +
     &                 (-ur(:,:,:,:,yhat,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                   ui(:,:,:,:,yhat,ic,kc) * usr(:,:,:,:,jc,kc) )
               end do
            end do
         end do

         t2r = 0.0d0
         t2i = 0.0d0
         usr = cshift(cshift(ur(:,:,:,:,xhat,:,:),
     &                dim=xhat,shift=-2),dim=yhat,shift=1)
         usi = cshift(cshift(ui(:,:,:,:,xhat,:,:),
     &                dim=xhat,shift=-2),dim=yhat,shift=1)
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  t2r(:,:,:,:,ic,jc) = t2r(:,:,:,:,ic,jc)        +
     &                 ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                   t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                  t2i(:,:,:,:,ic,jc) = t2i(:,:,:,:,ic,jc)        +
     &                 (-t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                   t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
               end do
            end do
         end do

         t1r = 0.0d0
         t1i = 0.0d0
         usr = cshift(ur(:,:,:,:,yhat,:,:),dim=xhat,shift=-2)
         usi = cshift(ui(:,:,:,:,yhat,:,:),dim=xhat,shift=-2)
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)        +
     &                 ( t2r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                   t2i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                  t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)        +
     &                 (-t2r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                   t2i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
               end do
            end do
         end do

         t2r = 0.0d0
         t2i = 0.0d0
         usr = cshift(ur(:,:,:,:,xhat,:,:),dim=xhat,shift=-2)
         usi = cshift(ui(:,:,:,:,xhat,:,:),dim=xhat,shift=-2)
```

```
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     t2r(:,:,:,:,ic,jc) = t2r(:,:,:,:,ic,jc) +
     &                    ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) -
     &                      t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) )
                     t2i(:,:,:,:,ic,jc) = t2i(:,:,:,:,ic,jc) +
     &                    ( t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) +
     &                      t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) )
                  end do
               end do
            end do
c
c     Fnew = Fold + Im{t2*Udagger(x-xhat,xhat)}/4
c
            usr = cshift(ur(:,:,:,:,xhat,:,:),dim=xhat,shift=-1)
            usi = cshift(ui(:,:,:,:,xhat,:,:),dim=xhat,shift=-1)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     rectCr(:,:,:,:,ic,jc) = rectCr(:,:,:,:,ic,jc) +
     &                    ( t2r(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc)  -
     &                      t2i(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc)  ) / 4.0d0
                     rectCi(:,:,:,:,ic,jc) = rectCi(:,:,:,:,ic,jc) +
     &                    ( t2r(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc)  +
     &                      t2i(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc)  ) / 4.0d0
                  end do
               end do
            end do

c     1x2 upper backward

            t1r = 0.0d0
            t1i = 0.0d0
            usr = cshift(ur(:,:,:,:,yhat,:,:),dim=yhat,shift=1)
            usi = cshift(ui(:,:,:,:,yhat,:,:),dim=yhat,shift=1)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)        +
     &                    ( ur(:,:,:,:,yhat,ic,kc) * usr(:,:,:,:,kc,jc) -
     &                      ui(:,:,:,:,yhat,ic,kc) * usi(:,:,:,:,kc,jc) )
                     t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)        +
     &                    ( ur(:,:,:,:,yhat,ic,kc) * usi(:,:,:,:,kc,jc) +
     &                      ui(:,:,:,:,yhat,ic,kc) * usr(:,:,:,:,kc,jc) )
                  end do
               end do
            end do

            t2r = 0.0d0
            t2i = 0.0d0
            usr = cshift(cshift(ur(:,:,:,:,xhat,:,:),dim=xhat,shift=-1),
     &                 dim=yhat,shift=2)
            usi = cshift(cshift(ui(:,:,:,:,xhat,:,:),dim=xhat,shift=-1),
     &                 dim=yhat,shift=2)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     t2r(:,:,:,:,ic,jc) = t2r(:,:,:,:,ic,jc)       +
     &                    ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                      t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                     t2i(:,:,:,:,ic,jc) = t2i(:,:,:,:,ic,jc)       +
     &                    (-t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                      t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
                  end do
               end do
            end do

            t1r = 0.0d0
            t1i = 0.0d0
            usr = cshift(cshift(ur(:,:,:,:,yhat,:,:),dim=xhat,shift=-1),dim=yhat,shift=1)
            usi = cshift(cshift(ui(:,:,:,:,yhat,:,:),dim=xhat,shift=-1),dim=yhat,shift=1)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)       +
     &                    ( t2r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                      t2i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                     t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)       +
     &                    (-t2r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                      t2i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
                  end do
               end do
            end do

            t2r = 0.0d0
            t2i = 0.0d0
            usr = cshift(ur(:,:,:,:,yhat,:,:),dim=xhat,shift=-1)
            usi = cshift(ui(:,:,:,:,yhat,:,:),dim=xhat,shift=-1)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     t2r(:,:,:,:,ic,jc) = t2r(:,:,:,:,ic,jc)       +
     &                    ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                      t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                     t2i(:,:,:,:,ic,jc) = t2i(:,:,:,:,ic,jc)       +
     &                    (-t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                      t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
                  end do
               end do
            end do
c
c     Fnew = Fold + Im{t2*U(x-xhat,xhat)}/4
c
```

```
            usr = cshift(ur(:,:,:,:,xhat,:,:),dim=xhat,shift=-1)
            usi = cshift(ui(:,:,:,:,xhat,:,:),dim=xhat,shift=-1)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     rectCr(:,:,:,:,ic,jc) = rectCr(:,:,:,:,ic,jc) +
     &                    ( t2r(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc)  -
     &                      t2i(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc)  ) / 4.0d0
                     rectCi(:,:,:,:,ic,jc) = rectCi(:,:,:,:,ic,jc) +
     &                    ( t2r(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc)  +
     &                      t2i(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc)  ) / 4.0d0
                  end do
               end do
            end do

c     calculation of the third set 2x1 downer backward + 1x2 downer backward.

c     2x1 downer backward

            t1r = 0.0d0
            t1i = 0.0d0
            t2r = cshift(ur(:,:,:,:,xhat,:,:),dim=xhat,shift=-1)
            t2i = cshift(ui(:,:,:,:,xhat,:,:),dim=xhat,shift=-1)
            usr = cshift(ur(:,:,:,:,xhat,:,:),dim=xhat,shift=-2)
            usi = cshift(ui(:,:,:,:,xhat,:,:),dim=xhat,shift=-2)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)        +
     &                    ( t2r(:,:,:,:,kc,ic) * usr(:,:,:,:,jc,kc) -
     &                      t2i(:,:,:,:,kc,ic) * usi(:,:,:,:,jc,kc) )
                     t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)        +
     &                    (-t2r(:,:,:,:,kc,ic) * usi(:,:,:,:,jc,kc) -
     &                      t2i(:,:,:,:,kc,ic) * usr(:,:,:,:,jc,kc) )
                  end do
               end do
            end do

            t2r = 0.0d0
            t2i = 0.0d0
            usr = cshift(cshift(ur(:,:,:,:,yhat,:,:),
     &                   dim=xhat,shift=-2),dim=yhat,shift=-1)
            usi = cshift(cshift(ui(:,:,:,:,yhat,:,:),
     &                   dim=xhat,shift=-2),dim=yhat,shift=-1)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     t2r(:,:,:,:,ic,jc) = t2r(:,:,:,:,ic,jc)        +
     &                    ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                      t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                     t2i(:,:,:,:,ic,jc) = t2i(:,:,:,:,ic,jc)        +
     &                    (-t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                      t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
                  end do
               end do
            end do

            t1r = 0.0d0
            t1i = 0.0d0
            usr = cshift(cshift(ur(:,:,:,:,xhat,:,:),
     &                  dim=xhat,shift=-2),dim=yhat,shift=-1)
            usi = cshift(cshift(ui(:,:,:,:,xhat,:,:),
     &                  dim=xhat,shift=-2),dim=yhat,shift=-1)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)        +
     &                    ( t2r(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) -
     &                      t2i(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) )
                     t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)        +
     &                    ( t2r(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) +
     &                      t2i(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) )
                  end do
               end do
            end do

            t2r = 0.0d0
            t2i = 0.0d0
            usr = cshift(cshift(ur(:,:,:,:,xhat,:,:),
     &                  dim=xhat,shift=-1),dim=yhat,shift=-1)
            usi = cshift(cshift(ui(:,:,:,:,xhat,:,:),
     &                  dim=xhat,shift=-1),dim=yhat,shift=-1)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     t2r(:,:,:,:,ic,jc) = t2r(:,:,:,:,ic,jc)         +
     &                    ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc)  -
     &                      t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc)  )
                     t2i(:,:,:,:,ic,jc) = t2i(:,:,:,:,ic,jc)         +
     &                    ( t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc)  +
     &                      t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc)  )
                  end do
               end do
            end do
c
c     Fnew = Fold + Im{t2*U(x-yhat,yhat)}/4
c
            usr = cshift(ur(:,:,:,:,yhat,:,:),dim=yhat,shift=-1)
            usi = cshift(ui(:,:,:,:,yhat,:,:),dim=yhat,shift=-1)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     rectCr(:,:,:,:,ic,jc) = rectCr(:,:,:,:,ic,jc) +
     &                    ( t2r(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc)  -
     &                     t2i(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc)  ) / 4.0d0
                     rectCi(:,:,:,:,ic,jc) = rectCi(:,:,:,:,ic,jc) +
     &                    ( t2r(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc)  +
     &                      t2i(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc)  ) / 4.0d0
                  end do
               end do
            end do

c     1x2 downer backward.

            t1r = 0.0d0
            t1i = 0.0d0
            t2r = cshift(ur(:,:,:,:,xhat,:,:),dim=xhat,shift=-1)
            t2i = cshift(ui(:,:,:,:,xhat,:,:),dim=xhat,shift=-1)
            usr = cshift(cshift(ur(:,:,:,:,yhat,:,:),
     &                dim=xhat,shift=-1),dim=yhat,shift=-1)
            usi = cshift(cshift(ui(:,:,:,:,yhat,:,:),
     &                dim=xhat,shift=-1),dim=yhat,shift=-1)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)       +
     &                    ( t2r(:,:,:,:,kc,ic) * usr(:,:,:,:,jc,kc) -
     &                      t2i(:,:,:,:,kc,ic) * usi(:,:,:,:,jc,kc) )
                     t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)       +
     &                    (-t2r(:,:,:,:,kc,ic) * usi(:,:,:,:,jc,kc) -
     &                      t2i(:,:,:,:,kc,ic) * usr(:,:,:,:,jc,kc) )
                  end do
               end do
            end do

            t2r = 0.0d0
            t2i = 0.0d0
            usr = cshift(cshift(ur(:,:,:,:,yhat,:,:),
     &                dim=xhat,shift=-1),dim=yhat,shift=-2)
            usi = cshift(cshift(ui(:,:,:,:,yhat,:,:),
     &                dim=xhat,shift=-1),dim=yhat,shift=-2)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     t2r(:,:,:,:,ic,jc) = t2r(:,:,:,:,ic,jc)       +
     &                    ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) +
     &                      t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) )
                     t2i(:,:,:,:,ic,jc) = t2i(:,:,:,:,ic,jc)       +
     &                    (-t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc) +
     &                      t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc) )
                  end do
               end do
            end do

            t1r = 0.0d0
            t1i = 0.0d0
```

```
            usr = cshift(cshift(ur(:,:,:,:,xhat,:,:),
     &                   dim=xhat,shift=-1),dim=yhat,shift=-2)
            usi = cshift(cshift(ui(:,:,:,:,xhat,:,:),
     &                   dim=xhat,shift=-1),dim=yhat,shift=-2)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)        +
     &                  ( t2r(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) -
     &                    t2i(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) )
                     t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)        +
     &                  ( t2r(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) +
     &                    t2i(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) )
                  end do
               end do
            end do

            t2r = 0.0d0
            t2i = 0.0d0
            usr = cshift(ur(:,:,:,:,yhat,:,:),dim=yhat,shift=-2)
            usi = cshift(ui(:,:,:,:,yhat,:,:),dim=yhat,shift=-2)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     t2r(:,:,:,:,ic,jc) = t2r(:,:,:,:,ic,jc) +
     &                  ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc)  -
     &                    t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc)  )
                     t2i(:,:,:,:,ic,jc) = t2i(:,:,:,:,ic,jc) +
     &                  ( t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc)  +
     &                    t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc)  )
                  end do
               end do
            end do
c
c     Fnew = Fold + Im{t2*U(x-yhat,yhat)}/4
c
            usr = cshift(ur(:,:,:,:,yhat,:,:),dim=yhat,shift=-1)
            usi = cshift(ui(:,:,:,:,yhat,:,:),dim=yhat,shift=-1)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     rectCr(:,:,:,:,ic,jc) = rectCr(:,:,:,:,ic,jc) +
     &                  ( t2r(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc)  -
     &                    t2i(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc)  ) / 4.0d0
                     rectCi(:,:,:,:,ic,jc) = rectCi(:,:,:,:,ic,jc) +
     &                  ( t2r(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc)  +
     &                    t2i(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc)  ) / 4.0d0
                  end do
               end do
            end do

c     calculation of the fourth set 2x1 downer forward + 1x2 downer forward.
```

```
c     2x1 downer forward

            t1r = 0.0d0
            t1i = 0.0d0
            t2r = cshift(ur(:,:,:,:,yhat,:,:),dim=yhat,shift=-1)
            t2i = cshift(ui(:,:,:,:,yhat,:,:),dim=yhat,shift=-1)
            usr = cshift(ur(:,:,:,:,xhat,:,:),dim=yhat,shift=-1)
            usi = cshift(ui(:,:,:,:,xhat,:,:),dim=yhat,shift=-1)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)       +
     &                   ( t2r(:,:,:,:,kc,ic) * usr(:,:,:,:,kc,jc) +
     &                     t2i(:,:,:,:,kc,ic) * usi(:,:,:,:,kc,jc) )
                     t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)       +
     &                   ( t2r(:,:,:,:,kc,ic) * usi(:,:,:,:,kc,jc) -
     &                     t2i(:,:,:,:,kc,ic) * usr(:,:,:,:,kc,jc) )
                  end do
               end do
            end do

            t2r = 0.0d0
            t2i = 0.0d0
            usr = cshift(cshift(ur(:,:,:,:,xhat,:,:),dim=xhat,shift=1),
     &                  dim=yhat,shift=-1)
            usi = cshift(cshift(ui(:,:,:,:,xhat,:,:),dim=xhat,shift=1),
     &                  dim=yhat,shift=-1)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     t2r(:,:,:,:,ic,jc) = t2r(:,:,:,:,ic,jc)       +
     &                   ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) -
     &                     t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) )
                     t2i(:,:,:,:,ic,jc) = t2i(:,:,:,:,ic,jc)       +
     &                   ( t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) +
     &                     t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) )
                  end do
               end do
            end do

            t1r = 0.0d0
            t1i = 0.0d0
            usr = cshift(cshift(ur(:,:,:,:,yhat,:,:),
     &                  dim=xhat,shift=2),dim=yhat,shift=-1)
            usi = cshift(cshift(ui(:,:,:,:,yhat,:,:),
     &                  dim=xhat,shift=2),dim=yhat,shift=-1)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)       +
     &                   ( t2r(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) -
     &                    t2i(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) )
                     t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)        +
     &                   ( t2r(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) +
     &                     t2i(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) )
                  end do
               end do
            end do

            t2r = 0.0d0
            t2i = 0.0d0
            usr = cshift(ur(:,:,:,:,xhat,:,:),dim=xhat,shift=1)
            usi = cshift(ui(:,:,:,:,xhat,:,:),dim=xhat,shift=1)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     t2r(:,:,:,:,ic,jc) = t2r(:,:,:,:,ic,jc) +
     &                   ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc)  +
     &                     t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc)  )
                     t2i(:,:,:,:,ic,jc) = t2i(:,:,:,:,ic,jc) +
     &                   (-t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,jc,kc)  +
     &                     t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,jc,kc)  )
                  end do
               end do
            end do
c
c     Fnew = Fold + Im{t2*Udagger(x,xhat)}/4
c
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     rectCr(:,:,:,:,ic,jc) = rectCr(:,:,:,:,ic,jc) +
     &                   ( t2r(:,:,:,:,ic,kc) * ur(:,:,:,:,xhat,jc,kc)  +
     &                     t2i(:,:,:,:,ic,kc) * ui(:,:,:,:,xhat,jc,kc)  ) / 4.0d0
                     rectCi(:,:,:,:,ic,jc) = rectCi(:,:,:,:,ic,jc) +
     &                   (-t2r(:,:,:,:,ic,kc) * ui(:,:,:,:,xhat,jc,kc)  +
     &                     t2i(:,:,:,:,ic,kc) * ur(:,:,:,:,xhat,jc,kc)  ) / 4.0d0
                  end do
               end do
            end do

c     1x2 downer forward.

            t1r = 0.0d0
            t1i = 0.0d0
            t2r = cshift(ur(:,:,:,:,yhat,:,:),dim=yhat,shift=-1)
            t2i = cshift(ui(:,:,:,:,yhat,:,:),dim=yhat,shift=-1)
            usr = cshift(ur(:,:,:,:,yhat,:,:),dim=yhat,shift=-2)
            usi = cshift(ui(:,:,:,:,yhat,:,:),dim=yhat,shift=-2)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)        +
     &                      ( t2r(:,:,:,:,kc,ic) * usr(:,:,:,:,jc,kc) -
     &                        t2i(:,:,:,:,kc,ic) * usi(:,:,:,:,jc,kc) )
                     t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)       +
     &                      (-t2r(:,:,:,:,kc,ic) * usi(:,:,:,:,jc,kc) -
     &                        t2i(:,:,:,:,kc,ic) * usr(:,:,:,:,jc,kc) )
                  end do
               end do
            end do

            t2r = 0.0d0
            t2i = 0.0d0
            usr = cshift(ur(:,:,:,:,xhat,:,:),dim=yhat,shift=-2)
            usi = cshift(ui(:,:,:,:,xhat,:,:),dim=yhat,shift=-2)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     t2r(:,:,:,:,ic,jc) = t2r(:,:,:,:,ic,jc)       +
     &                      ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) -
     &                        t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) )
                     t2i(:,:,:,:,ic,jc) = t2i(:,:,:,:,ic,jc)       +
     &                      ( t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) +
     &                        t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) )
                  end do
               end do
            end do

            t1r = 0.0d0
            t1i = 0.0d0
            usr = cshift(cshift(ur(:,:,:,:,yhat,:,:),
     &                     dim=xhat,shift=1),dim=yhat,shift=-2)
            usi = cshift(cshift(ui(:,:,:,:,yhat,:,:),
     &                     dim=xhat,shift=1),dim=yhat,shift=-2)
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     t1r(:,:,:,:,ic,jc) = t1r(:,:,:,:,ic,jc)       +
     &                      ( t2r(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) -
     &                        t2i(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) )
                     t1i(:,:,:,:,ic,jc) = t1i(:,:,:,:,ic,jc)       +
     &                      ( t2r(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) +
     &                        t2i(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) )
                  end do
               end do
            end do

            t2r = 0.0d0
            t2i = 0.0d0
            usr = cshift(cshift(ur(:,:,:,:,yhat,:,:),dim=xhat,shift=1),
     &                     dim=yhat,shift=-1)
            usi = cshift(cshift(ui(:,:,:,:,yhat,:,:),dim=xhat,shift=1),
     &                     dim=yhat,shift=-1)
            do ic=1,nc
```

```
                do jc=1,nc
                   do kc=1,nc
                      t2r(:,:,:,:,ic,jc) = t2r(:,:,:,:,ic,jc)       +
     &                   ( t1r(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) -
     &                     t1i(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) )
                      t2i(:,:,:,:,ic,jc) = t2i(:,:,:,:,ic,jc)       +
     &                   ( t1r(:,:,:,:,ic,kc) * usi(:,:,:,:,kc,jc) +
     &                     t1i(:,:,:,:,ic,kc) * usr(:,:,:,:,kc,jc) )
                   end do
                end do
             end do
c
c     Fnew = Fold + Im{t2*Udagger(x,xhat)}/4
c
             do ic=1,nc
                do jc=1,nc
                   do kc=1,nc
                      rectCr(:,:,:,:,ic,jc) = rectCr(:,:,:,:,ic,jc) +
     &                   ( t2r(:,:,:,:,ic,kc) * ur(:,:,:,:,xhat,jc,kc)  +
     &                     t2i(:,:,:,:,ic,kc) * ui(:,:,:,:,xhat,jc,kc)  ) / 4.0d0
                      rectCi(:,:,:,:,ic,jc) = rectCi(:,:,:,:,ic,jc) +
     &                   (-t2r(:,:,:,:,ic,kc) * ui(:,:,:,:,xhat,jc,kc)  +
     &                     t2i(:,:,:,:,ic,kc) * ur(:,:,:,:,xhat,jc,kc)  ) / 4.0d0
                   end do
                end do
             end do

      return
      end subroutine impclover
```

E.18 Cooling the $SU_c(3)$ matrices

```
c
c     cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Author: Frederic D.R. Bonnet & D.B. Leinweber: February 1999.
c     subroutine coolSweep updates the current link using a psedo-heatbath
c     algorithm. It calls the staples and cooling.
c
      subroutine coolSweep(ur,ui,mask,umask,beta,itype,nsub,uzero)
      implicit none
      include 'latticeSize.h'

c     global variables

      integer,parameter                                       :: nc=3,ncsu2=2
      integer,parameter                                       :: nsigma=ncsu2*ncsu2
      integer,parameter                                       :: mu=4
      integer,parameter                                       :: nmask=16

      integer                                                 :: itype
      integer                                                 :: nsub
      integer                                                  :: umask

      double precision                                         :: uzero
      double precision                                         :: beta

      double precision,dimension(nx,ny,nz,nt,mu,nc,nc)         :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)
      logical,dimension(nx,ny,nz,nt,mu,nmask)                  :: mask
cmf$  layout mask(:news,:news,:news,:news,:serial,:serial)

c     local variables

      double precision,dimension(nx,ny,nz,nt,nc,nc)            :: stapler,staplei
      double precision,dimension(nx,ny,nz,nt,nc,nc)            :: ltsr,ltsi
      double precision,dimension(nx,ny,nz,nt,nc,nc)            :: urprmsu3,uiprmsu3
cmf$  layout stapler(:news,:news,:news,:news,:serial,:serial)
cmf$  layout staplei(:news,:news,:news,:news,:serial,:serial)
cmf$  layout ltsr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout ltsi(:news,:news,:news,:news,:serial,:serial)
cmf$  layout urprmsu3(:news,:news,:news,:news,:serial,:serial)
cmf$  layout uiprmsu3(:news,:news,:news,:news,:serial,:serial)
      double precision,dimension(nx,ny,nz,nt,ncsu2,ncsu2)      :: phbsr,phbsi
      double precision,dimension(nx,ny,nz,nt,ncsu2,ncsu2)      :: ursu2,uisu2
cmf$  layout phbsr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout phbsi(:news,:news,:news,:news,:serial,:serial)
cmf$  layout ursu2(:news,:news,:news,:news,:serial,:serial)
cmf$  layout uisu2(:news,:news,:news,:news,:serial,:serial)

      double precision                                         :: betanew
      integer                                                  :: of1
      integer                                                  :: imask,ihat,ic,jc,kc,ic3
      integer                                                  :: isub

      interface
         SUBROUTINE staples(ur,ui,stapler,staplei,xhat,local,itype,uzero)
          IMPLICIT NONE
          include 'latticeSize.h'
          integer,parameter                                    :: nc=3
          integer,parameter                                    :: mu=4
          integer                                              :: xhat
          integer                                              :: itype
          double precision                                     :: uzero
          logical                                              :: local
          double precision,dimension(nx,ny,nz,nt,nc,nc)        :: stapler,staplei
cmf$  layout stapler(:news,:news,:news,:news,:serial,:serial)
cmf$  layout staplei(:news,:news,:news,:news,:serial,:serial)
          double precision,dimension(nx,ny,nz,nt,mu,nc,nc)     :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)
         end SUBROUTINE staples
         subroutine cooling(urnewsu2,uinewsu2,phbsr,phbsi)
```

```
          implicit none
          include 'latticeSize.h'
          integer,parameter                                 :: ncsu2=2
          double precision,dimension(nx,ny,nz,nt,ncsu2,ncsu2) :: phbsr,phbsi
          double precision,dimension(nx,ny,nz,nt,ncsu2,ncsu2) :: urnewsu2,uinewsu2
cmf$  layout phbsr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout phbsi(:news,:news,:news,:news,:serial,:serial)
cmf$  layout urnewsu2(:news,:news,:news,:news,:serial,:serial)
cmf$  layout uinewsu2(:news,:news,:news,:news,:serial,:serial)
         end subroutine cooling
      end interface

c     starts of the running commands

c     Here we need to multiply Beta by a factor of 2/nc=3 because for the su2 case
c     there is a factor of which cancels the one 1/ncsu2 in the probability
c     distribution function. This beta value is to be passed into cooling

      betanew = ( 2.0d0 / nc ) * beta

c     the ihat do loop, is to loop over all the possible Euclidean directions.
c     The imask do loop, is to ensure that the entire lattice is considered:
c     all true then all false.

      do ihat=1,mu
         do imask=1,umask

c     calculate the staple in the ihat direction

            call staples(ur,ui,stapler,staplei,ihat,.true.,itype,uzero)

            do isub=1,nsub
c
c     case 1. a_1
c
c     The Wilson action can be written as S(U)=Sum_p(Re(U*U_p))+constant
c                                             =Re(Tr(U*R))+constant
c     R is just the sum over the six plaquette namely the staples: stapler,staplei.

c     now we need to calculate the product of the staples time the link variable in
c     each direction U*R=ltsr+iltsi=( ur + iui ) * ( stapler + istaplei )
c                                 =(ur*stapler - ui*staplei)+i(ur*staplei + ui*staplei)
            ltsr = 0.0d0
            ltsi = 0.0d0
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     ltsr(:,:,:,:,ic,jc) = ltsr(:,:,:,:,ic,jc) +
     &                  ( ur(:,:,:,:,ihat,ic,kc) * stapler(:,:,:,:,kc,jc) -
     &                    ui(:,:,:,:,ihat,ic,kc) * staplei(:,:,:,:,kc,jc) )
                     ltsi(:,:,:,:,ic,jc) = ltsi(:,:,:,:,ic,jc) +
     &                  ( ur(:,:,:,:,ihat,ic,kc) * staplei(:,:,:,:,kc,jc) +
     &                     ui(:,:,:,:,ihat,ic,kc) * stapler(:,:,:,:,kc,jc) )
                 end do
              end do
           end do

c     The distribution we are interested in the following
c     dP(a_k) = d^{(k)}a_k * exp[-beta * S(a_k*U^{(k-1)})]/Z_k(U^{(k-1)})
c     where d^{(k)}a_k is the Haar measure on the SU(2)_k and
c     Z_k(U^{(k-1)}) = \int_{SU(2)_k} da exp[-beta * S(aU)]. If a{\in}SU(2)_K then
c     Z_k(aU) = Z_k(U). In order to generate the above distribution for the minimal
c     set F of SU(2) subgroup of SU(N) namely {F:SU(2)_k, k=1,..,nc-1}, let's first
c     note that S(a_k*U)=Re(Tr a_k * U*R) = Re(Tr \alpha_k * r_k)
c                                        + terms independent of \alpha_k.
c     Where r_k is the 2x2 subgroup of the U*R (i.e. link*staples : lts), this matix
c     has the same block structure as the matrix a_k. This means that \alpha_k and
c     r_k = r_0 * I + i \vec{\sigma}\dot\vec{r} are the same block in the ncxnc
c     matrix located at the (k,k+1)th rows and column.

c     For the first call of the pseudo-heatbath we need to match the matrix block,
c     if this is not done wrong information will be sent to the subroutine therefore
c     returning the wrong numbers. To do that we use an offset parameter called of1
c     with this of1 we can acces the correct entries of the ncxnc lts matrix. We then
c     reconstruct a SU(2) matrix that can be passed into cooling which will also
c     return a SU(2) matrix.

           of1 = 0

           phbsr(:,:,:,:,1,1) = (  ltsr(:,:,:,:,1+of1,1+of1) +
     &                             ltsr(:,:,:,:,2+of1,2+of1) ) / 2.0d0
           phbsr(:,:,:,:,1,2) = (  ltsr(:,:,:,:,1+of1,2+of1) -
     &                             ltsr(:,:,:,:,2+of1,1+of1) ) / 2.0d0
           phbsr(:,:,:,:,2,1) = - phbsr(:,:,:,:,1,2)
           phbsr(:,:,:,:,2,2) =   phbsr(:,:,:,:,1,1)

           phbsi(:,:,:,:,1,1) = (  ltsi(:,:,:,:,1+of1,1+of1) -
     &                             ltsi(:,:,:,:,2+of1,2+of1) ) / 2.0d0
           phbsi(:,:,:,:,1,2) = (  ltsi(:,:,:,:,1+of1,2+of1) +
     &                             ltsi(:,:,:,:,2+of1,1+of1) ) / 2.0d0
           phbsi(:,:,:,:,2,1) =   phbsi(:,:,:,:,1,2)
           phbsi(:,:,:,:,2,2) = - phbsi(:,:,:,:,1,1)

c     now calling the pseudo-heatbath algorithm to update, where the
c     mask is .true., the full QCD configuration

c     we call cooling to form 2 SU(3) matrices.
c     These two matrices have the form matrixa = [ [SU(2)] 0 ],  matrixb[ 1    0    ].
c                                                [    0    1 ],         [ 0 [SU(2)] ]
c     Where [SU(2)] is hotwired via ursu2,uisu2 and ursu2,uisu2.

           call cooling(ursu2,uisu2,phbsr,phbsi)

c     We next make a product of the matrices in such a way that
```

```
c     link Uprmsu3 = urprmsu3+iuiprmsu3 = ( ar + iai ) * ( ur + iui )
c                                 = ( ar*ur - ai*ui ) + i( ar*ui + ai*ur)
c     by hotwiring the matrix indices for optimization.

            do ic=1,nc-1
               do jc=1,nc
                  where( mask(:,:,:,:,ihat,imask) )
              urprmsu3(:,:,:,:,ic,jc) = ( ursu2(:,:,:,:,ic,1) * ur(:,:,:,:,ihat,1,jc) +
     &                                    ursu2(:,:,:,:,ic,2) * ur(:,:,:,:,ihat,2,jc) -
     &                                    uisu2(:,:,:,:,ic,1) * ui(:,:,:,:,ihat,1,jc) -
     &                                    uisu2(:,:,:,:,ic,2) * ui(:,:,:,:,ihat,2,jc) )
              uiprmsu3(:,:,:,:,ic,jc) = ( ursu2(:,:,:,:,ic,1) * ui(:,:,:,:,ihat,1,jc) +
     &                                    ursu2(:,:,:,:,ic,2) * ui(:,:,:,:,ihat,2,jc) +
     &                                    uisu2(:,:,:,:,ic,1) * ur(:,:,:,:,ihat,1,jc) +
     &                                    uisu2(:,:,:,:,ic,2) * ur(:,:,:,:,ihat,2,jc) )
                  elsewhere
                     urprmsu3(:,:,:,:,ic,jc) = ur(:,:,:,:,ihat,ic,jc)
                     uiprmsu3(:,:,:,:,ic,jc) = ui(:,:,:,:,ihat,ic,jc)
                  end where
               end do
            end do

            do jc=1,nc
               urprmsu3(:,:,:,:,3,jc) = ur(:,:,:,:,ihat,3,jc)
               uiprmsu3(:,:,:,:,3,jc) = ui(:,:,:,:,ihat,3,jc)
            end do
c
c     case 2. a_2
c
c     here we calculate the next bit of the product, link*staples.
c     We use the old staples to multiply it with Uprmsu3 = a_1 * U
c     lts = matrixa = [ [SU(2)] 0 ]*lts[ ele in SU(3)    0    ].
c                     [    0    1 ]    [      0          0    ]

            ltsr = 0.0d0
            ltsi = 0.0d0
            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     ltsr(:,:,:,:,ic,jc) = ltsr(:,:,:,:,ic,jc) +
     &                   ( urprmsu3(:,:,:,:,ic,kc) * stapler(:,:,:,:,kc,jc) -
     &                     uiprmsu3(:,:,:,:,ic,kc) * staplei(:,:,:,:,kc,jc) )
                     ltsi(:,:,:,:,ic,jc) = ltsi(:,:,:,:,ic,jc) +
     &                   ( urprmsu3(:,:,:,:,ic,kc) * staplei(:,:,:,:,kc,jc) +
     &                     uiprmsu3(:,:,:,:,ic,kc) * stapler(:,:,:,:,kc,jc) )
                  end do
               end do
            end do

c     Here again after having calculated the new ltsr and ltsi ( Uprmsu3*R ) with
c     the old staples R, Uprmsu3 = a_1 * U. We use an offset of1=1 to access the
c     lower block of the ncxnc lts matrix. Then once again we reshape these
```

```
c     entries by hardwiring them into a SU(2) matrix called pseudo-heatbath
c     staples namely phbsr and phbsi. The resulting SU(2) matrix is then
c     passed into cooling.

            of1 = 1

            phbsr(:,:,:,:,1,1) = (  ltsr(:,:,:,:,1+of1,1+of1) +
     &                              ltsr(:,:,:,:,2+of1,2+of1) ) / 2.0d0
            phbsr(:,:,:,:,1,2) = (  ltsr(:,:,:,:,1+of1,2+of1) -
     &                              ltsr(:,:,:,:,2+of1,1+of1) ) / 2.0d0
            phbsr(:,:,:,:,2,1) = - phbsr(:,:,:,:,1,2)
            phbsr(:,:,:,:,2,2) =   phbsr(:,:,:,:,1,1)


            phbsi(:,:,:,:,1,1) = (  ltsi(:,:,:,:,1+of1,1+of1) -
     &                              ltsi(:,:,:,:,2+of1,2+of1) ) / 2.0d0
            phbsi(:,:,:,:,1,2) = (  ltsi(:,:,:,:,1+of1,2+of1) +
     &                              ltsi(:,:,:,:,2+of1,1+of1) ) / 2.0d0
            phbsi(:,:,:,:,2,1) =   phbsi(:,:,:,:,1,2)
            phbsi(:,:,:,:,2,2) = - phbsi(:,:,:,:,1,1)

c     we next make a product of the matrices in such a way that link
c     Udblerpm=urdbleprmsu3+iuidbleprmsu3
c             =( br + ibi ) * ( urprmsu3 + iuiprmsu3 )
c             =( br*urprmsu3 - bi*uiprmsu3 ) + i( br*uiprmsu3 + bi*urprmsu3)
c             = a_2 * Uprmsu3
c     Where the a_1 and a_2 have the form of matrixa and matrixb mentioned above.

c     Calling cooling to obtain two new SU(2) matrices with the linkprmsu3*staples
c     calculated above.
c     Allocating by hardwiring to the urnew and uinew which are the return
c     variables of these products.

            call cooling(ursu2,uisu2,phbsr,phbsi)

c     Now mapping these variables to the full QCD link, ur and ui. For each
c     direction mu, the main loop imu.

            do ic=2,nc
               do jc=1,nc
                  where( mask(:,:,:,:,ihat,imask) )
              ur(:,:,:,:,ihat,ic,jc) = ( ursu2(:,:,:,:,ic-1,1) * urprmsu3(:,:,:,:,2,jc) +
     &                                   ursu2(:,:,:,:,ic-1,2) * urprmsu3(:,:,:,:,3,jc) -
     &                                   uisu2(:,:,:,:,ic-1,1) * uiprmsu3(:,:,:,:,2,jc) -
     &                                   uisu2(:,:,:,:,ic-1,2) * uiprmsu3(:,:,:,:,3,jc) )
              ui(:,:,:,:,ihat,ic,jc) = ( ursu2(:,:,:,:,ic-1,1) * uiprmsu3(:,:,:,:,2,jc) +
     &                                   ursu2(:,:,:,:,ic-1,2) * uiprmsu3(:,:,:,:,3,jc) +
     &                                   uisu2(:,:,:,:,ic-1,1) * urprmsu3(:,:,:,:,2,jc) +
     &                                   uisu2(:,:,:,:,ic-1,2) * urprmsu3(:,:,:,:,3,jc) )
                  elsewhere
                     ur(:,:,:,:,ihat,ic,jc) = urprmsu3(:,:,:,:,ic,jc)
                     ui(:,:,:,:,ihat,ic,jc) = uiprmsu3(:,:,:,:,ic,jc)
              end where
           end do
        end do

        do jc=1,nc
           ur(:,:,:,:,ihat,1,jc) = urprmsu3(:,:,:,:,1,jc)
           ui(:,:,:,:,ihat,1,jc) = uiprmsu3(:,:,:,:,1,jc)
        end do
c
c     case 3. a_3
c
c     forming another SU(2) subgroup of the form
c     These two matrices have the form matrixa = [ x  0   x ]
c                                                [ 0  1   0 ]
c                                                [ x  0   x ]
        ltsr = 0.0d0
        ltsi = 0.0d0
        do ic=1,nc
           do jc=1,nc
              do kc=1,nc
                 ltsr(:,:,:,:,ic,jc) = ltsr(:,:,:,:,ic,jc) +
     &                 ( ur(:,:,:,:,ihat,ic,kc) * stapler(:,:,:,:,kc,jc) -
     &                   ui(:,:,:,:,ihat,ic,kc) * staplei(:,:,:,:,kc,jc) )
                 ltsi(:,:,:,:,ic,jc) = ltsi(:,:,:,:,ic,jc) +
     &                 ( ur(:,:,:,:,ihat,ic,kc) * staplei(:,:,:,:,kc,jc) +
     &                   ui(:,:,:,:,ihat,ic,kc) * stapler(:,:,:,:,kc,jc) )
              end do
           end do
        end do


        phbsr(:,:,:,:,1,1) = (  ltsr(:,:,:,:,1,1) + ltsr(:,:,:,:,3,3) ) / 2.0d0
        phbsr(:,:,:,:,1,2) = (  ltsr(:,:,:,:,1,3) - ltsr(:,:,:,:,3,1) ) / 2.0d0
        phbsr(:,:,:,:,2,1) = - phbsr(:,:,:,:,1,2)
        phbsr(:,:,:,:,2,2) =   phbsr(:,:,:,:,1,1)

        phbsi(:,:,:,:,1,1) = (  ltsi(:,:,:,:,1,1) - ltsi(:,:,:,:,3,3) ) / 2.0d0
        phbsi(:,:,:,:,1,2) = (  ltsi(:,:,:,:,1,3) + ltsi(:,:,:,:,3,1) ) / 2.0d0
        phbsi(:,:,:,:,2,1) =   phbsi(:,:,:,:,1,2)
        phbsi(:,:,:,:,2,2) = - phbsi(:,:,:,:,1,1)

c     now calling the cooling algorithm.

        call cooling(ursu2,uisu2,phbsr,phbsi)

c     We next make a product of the matrices in such a way that
c     link Uprmsu3 = urprmsu3+iuiprmsu3 = ( ar + iai ) * ( ur + iui )
c                                = ( ar*ur - ai*ui ) + i( ar*ui + ai*ur)
c     by hotwiring the matrix indices for optimization.

        do ic=1,nc-1
           do jc=1,nc
```

```
              ic3 = ic
              if(ic3 == 2 ) ic3 = 3
              where( mask(:,:,:,:,ihat,imask) )
                 urprmsu3(:,:,:,:,ic3,jc) =
     &                      ( ursu2(:,:,:,:,ic,1) * ur(:,:,:,:,ihat,1,jc) +
     &                        ursu2(:,:,:,:,ic,2) * ur(:,:,:,:,ihat,3,jc) -
     &                        uisu2(:,:,:,:,ic,1) * ui(:,:,:,:,ihat,1,jc) -
     &                        uisu2(:,:,:,:,ic,2) * ui(:,:,:,:,ihat,3,jc) )
                 uiprmsu3(:,:,:,:,ic3,jc) =
     &                      ( ursu2(:,:,:,:,ic,1) * ui(:,:,:,:,ihat,1,jc) +
     &                        ursu2(:,:,:,:,ic,2) * ui(:,:,:,:,ihat,3,jc) +
     &                        uisu2(:,:,:,:,ic,1) * ur(:,:,:,:,ihat,1,jc) +
     &                        uisu2(:,:,:,:,ic,2) * ur(:,:,:,:,ihat,3,jc) )
              elsewhere
                 urprmsu3(:,:,:,:,ic3,jc) = ur(:,:,:,:,ihat,ic3,jc)
                 uiprmsu3(:,:,:,:,ic3,jc) = ui(:,:,:,:,ihat,ic3,jc)
              end where
           end do
        end do

        do jc=1,nc
           ur(:,:,:,:,ihat,1,jc) = urprmsu3(:,:,:,:,1,jc)
           ui(:,:,:,:,ihat,1,jc) = uiprmsu3(:,:,:,:,1,jc)
           ur(:,:,:,:,ihat,3,jc) = urprmsu3(:,:,:,:,3,jc)
           ui(:,:,:,:,ihat,3,jc) = uiprmsu3(:,:,:,:,3,jc)
        end do

        end do

     end do
  end do

  return
  end subroutine coolSweep
```

E.19 Cooling the Gauge Field Configurations at the $SU_c(2)$ Level

```
c
c     cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Author: Frederic D.R. Bonnet & D.B. Leinweber: date: February 1999.
c     initializes the link variables
c     subroutine that implements the pseudo-heatbath algorithm
c
      subroutine cooling(urnewsu2,uinewsu2,phbsr,phbsi)
      implicit none
      include 'latticeSize.h'

c     global variables
```

```
      integer,parameter                                     :: ncsu2=2

      double precision,dimension(nx,ny,nz,nt,ncsu2,ncsu2)     :: urnewsu2,uinewsu2
      double precision,dimension(nx,ny,nz,nt,ncsu2,ncsu2)     :: phbsr,phbsi
cmf$  layout urnewsu2(:news,:news,:news,:news,:serial,:serial)
cmf$  layout uinewsu2(:news,:news,:news,:news,:serial,:serial)
cmf$  layout phbsr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout phbsi(:news,:news,:news,:news,:serial,:serial)

c     local variables

      double precision,dimension(nx,ny,nz,nt)               :: k
cmf$  layout k(:news,:news,:news,:news)

      integer                                               :: ic,jc

c     start of the execution commands

c     calculates the determinant k=|\sum_{\alpha=1}^6\widetilde{U}_{\alpha}|^{1/2}
c     where $\widetilde{U}_{\alpha}\equiv$ the six product of the three links
c     variable which interact with the link in question, i.e. stapler and staplei

      k = sqrt( abs( phbsr(:,:,:,:,1,1) * phbsr(:,:,:,:,2,2) -
     &               phbsr(:,:,:,:,1,2) * phbsr(:,:,:,:,2,1) -
     &               phbsi(:,:,:,:,1,1) * phbsi(:,:,:,:,2,2) +
     &               phbsi(:,:,:,:,1,2) * phbsi(:,:,:,:,2,1) ) )

c     this calculates U --> U'= staple^{\dag} / k, the U (full link) coming out
c     of su2random is here replaced by U'(partial link, depends on ihat the direction)
c     this is for minimising the local action such that U*{\overline{U}}^{\dag}=I

      urnewsu2 = 0.0d0
      uinewsu2 = 0.0d0
      do ic=1,ncsu2
         do jc=1,ncsu2
                urnewsu2(:,:,:,:,ic,jc) =   phbsr(:,:,:,:,jc,ic) / k
                uinewsu2(:,:,:,:,ic,jc) = - phbsi(:,:,:,:,jc,ic) / k
         end do
      end do

      return
      end subroutine cooling
```

E.20 APE Smearing and AUS Smearing

```
c
c     cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Authors: F.D.R. Bonnet, D.B. Leinweber and M. Standford:  February 1999
c     Subroutine that simulates ape smearing taking the product of
c     links in the opposite
c     direction to 'staples'
```

```
c
      subroutine ape_smear(ur,ui,wparam,itype,nsub,uzero)
      implicit none
      include 'latticeSize.h'

c     global variables

      integer,parameter                                  :: nc=3 !sigma,color
      integer,parameter                                  :: mu=4 !direction

      integer                                                :: nsub
      integer                                                :: itype
      double precision                                       :: uzero
      double precision                       :: wparam

      double precision,dimension(nx,ny,nz,nt,mu,nc,nc)       :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)

c     local variables

      integer                                           :: xhat
      double precision                       :: g !g = (f)/(2*(mu-1))
      double precision                                       :: alpha  !alpha = (1-f)

      double precision,dimension(nx,ny,nz,nt,nc,nc)          ::stapler,staplei
cmf$  layout stapler(:news,:news,:news,:news,:serial,:serial)
cmf$  layout staplei(:news,:news,:news,:news,:serial,:serial)

      double precision,dimension(nx,ny,nz,nt,mu,nc,nc)       ::  urprm,uiprm
cmf$  layout urprm(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout uiprm(:news,:news,:news,:news,:serial,:serial,:serial)

      integer                                        :: ic,jc

      INTERFACE
         SUBROUTINE staples(ur,ui,stapler,staplei,xhat,local,itype,uzero)
          IMPLICIT NONE
          include 'latticeSize.h'
          integer,parameter                                  :: nc=3
          integer,parameter                                  :: mu=4
          integer                                            :: xhat
          integer                                            :: itype
          double precision                                   :: uzero
          logical                                            :: local
          double precision,dimension(nx,ny,nz,nt,nc,nc)      :: stapler,staplei
cmf$  layout stapler(:news,:news,:news,:news,:serial,:serial)
cmf$  layout staplei(:news,:news,:news,:news,:serial,:serial)
          double precision,dimension(nx,ny,nz,nt,mu,nc,nc)   :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)
         end SUBROUTINE staples
         subroutine fixsu3(ur,ui)
          implicit none
          include 'latticeSize.h'
          integer,parameter                            :: nc=3
          integer,parameter                            :: mu=4
          double precision,dimension(nx,ny,nz,nt,mu,nc,nc)    :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)
         end subroutine fixsu3
         subroutine Newfixsu3(ur,ui,urprm,uiprm,nsub)
          implicit none
          include 'latticeSize.h'
          integer,parameter                                   :: nc=3,ncsu2=2
          integer,parameter                                   :: mu=4
          integer                                             :: nsub
          double precision,dimension(nx,ny,nz,nt,mu,nc,nc)    ::  urprm,uiprm
cmf$  layout urprm(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout uiprm(:news,:news,:news,:news,:serial,:serial,:serial)
          double precision,dimension(nx,ny,nz,nt,mu,nc,nc)    :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)
         end subroutine Newfixsu3
      END INTERFACE

c     starting the execution commands

      g = ( wparam ) / ( 2 * (mu-1) )
      alpha = ( 1.0d0 - wparam )

      do xhat=1,mu
         call staples(ur,ui,stapler,staplei,xhat,.true.,itype,uzero)
         do ic=1,nc
    do jc=1,nc
               urprm(:,:,:,:,xhat,ic,jc) =
     &         alpha * ur(:,:,:,:,xhat,ic,jc) + g * stapler(:,:,:,:,jc,ic)
               uiprm(:,:,:,:,xhat,ic,jc) =
     &         alpha * ui(:,:,:,:,xhat,ic,jc) - g * staplei(:,:,:,:,jc,ic)
    end do
         end do
      end do

c      call fixsu3(urprm,uiprm)
c      ur = urprm
c      ui = uiprm

      call Newfixsu3(ur,ui,urprm,uiprm,nsub)

      return
      end subroutine ape_smear
c
c     cccccsssssssccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Authors: F.D.R. Bonnet, D.B. Leinweber and M. Standford:  February 1999
```

```
c     Subroutine that simulates ape smearing taking the product of links in the
c     opposite direction to 'staples'
c
      subroutine cool_smear(ur,ui,wparam,itype,uzero)
      implicit none
      include 'latticeSize.h'

c     global variables

      integer,parameter                                   :: nc=3
      integer,parameter                                   :: mu=4

      integer                                                 :: itype
      double precision                                        :: uzero
      double precision                                        :: wparam

      double precision,dimension(nx,ny,nz,nt,mu,nc,nc)        :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)

c     local variables

      integer                                            :: xhat
      double precision                                        :: g  !g = (f)/(2*(mu-1))
      double precision                                        :: alpha !alpha = (1-f)

      double precision,dimension(nx,ny,nz,nt,nc,nc)           ::stapler,staplei
cmf$  layout stapler(:news,:news,:news,:news,:serial,:serial)
cmf$  layout staplei(:news,:news,:news,:news,:serial,:serial)

      integer                                                 :: ic,jc

      INTERFACE
        SUBROUTINE staples(ur,ui,stapler,staplei,xhat,local,itype,uzero)
          IMPLICIT NONE
          include 'latticeSize.h'
          integer,parameter                                   :: nc=3
          integer,parameter                                   :: mu=4
          integer                                             :: xhat
          integer                                             :: itype
          double precision                                    :: uzero
          logical                                             :: local
          double precision,dimension(nx,ny,nz,nt,nc,nc)       :: stapler,staplei
cmf$  layout stapler(:news,:news,:news,:news,:serial,:serial)
cmf$  layout staplei(:news,:news,:news,:news,:serial,:serial)
          double precision,dimension(nx,ny,nz,nt,mu,nc,nc)    :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)
        end SUBROUTINE staples
        SUBROUTINE fixsu3mu(ur,ui,imu)
          IMPLICIT NONE
          include 'latticeSize.h'
          integer,parameter                                       :: nc=3
          integer,parameter                                       :: mu=4
          double precision,dimension(nx,ny,nz,nt,mu,nc,nc)        :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)
          integer                                                 :: imu
        end SUBROUTINE fixsu3mu
      END INTERFACE

c     starting of the execution commands

      g = ( wparam ) / ( 2 * (mu-1) )
      alpha = ( 1.0d0 - wparam )

      do xhat=1,mu
         call staples(ur,ui,stapler,staplei,xhat,.true.,itype,uzero)
         do ic=1,nc
    do jc=1,nc
            ur(:,:,:,:,xhat,ic,jc) =
     &      alpha * ur(:,:,:,:,xhat,ic,jc) + g * stapler(:,:,:,:,jc,ic)
            ui(:,:,:,:,xhat,ic,jc) =
     &      alpha * ui(:,:,:,:,xhat,ic,jc) - g * staplei(:,:,:,:,jc,ic)
    end do
         end do
         call fixsu3mu(ur,ui,xhat)
      end do

      return
      end subroutine cool_smear
```

E.21 Reunitarization Method Based on $max \mathrm{Re}\,\mathrm{Tr}(UU^{\dagger\prime})$

```
c
c     cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Authors: F.D.R. Bonnet, D.B. Leinweber:  Date 28/4/2000.
c     subroutine that fixes the su3 links, using a smearing and cooling
c     techniques.
c
      subroutine Newfixsu3(ur,ui,urprm,uiprm,nsub)
      implicit none
      include 'latticeSize.h'

c     global variables.

      integer,parameter                                       :: nc=3,ncsu2=2
      integer,parameter                                       :: mu=4

      integer                                                 :: nsub
      integer                                                 :: itype
      double precision                                        :: uzero
      double precision                     :: wparam
      double precision,dimension(nx,ny,nz,nt,mu,nc,nc)        ::  urprm,uiprm
cmf$  layout urprm(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout uiprm(:news,:news,:news,:news,:serial,:serial,:serial)
      double precision,dimension(nx,ny,nz,nt,mu,nc,nc)        :: ur,ui
cmf$  layout ur(:news,:news,:news,:news,:serial,:serial,:serial)
cmf$  layout ui(:news,:news,:news,:news,:serial,:serial,:serial)

c     local variables.

      double precision,dimension(nx,ny,nz,nt,nc,nc)           ::stapler,staplei
cmf$  layout stapler(:news,:news,:news,:news,:serial,:serial)
cmf$  layout staplei(:news,:news,:news,:news,:serial,:serial)

      double precision,dimension(nx,ny,nz,nt,nc,nc)           :: ltsr,ltsi
      double precision,dimension(nx,ny,nz,nt,nc,nc)           :: urprmsu3,uiprmsu3
cmf$  layout ltsr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout ltsi(:news,:news,:news,:news,:serial,:serial)
cmf$  layout urprmsu3(:news,:news,:news,:news,:serial,:serial)
cmf$  layout uiprmsu3(:news,:news,:news,:news,:serial,:serial)
      double precision,dimension(nx,ny,nz,nt,ncsu2,ncsu2)     :: phbsr,phbsi
      double precision,dimension(nx,ny,nz,nt,ncsu2,ncsu2)     :: ursu2,uisu2
cmf$  layout phbsr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout phbsi(:news,:news,:news,:news,:serial,:serial)
cmf$  layout ursu2(:news,:news,:news,:news,:serial,:serial)
cmf$  layout uisu2(:news,:news,:news,:news,:serial,:serial)

      integer                                            :: ihat,isub
      integer                                                 :: of1
      integer                                        :: ic,jc,kc,ic3

      INTERFACE
         subroutine cooling(urnewsu2,uinewsu2,phbsr,phbsi)
          implicit none
          include 'latticeSize.h'
          integer,parameter                                   :: ncsu2=2
          double precision,dimension(nx,ny,nz,nt,ncsu2,ncsu2) :: phbsr,phbsi
          double precision,dimension(nx,ny,nz,nt,ncsu2,ncsu2) :: urnewsu2,uinewsu2
cmf$  layout phbsr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout phbsi(:news,:news,:news,:news,:serial,:serial)
cmf$  layout urnewsu2(:news,:news,:news,:news,:serial,:serial)
cmf$  layout uinewsu2(:news,:news,:news,:news,:serial,:serial)
         end subroutine cooling
      END INTERFACE

c     starting the execution commands

c     first calculate the uprimes.

      do ihat=1,mu

c     now setting the urprm and uiprm dageer to the staples.
```

```
            do ic=1,nc
               do jc=1,nc
                  stapler(:,:,:,:,ic,jc) =   urprm(:,:,:,:,ihat,jc,ic)
                  staplei(:,:,:,:,ic,jc) = - uiprm(:,:,:,:,ihat,jc,ic)
               end do
            end do

c     now using the pseudosweep routine to cool the imformation at the
c     su(2) level.

c     here looping over the su(2) subgroups.

            do isub=1,nsub
c
c     case 1. a_1
c
c     The Wilson action can be written as S(U)=Sum_p(Re(U*U_p))+constant
c                                             =Re(Tr(U*R))+constant
c     R is just the sum over the six plaquette namely the staples: stapler,staplei.

               ltsr = 0.0d0
               ltsi = 0.0d0
               do ic=1,nc
                  do jc=1,nc
                     do kc=1,nc
                        ltsr(:,:,:,:,ic,jc) = ltsr(:,:,:,:,ic,jc) +
     &                     ( ur(:,:,:,:,ihat,ic,kc) * stapler(:,:,:,:,kc,jc) -
     &                       ui(:,:,:,:,ihat,ic,kc) * staplei(:,:,:,:,kc,jc) )
                        ltsi(:,:,:,:,ic,jc) = ltsi(:,:,:,:,ic,jc) +
     &                     ( ur(:,:,:,:,ihat,ic,kc) * staplei(:,:,:,:,kc,jc) +
     &                       ui(:,:,:,:,ihat,ic,kc) * stapler(:,:,:,:,kc,jc) )
                     end do
                  end do
               end do

               of1 = 0

               phbsr(:,:,:,:,1,1) =  ( ltsr(:,:,:,:,1+of1,1+of1) +
     &                                 ltsr(:,:,:,:,2+of1,2+of1) ) / 2.0d0
               phbsr(:,:,:,:,1,2) =  ( ltsr(:,:,:,:,1+of1,2+of1) -
     &                                 ltsr(:,:,:,:,2+of1,1+of1) ) / 2.0d0
               phbsr(:,:,:,:,2,1) = - phbsr(:,:,:,:,1,2)
               phbsr(:,:,:,:,2,2) =   phbsr(:,:,:,:,1,1)

               phbsi(:,:,:,:,1,1) = (  ltsi(:,:,:,:,1+of1,1+of1) -
     &                                 ltsi(:,:,:,:,2+of1,2+of1) ) / 2.0d0
               phbsi(:,:,:,:,1,2) = (  ltsi(:,:,:,:,1+of1,2+of1) +
     &                                 ltsi(:,:,:,:,2+of1,1+of1) ) / 2.0d0
               phbsi(:,:,:,:,2,1) =   phbsi(:,:,:,:,1,2)
               phbsi(:,:,:,:,2,2) = - phbsi(:,:,:,:,1,1)
```

```
         call cooling(ursu2,uisu2,phbsr,phbsi)

         do ic=1,nc-1
            do jc=1,nc
               urprmsu3(:,:,:,:,ic,jc) =
     &              ( ursu2(:,:,:,:,ic,1) * ur(:,:,:,:,ihat,1,jc) +
     &                ursu2(:,:,:,:,ic,2) * ur(:,:,:,:,ihat,2,jc) -
     &                uisu2(:,:,:,:,ic,1) * ui(:,:,:,:,ihat,1,jc) -
     &                uisu2(:,:,:,:,ic,2) * ui(:,:,:,:,ihat,2,jc) )
               uiprmsu3(:,:,:,:,ic,jc) =
     &              ( ursu2(:,:,:,:,ic,1) * ui(:,:,:,:,ihat,1,jc) +
     &                ursu2(:,:,:,:,ic,2) * ui(:,:,:,:,ihat,2,jc) +
     &                uisu2(:,:,:,:,ic,1) * ur(:,:,:,:,ihat,1,jc) +
     &                uisu2(:,:,:,:,ic,2) * ur(:,:,:,:,ihat,2,jc) )
            end do
         end do

         do jc=1,nc
            urprmsu3(:,:,:,:,3,jc) = ur(:,:,:,:,ihat,3,jc)
            uiprmsu3(:,:,:,:,3,jc) = ui(:,:,:,:,ihat,3,jc)
         end do
c
c     case 2. a_2
c
c     here we calculate the next bit of the product, link*staples.
c     We use the old staples to multiply it with Uprmsu3 = a_1 * U
c     lts = matrixa = [ [SU(2)] 0 ]*lts[ ele in SU(3)    0    ].
c                     [    0    1 ]    [      0          0    ]

         ltsr = 0.0d0
         ltsi = 0.0d0
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  ltsr(:,:,:,:,ic,jc) = ltsr(:,:,:,:,ic,jc) +
     &                ( urprmsu3(:,:,:,:,ic,kc) * stapler(:,:,:,:,kc,jc) -
     &                  uiprmsu3(:,:,:,:,ic,kc) * staplei(:,:,:,:,kc,jc) )
                  ltsi(:,:,:,:,ic,jc) = ltsi(:,:,:,:,ic,jc) +
     &                ( urprmsu3(:,:,:,:,ic,kc) * staplei(:,:,:,:,kc,jc) +
     &                  uiprmsu3(:,:,:,:,ic,kc) * stapler(:,:,:,:,kc,jc) )
               end do
            end do
         end do

         of1 = 1

         phbsr(:,:,:,:,1,1) = (  ltsr(:,:,:,:,1+of1,1+of1) +
     &                           ltsr(:,:,:,:,2+of1,2+of1) ) / 2.0d0
         phbsr(:,:,:,:,1,2) = (  ltsr(:,:,:,:,1+of1,2+of1) -
     &                           ltsr(:,:,:,:,2+of1,1+of1) ) / 2.0d0
         phbsr(:,:,:,:,2,1) = - phbsr(:,:,:,:,1,2)
         phbsr(:,:,:,:,2,2) =   phbsr(:,:,:,:,1,1)
         phbsi(:,:,:,:,1,1) = (  ltsi(:,:,:,:,1+of1,1+of1) -
     &                           ltsi(:,:,:,:,2+of1,2+of1) ) / 2.0d0
         phbsi(:,:,:,:,1,2) = (  ltsi(:,:,:,:,1+of1,2+of1) +
     &                           ltsi(:,:,:,:,2+of1,1+of1) ) / 2.0d0
         phbsi(:,:,:,:,2,1) =   phbsi(:,:,:,:,1,2)
         phbsi(:,:,:,:,2,2) = - phbsi(:,:,:,:,1,1)

         call cooling(ursu2,uisu2,phbsr,phbsi)

         do ic=2,nc
            do jc=1,nc
               ur(:,:,:,:,ihat,ic,jc) =
     &              ( ursu2(:,:,:,:,ic-1,1) * urprmsu3(:,:,:,:,2,jc) +
     &                ursu2(:,:,:,:,ic-1,2) * urprmsu3(:,:,:,:,3,jc) -
     &                uisu2(:,:,:,:,ic-1,1) * uiprmsu3(:,:,:,:,2,jc) -
     &                uisu2(:,:,:,:,ic-1,2) * uiprmsu3(:,:,:,:,3,jc) )
               ui(:,:,:,:,ihat,ic,jc) =
     &              ( ursu2(:,:,:,:,ic-1,1) * uiprmsu3(:,:,:,:,2,jc) +
     &                ursu2(:,:,:,:,ic-1,2) * uiprmsu3(:,:,:,:,3,jc) +
     &                uisu2(:,:,:,:,ic-1,1) * urprmsu3(:,:,:,:,2,jc) +
     &                uisu2(:,:,:,:,ic-1,2) * urprmsu3(:,:,:,:,3,jc) )
            end do
         end do

         do jc=1,nc
            ur(:,:,:,:,ihat,1,jc) = urprmsu3(:,:,:,:,1,jc)
            ui(:,:,:,:,ihat,1,jc) = uiprmsu3(:,:,:,:,1,jc)
         end do
c
c     case 3. a_3
c
c     forming another SU(2) subgroup of the form
c     These two matrices have the form matrixa = [ x  0   x ]
c                                                [ 0  1   0 ]
c                                                [ x  0   x ]
         ltsr = 0.0d0
         ltsi = 0.0d0
         do ic=1,nc
            do jc=1,nc
               do kc=1,nc
                  ltsr(:,:,:,:,ic,jc) = ltsr(:,:,:,:,ic,jc) +
     &                ( ur(:,:,:,:,ihat,ic,kc) * stapler(:,:,:,:,kc,jc) -
     &                  ui(:,:,:,:,ihat,ic,kc) * staplei(:,:,:,:,kc,jc) )
                  ltsi(:,:,:,:,ic,jc) = ltsi(:,:,:,:,ic,jc) +
     &                ( ur(:,:,:,:,ihat,ic,kc) * staplei(:,:,:,:,kc,jc) +
     &                  ui(:,:,:,:,ihat,ic,kc) * stapler(:,:,:,:,kc,jc) )
               end do
            end do
         end do
```

```
            phbsr(:,:,:,:,1,1) = (  ltsr(:,:,:,:,1,1) + ltsr(:,:,:,:,3,3) ) / 2.0d0
            phbsr(:,:,:,:,1,2) = (  ltsr(:,:,:,:,1,3) - ltsr(:,:,:,:,3,1) ) / 2.0d0
            phbsr(:,:,:,:,2,1) = - phbsr(:,:,:,:,1,2)
            phbsr(:,:,:,:,2,2) =   phbsr(:,:,:,:,1,1)

            phbsi(:,:,:,:,1,1) = ( ltsi(:,:,:,:,1,1) - ltsi(:,:,:,:,3,3) ) / 2.0d0
            phbsi(:,:,:,:,1,2) = ( ltsi(:,:,:,:,1,3) + ltsi(:,:,:,:,3,1) ) / 2.0d0
            phbsi(:,:,:,:,2,1) =   phbsi(:,:,:,:,1,2)
            phbsi(:,:,:,:,2,2) = - phbsi(:,:,:,:,1,1)

            call cooling(ursu2,uisu2,phbsr,phbsi)

c      We next make a product of the matrices in such a way that
c      link Uprmsu3 = urprmsu3+iuiprmsu3 = ( ar + iai ) * ( ur + iui )
c                                 = ( ar*ur - ai*ui ) + i( ar*ui + ai*ur)
c      by hotwiring the matrix indices for optimization.

            do ic=1,nc-1
               do jc=1,nc
                  ic3 = ic
                  if(ic3 == 2 ) ic3 = 3
                  urprmsu3(:,:,:,:,ic3,jc) =
     &                 ( ursu2(:,:,:,:,ic,1) * ur(:,:,:,:,ihat,1,jc) +
     &                   ursu2(:,:,:,:,ic,2) * ur(:,:,:,:,ihat,3,jc) -
     &                   uisu2(:,:,:,:,ic,1) * ui(:,:,:,:,ihat,1,jc) -
     &                   uisu2(:,:,:,:,ic,2) * ui(:,:,:,:,ihat,3,jc) )
                  uiprmsu3(:,:,:,:,ic3,jc) =
     &                 ( ursu2(:,:,:,:,ic,1) * ui(:,:,:,:,ihat,1,jc) +
     &                   ursu2(:,:,:,:,ic,2) * ui(:,:,:,:,ihat,3,jc) +
     &                   uisu2(:,:,:,:,ic,1) * ur(:,:,:,:,ihat,1,jc) +
     &                   uisu2(:,:,:,:,ic,2) * ur(:,:,:,:,ihat,3,jc) )
               end do
            end do

            do jc=1,nc
               ur(:,:,:,:,ihat,1,jc) = urprmsu3(:,:,:,:,1,jc)
               ui(:,:,:,:,ihat,1,jc) = uiprmsu3(:,:,:,:,1,jc)
               ur(:,:,:,:,ihat,3,jc) = urprmsu3(:,:,:,:,3,jc)
               ui(:,:,:,:,ihat,3,jc) = uiprmsu3(:,:,:,:,3,jc)
            end do

         end do

      end do

      return
      end subroutine Newfixsu3
```

E.22 The Reunitarization of the Gauge Transformation

```
c
c     ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Author: Frederic D.R. Bonnet; date: 29th of June 1998.
c     subroutine that fixes the gauge links. This subroutine needs to
c     be called after a the transformation the
c     purpose being to keep the links within the SU(3) algebra. This
c     routine reconstruct the su3 element by doing the row by row
c     orthonormailzation method. Unlike fixsu3 the direction mu is not
c     taken into consideration.
c
      subroutine fixsgauge(tgr,tgi)
      implicit none
      include 'latticeSize.h'
      include 'LatParamtrans.h'

c     global variables

      double precision,dimension(nx,ny,nz,nt,nc,nc)          :: tgr,tgi
cmf$  layout tgr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout tgi(:news,:news,:news,:news,:serial,:serial)

c     local variables

      integer,dimension(5)                                    :: yvector
cmf$  layout yvector(:serial)
      double precision,dimension(nx,ny,nz,nt)                 :: normr,normi
cmf$  layout normr(:news,:news,:news,:news)
cmf$  layout normi(:news,:news,:news,:news)
      integer                                                 :: jc,imu

c
c     First create an array to be looped over below
c     y(1) = 1 , y(2) = 2 , y(3) = 3 , y(4) = 1 , y(5) = 2

      do jc=1,nc
         yvector(jc) = jc
      end do
      do jc=4,5
         yvector(jc) = jc - 3
      end do

c     We'll do a loop here to save memory demands


c
c     first normalise first row
c
      normr = sqrt( tgr(:,:,:,:,1,1)**2 + tgr(:,:,:,:,1,2)**2 +
     &              tgi(:,:,:,:,1,1)**2 + tgi(:,:,:,:,1,2)**2 +
     &              tgr(:,:,:,:,1,3)**2 + tgi(:,:,:,:,1,3)**2 )

      do jc=1,nc
         tgr(:,:,:,:,1,jc) = tgr(:,:,:,:,1,jc) / normr
         tgi(:,:,:,:,1,jc) = tgi(:,:,:,:,1,jc) / normr
      end do
c
c     now compute row2 - (row2 dot row1)*row1
c
      normr = tgr(:,:,:,:,2,1) * tgr(:,:,:,:,1,1) +
     &        tgi(:,:,:,:,2,1) * tgi(:,:,:,:,1,1) +
     &        tgr(:,:,:,:,2,2) * tgr(:,:,:,:,1,2) +
     &        tgi(:,:,:,:,2,2) * tgi(:,:,:,:,1,2) +
     &        tgr(:,:,:,:,2,3) * tgr(:,:,:,:,1,3) +
     &        tgi(:,:,:,:,2,3) * tgi(:,:,:,:,1,3)

      normi = tgi(:,:,:,:,2,1) * tgr(:,:,:,:,1,1) -
     &        tgr(:,:,:,:,2,1) * tgi(:,:,:,:,1,1) +
     &        tgi(:,:,:,:,2,2) * tgr(:,:,:,:,1,2) -
     &        tgr(:,:,:,:,2,2) * tgi(:,:,:,:,1,2) +
     &        tgi(:,:,:,:,2,3) * tgr(:,:,:,:,1,3) -
     &        tgr(:,:,:,:,2,3) * tgi(:,:,:,:,1,3)

      do jc=1,nc
         tgr(:,:,:,:,2,jc) = tgr(:,:,:,:,2,jc) -
     &                       (  normr * tgr(:,:,:,:,1,jc) - normi * tgi(:,:,:,:,1,jc) )
         tgi(:,:,:,:,2,jc) = tgi(:,:,:,:,2,jc) -
     &                       (  normr * tgi(:,:,:,:,1,jc) + normi * tgr(:,:,:,:,1,jc) )
      end do

c     Now normalise the second row

      normr = sqrt( tgr(:,:,:,:,2,1)**2 + tgi(:,:,:,:,2,1)**2 +
     &              tgr(:,:,:,:,2,2)**2 + tgi(:,:,:,:,2,2)**2 +
     &              tgr(:,:,:,:,2,3)**2 + tgi(:,:,:,:,2,3)**2 )

      do jc=1,nc
         tgr(:,:,:,:,2,jc) = tgr(:,:,:,:,2,jc) / normr
         tgi(:,:,:,:,2,jc) = tgi(:,:,:,:,2,jc) / normr
      end do

c     now generate row3 = row1 cross row2

      do jc=1,nc
         tgr(:,:,:,:,3,jc) =
     &       tgr(:,:,:,:,1,yvector(jc+1)) * tgr(:,:,:,:,2,yvector(jc+2)) -
     &       tgi(:,:,:,:,1,yvector(jc+1)) * tgi(:,:,:,:,2,yvector(jc+2)) -
     &       tgr(:,:,:,:,1,yvector(jc+2)) * tgr(:,:,:,:,2,yvector(jc+1)) +
     &       tgi(:,:,:,:,1,yvector(jc+2)) * tgi(:,:,:,:,2,yvector(jc+1))
         tgi(:,:,:,:,3,jc) =
     &     - tgr(:,:,:,:,1,yvector(jc+1)) * tgi(:,:,:,:,2,yvector(jc+2)) -
     &       tgi(:,:,:,:,1,yvector(jc+1)) * tgr(:,:,:,:,2,yvector(jc+2)) +
     &       tgr(:,:,:,:,1,yvector(jc+2)) * tgi(:,:,:,:,2,yvector(jc+1)) +
     &       tgi(:,:,:,:,1,yvector(jc+2)) * tgr(:,:,:,:,2,yvector(jc+1))
      end do

      return
      end subroutine fixsgauge
```

E.23 Program to calculate the $\mathcal{C}_\mu(p)$ and $\mathcal{B}(p)$.

```
c     program to first read in gauge field configuration and quark
c     propagators. The program will then calcultate the color trace
c     fourier transform the data and (optional) will calculate the invert
c     of the 4x4 matrix in the Dirac space. Then calculate the full color trace
c     multiply by an arbitrary gamma matrix the inverted propagator to give the
c     Z(p) function.
c     ------------------------------------------------------------------------------
c         Quark-gluon vertex utility for SU(3) Gauge configuration
c     ------------------------------------------------------------------------------
c     Author: Frederic D.R. Bonnet; date: 14 of April 2000.
c                                         22 of November 2000.
c                                            Included the Overlap fermion routines
c                                            and front end ReadZlinks,ReadZprop
c                                         4th June 2001.
c                                             Include the write out on a configuration
c                                             basis of the curli B and curli C_\mu
c     output files:
c     are propagator.
c     Transform.log.
c
c     To compile
c
c     f95 -fast -extend_source -convert big_endian
c              -assume byterecl overlap.f Newroutine.f transform.f -o
c     f95 -fast -extend_source overlap.f Newroutine.f Transform.f -o
c                                          non overlap fermions.
c     cmf -cm5 -vu -extend_source -f90syntax Transform.fcm -o outputfile
c     ------------------------------------------------------------------------------
      program Trans_Jack
      implicit none
      include 'latticeSize.h'
      include 'LatParamtrans.h'

c     global variables.

      integer,parameter           :: nmx=-nx/2+1,npx=nx/2 !momentum boundaries in x
      integer,parameter           :: nmy=-ny/2+1,npy=ny/2 !momentum boundaries in y
      integer,parameter           :: nmz=-nz/2+1,npz=nz/2 !momentum boundaries in z
      integer,parameter           :: nmt=-nt/2+1,npt=nt/2 !momentum boundaries in t

      double precision                  :: bcx,bcy,bcz,bct      !boundary conditions
```

```
c     local variables.

      integer,parameter                                       :: nreal=2
      integer,parameter                                       :: ngamma=3

      logical                                                 :: uexists=.true.

      integer                                                 :: nkappaQP
      integer                                                 :: jx,jy,jz,jt !source
      integer                                                 :: gfix !gfixed or not
      integer                                                 :: flq  !flink or not
      character(len=4)                                        :: fixname
      character(len=4)                                        :: cfg
      character(len=80)                                       :: quarkprop
      character(len=80)                                       :: lastconfig
      character(len=80)                                       :: corename,trivconf
      character(len=100)                                      :: datafile
      character(len=100),dimension(10)                        :: dataf
      character(len=5),dimension(nkappa)                      :: qpk
      character(len=100),dimension(nkappa)                    :: qp
cmf$  layout dataf(:serial)
cmf$  layout qpk(:serial)
cmf$  layout qp(:serial)

      double precision,dimension(nkappa)                      :: xkappa  !kappa values
cmf$  layout xkappa(:serial)

      double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt)
     &                                                        :: Bcurlr,Bcurli
cmf$  layout Bcurlr(:news,:news,:news,:news)
cmf$  layout Bcurli(:news,:news,:news,:news)
      double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt,mu)
     &                                                        :: Ccurlmur,Ccurlmui
cmf$  layout Ccurlmur(:news,:news,:news,:news,:serial)
cmf$  layout Ccurlmui(:news,:news,:news,:news,:serial)
      double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt)
     &                                                        :: pmrSCT,pmiSCT
cmf$  layout pmrSCT(:news,:news,:news,:news)
cmf$  layout pmiSCT(:news,:news,:news,:news)
      double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt,nd,nd)
     &                                                        :: pmrCT,pmiCT
cmf$  layout pmrCT(:news,:news,:news,:news,:serial,:serial)
cmf$  layout pmiCT(:news,:news,:news,:news,:serial,:serial)

      double precision,dimension(nx,ny,nz,nt,nd,nd)           :: prCT,piCT
cmf$  layout prCT(:news,:news,:news,:news,:serial,:serial)
cmf$  layout piCT(:news,:news,:news,:news,:serial,:serial)
      double precision,dimension(nx*ny*nz,nt,nc,nd,nc,nd)     :: pr,pi
cmf$  layout pr(:news,:news,:serial,:serial,:serial,:serial)
cmf$  layout pi(:news,:news,:serial,:serial,:serial,:serial)
      double precision,dimension(nx*ny*nz,nt,nc,nd,nc,nd)     :: pgr,pgi
```

```
cmf$  layout pgr(:news,:news,:serial,:serial,:serial,:serial)
cmf$  layout pgi(:news,:news,:serial,:serial,:serial,:serial)


c     overlap stuff

      double precision,dimension(nt)                         :: pionm,pionpr,pionpi
cmf$  layout pionm(:serial)
cmf$  layout pionpr(:serial)
cmf$  layout pionpi(:serial)
      double precision,dimension(nd,nd)                      :: deltafunc,gam5
cmf$  layout deltafunc(:serial,:serial)
cmf$  layout gam5(:serial,:serial)
      double precision,dimension(nd,nd,ngamma,nreal)         :: gammai,gamma5i
cmf$  layout  gammai(:serial,:serial,:serial,:serial)
cmf$  layout gamma5i(:serial,:serial,:serial,:serial)

      integer                                                :: ikappa,iprop
      integer                                                :: imu
      integer                                                :: id,jd,ic
      integer                                                :: ix,iy,iz,it
      integer                                                :: counter

      INTERFACE
         subroutine ReadProp(qp,pr,pi,flq,xkappa,ikappa,jx,jy,jz,jt)
          implicit none
          include 'latticeSize.h'
          include 'LatParamtrans.h'
          integer,parameter                               :: nmx=-nx/2+1,npx=nx/2
          integer,parameter                               :: nmy=-ny/2+1,npy=ny/2
          integer,parameter                               :: nmz=-nz/2+1,npz=nz/2
          integer,parameter                               :: nmt=-nt/2+1,npt=nt/2
          integer                                            :: ikappa
          integer                                            :: flq
          integer                                            :: jx,jy,jz,jt
          character(len=100),dimension(nkappa)               :: qp
          double precision,dimension(nkappa)                 :: xkappa
cmf$  layout xkappa(:serial)
          double precision,dimension(nx*ny*nz,nt,nc,nd,nc,nd)   :: pr,pi
cmf$  layout pr(:news,:news,:serial,:serial,:serial,:serial)
cmf$  layout pi(:news,:news,:serial,:serial,:serial,:serial)
         end subroutine ReadProp
         subroutine spintrace(pmrSCT,pmiSCT,pmrCT,pmiCT)
          implicit none
          include 'latticeSize.h'
          include 'LatParamtrans.h'
          integer,parameter                                  :: nmx=-nx/2+1,npx=nx/2
          integer,parameter                                  :: nmy=-ny/2+1,npy=ny/2
          integer,parameter                                  :: nmz=-nz/2+1,npz=nz/2
          integer,parameter                                  :: nmt=-nt/2+1,npt=nt/2
          double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt)
     &                                                       :: pmrSCT,pmiSCT
```

```
cmf$  layout pmrSCT(:news,:news,:news,:news)
cmf$  layout pmiSCT(:news,:news,:news,:news)
          double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt,nd,nd)
     &                                                        :: pmrCT,pmiCT
cmf$  layout pmrCT(:news,:news,:news,:news,:serial,:serial)
cmf$  layout pmiCT(:news,:news,:news,:news,:serial,:serial)
         end subroutine spintrace
         subroutine curlCmu(Ccurlmur,Ccurlmui,pmrCT,pmiCT)
          implicit none
          include 'latticeSize.h'
          include 'LatParamtrans.h'
          integer,parameter                                  :: nmx=-nx/2+1,npx=nx/2
          integer,parameter                                  :: nmy=-ny/2+1,npy=ny/2
          integer,parameter                                  :: nmz=-nz/2+1,npz=nz/2
          integer,parameter                                  :: nmt=-nt/2+1,npt=nt/2
        double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt,mu)
     &                                                        :: Ccurlmur,Ccurlmui
cmf$  layout Ccurlmur(:news,:news,:news,:news,:serial)
cmf$  layout Ccurlmui(:news,:news,:news,:news,:serial)
          double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt,nd,nd)
     &                                                        :: pmrCT,pmiCT
cmf$  layout pmrCT(:news,:news,:news,:news,:serial,:serial)
cmf$  layout pmiCT(:news,:news,:news,:news,:serial,:serial)
         end subroutine curlCmu
         subroutine CurliCandBout(Bcurlr,Bcurli,Ccurlmur,Ccurlmui,qpk,xkappa,
     &                          dataf,ikappa,bcx,bcy,bcz,bct)
          implicit none
          include 'latticeSize.h'
          include 'LatParamtrans.h'
          integer,parameter                                  :: nmx=-nx/2+1,npx=nx/2
          integer,parameter                                  :: nmy=-ny/2+1,npy=ny/2
          integer,parameter                                  :: nmz=-nz/2+1,npz=nz/2
          integer,parameter                                  :: nmt=-nt/2+1,npt=nt/2
          integer                                            :: ikappa
          double precision                                   :: bcx,bcy,bcz,bct
          double precision,dimension(nkappa)                 :: xkappa
cmf$  layout xkappa(:serial)
          character(len=100),dimension(10)                   :: dataf
cmf$  layout dataf(:serial)
          character(len=5),dimension(nkappa)                 :: qpk
cmf$  layout qpk(:serial)
          double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt):: Bcurlr,Bcurli
cmf$  layout Bcurlr(:news,:news,:news,:news)
cmf$  layout Bcurli(:news,:news,:news,:news)
      double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt,mu)
     &                                                        :: Ccurlmur,Ccurlmui
cmf$  layout Ccurlmur(:news,:news,:news,:news,:serial)
cmf$  layout Ccurlmui(:news,:news,:news,:news,:serial)
         end subroutine CurliCandBout
         subroutine pion(pr,pi,pionm,pionpr,pionpi,deltafunc,gam5,gammai,gamma5i,it)
          implicit none
          include 'latticeSize.h'
```

```
            include 'LatParamtrans.h'
            integer                                               :: it
            integer,parameter                                     :: nreal=2
            integer,parameter                                     :: ngamma=3
            double precision,dimension(nt)                        :: pionm,pionpr,pionpi
cmf$  layout pionm(:serial)
cmf$  layout pionpr(:serial)
cmf$  layout pionpi(:serial)
            double precision,dimension(nd,nd)                     :: deltafunc,gam5
cmf$  layout deltafunc(:serial,:serial)
cmf$  layout gam5(:serial,:serial)
            double precision,dimension(nd,nd,ngamma,nreal)        :: gammai,gamma5i
cmf$  layout  gammai(:serial,:serial,:serial,:serial)
cmf$  layout gamma5i(:serial,:serial,:serial,:serial)
            double precision,dimension(nx*ny*nz,nt,nc,nd,nc,nd)   :: pr,pi
cmf$  layout pr(:news,:news,:serial,:serial,:serial,:serial)
cmf$  layout pi(:news,:news,:serial,:serial,:serial,:serial)
           end subroutine pion
           subroutine deltaNDgam(deltafunc,gam5,gammai,gamma5i)
            implicit none
            integer,parameter                                     :: nd=4 !direction
            integer,parameter                                     :: nreal=2
            integer,parameter                                     :: ngamma=3
            double precision,dimension(nd,nd)                     :: deltafunc,gam5
cmf$  layout deltafunc(:serial,:serial)
cmf$  layout gam5(:serial,:serial)
            double precision,dimension(nd,nd,ngamma,nreal)        :: gammai,gamma5i
cmf$  layout  gammai(:serial,:serial,:serial,:serial)
cmf$  layout gamma5i(:serial,:serial,:serial,:serial)
           end subroutine deltaNDgam
           function strlen(string)
            implicit none
            character*(*) string
            integer                                             :: strlen
            integer                                             :: i, blank
           end function strlen
      end interface

c     start of the execution commands.

c     first read in the datafile ParamTrans.dat

      open(11,file='ParamTrans_CurliCB.dat',form='formatted',
     &              status='old',action='read')

      read (11,*) nkappaQP

      if (nkappaQP /= nkappa) pause 'mismatch in nkappa'
      do ikappa=1,nkappa
         read (11,*) xkappa(ikappa)
         read (11,'(a5)') qpk(ikappa)
      end do
      read (11,*) jx,jy,jz,jt
      read (11,*) bcx,bcy,bcz,bct
      close(11)

c     first start by sreen input.

      write(*,*)'Would you to examine fat link action:'
      write(*,*)'                         0:no  fat link'
      write(*,*)'                         1:yes fat link.'
      write(*,*)'                         2:overlap Fermions.'

      read(*,'(i3)') flq
      write(*,*)

       write(*,*) flq

      write(*,*)'Would you like to deal with gauge fixed propagators, 0=no,1=yes.'
      read(*,'(i3)') gfix
      write(*,*)

       write(*,*) gfix

      write(*,*)'Please enter the core name of the configuration:'
      write(*,*)'Ex: SU3B600S6T18C'
      read(*,'(a80)') corename
      write(*,*)

       write(*,'(a)') corename

      write(*,*)'Please enter the fix name name of the configuration:'
      write(*,*)'Ex: if 0.078 then enter .078. If no fixname leave blank'
      read (*,'(a4)') fixname
      write(*,*)

         write(*,'(a)') fixname

      write(*,*)
      write(*,*)'Which Gauge field configuration are we reading from?'
      write(*,*)'With its associated quark propagator.'
      read (*,'(a3)') cfg

         write(*,'(a)') cfg

      lastconfig = corename(1:strlen(corename))//cfg
      lastconfig = lastconfig(1:strlen(lastconfig))//fixname

        write(*,*) lastconfig

      open(1,file='CurliCmuandB.log',status='unknown',position='append')

      write(1,*)
      write(1,*)'=====================================================
     &          ========================'
      write(1,'(a)')'    The mass function for the Wilson Fermions for
     &                  SU(3) Gauge configuration    '
      write(1,*)'=====================================================
     &          ========================'

      write(1,'(4(a,i3))')'lattice size = ',nx,'x',ny,'x',nz,'x',nt
      write(1,'(a,i2)')'The number of **kappa** value that are to be
     &                 considered is =',nkappa
      write(1,'(a,4i3)')  'The source position x, y, z, t is =',jx,jy,jz,jt
      write(1,'(a,4f8.4)')'The bndary condition for
     &                    fermion fields are=',bcx,bcy,bcz,bct

c     constructing the quark propagator filename.

      do ikappa = 1,nkappa

c     get the tree level curli B.

         if ( gfix == 0 ) then
            qp(ikappa) = lastconfig(1:strlen(lastconfig))//'.'//qpk(ikappa)
            write(*,*)
            write(*,'(a,3x,a100)')'We are now reading from:',qp(ikappa)
            write(1,'(a,3x,a100)')'We are now reading from:',qp(ikappa)
            call ReadProp(qp,pr,pi,flq,xkappa,ikappa,jx,jy,jz,jt)
            write(*,'(a,3x,a100)')'We have read from:',qp(ikappa)
            write(1,'(a,3x,a100)')'We have read from:',qp(ikappa)
            write(*,'(a,3x,a100)')'We are now gauge rotating to Landau gauge:',qp(ikappa)
            write(*,'(a,3x,a100)')'using the gauge transformed :',lastconfig//'.gag'
            call GTransSofx(lastconfig,pr,pi,pgr,pgi,jx,jy,jz,jt)
            write(*,'(a,3x,a100)')'We have gauge rotated to Landau gauge:',qp(ikappa)
         elseif( gfix == 1 ) then
            qp(ikappa) = lastconfig(1:strlen(lastconfig))//'.fix'//qpk(ikappa)
            write(*,*)
            write(1,*)
            write(*,'(a,3x,a100)')'We are now reading from:',qp(ikappa)
            write(1,'(a,3x,a100)')'We are now reading from:',qp(ikappa)
            call ReadProp(qp,pr,pi,flq,xkappa,ikappa,jx,jy,jz,jt)
            write(*,'(a,3x,a100)')'We have read from:',qp(ikappa)
            write(1,'(a,3x,a100)')'We have read from:',qp(ikappa)
         end if

c     calculating the color Trace of the propagators.
c     And reforming the propagators into more manageable arrays.
c     first refreshing the variables by reopening the the file name.

         prCT = 0.0d0
         piCT = 0.0d0

         counter = 0
         do iz=1,nz
```

```
            do iy=1,ny
               do ix=1,nx
                  counter = counter + 1
                  do ic=1,nc
             prCT(ix,iy,iz,:,:,:) = prCT(ix,iy,iz,:,:,:) + pgr(counter,:,ic,:,ic,:)
             piCT(ix,iy,iz,:,:,:) = piCT(ix,iy,iz,:,:,:) + pgi(counter,:,ic,:,ic,:)
                  end do
               end do
            end do
         end do

         prCT = (1.0d0/3.0d0) * prCT
         piCT = (1.0d0/3.0d0) * piCT

         call fourier(prCT,piCT,pmrCT,pmiCT,jx,jy,jz,jt)

c     calculating the trace of Tr{S(p)}, for both the full n-point function p
c     and the psi g.

         call spintrace(Bcurlr,Bcurli,pmrCT,pmiCT)

         if ( flq == 0 .or. flq == 1 ) then
            pmrCT  = ( 2.0d0 * xkappa(ikappa) ) * pmrCT
            pmiCT  = ( 2.0d0 * xkappa(ikappa) ) * pmiCT
            Bcurlr = ( 2.0d0 * xkappa(ikappa) ) * Bcurlr
            Bcurli = ( 2.0d0 * xkappa(ikappa) ) * Bcurli
         end if

c     get the tree level curli C.

         call curlCmu(Ccurlmur,Ccurlmui,pmrCT,pmiCT)

c     now dump the tree level culi C and B to disk

         datafile = 'Bcurl'
         datafile = datafile(1:strlen(datafile))//corename
         datafile = datafile(1:strlen(datafile))//cfg
         dataf(1) = datafile(1:strlen(datafile))//fixname

         datafile = 'Ccurl'
         datafile = datafile(1:strlen(datafile))//corename
         datafile = datafile(1:strlen(datafile))//cfg
         dataf(2) = datafile(1:strlen(datafile))//fixname

         call CurliCandBout(Bcurlr,Bcurli,Ccurlmur,Ccurlmui,qpk,
     &                      xkappa,dataf,ikappa,bcx,bcy,bcz,bct)

      end do

      close(1)

      end program Trans_Jack
```

```
c
c     ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Author: Frederic D.R. Bonnet; date: 4th of June 2001.
c     subroutine to calculate the uncorrected curli C_\mu(p) for the Jacknife.
c     analysis.
c
      subroutine curlCmu(Ccurlmur,Ccurlmui,pmrCT,pmiCT)
      implicit none
      include 'latticeSize.h'
      include 'LatParamtrans.h'

c     global variables.

      integer,parameter                                         :: nmx=-nx/2+1,npx=nx/2
      integer,parameter                                         :: nmy=-ny/2+1,npy=ny/2
      integer,parameter                                         :: nmz=-nz/2+1,npz=nz/2
      integer,parameter                                         :: nmt=-nt/2+1,npt=nt/2

      double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt,mu)
     &                                                          :: Ccurlmur,Ccurlmui
cmf$  layout Ccurlmur(:news,:news,:news,:news,:serial)
cmf$  layout Ccurlmui(:news,:news,:news,:news,:serial)
      double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt,nd,nd):: pmrCT,pmiCT
cmf$  layout pmrCT(:news,:news,:news,:news,:serial,:serial)
cmf$  layout pmiCT(:news,:news,:news,:news,:serial,:serial)

c     local variables.

      double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt,nd,nd)
     &                                                          :: pmrCTG,pmiCTG
cmf$  layout pmrCTG(:news,:news,:news,:news,:serial,:serial)
cmf$  layout pmiCTG(:news,:news,:news,:news,:serial,:serial)

      double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt,nd,nd)
     &                                                          :: pmrCTGk,pmiCTGk
cmf$  layout pmrCTGk(:news,:news,:news,:news,:serial,:serial)
cmf$  layout pmiCTGk(:news,:news,:news,:news,:serial,:serial)

      integer                                                   :: id,imu

c     start of the execution commands.

      interface
       subroutine gammaXSp(pmrCTG,pmiCTG,pmrCT,pmiCT,imu)
        implicit none
        include 'latticeSize.h'
        include 'LatParamtrans.h'
        integer,parameter                                       :: nmx=-nx/2+1,npx=nx/2
        integer,parameter                                       :: nmy=-ny/2+1,npy=ny/2
        integer,parameter                                       :: nmz=-nz/2+1,npz=nz/2
        integer,parameter                                       :: nmt=-nt/2+1,npt=nt/2
        integer                                                 :: imu
        double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt,nd,nd)
     &                                                       :: pmrCT,pmiCT
cmf$  layout pmrCT(:news,:news,:news,:news,:serial,:serial)
cmf$  layout pmiCT(:news,:news,:news,:news,:serial,:serial)
        double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt,nd,nd)
     &                                                       :: pmrCTG,pmiCTG
cmf$  layout pmrCTG(:news,:news,:news,:news,:serial,:serial)
cmf$  layout pmiCTG(:news,:news,:news,:news,:serial,:serial)
       end subroutine gammaXSp
      end interface

c     start of the execution commands.

c     first calculating the CurcC_{\mu}=(i/4)Tr[\gamma_\mu*S(p)]

      Ccurlmur = 0.0d0
      Ccurlmui = 0.0d0

      do imu=1,mu
         call gammaXSp(pmrCTG,pmiCTG,pmrCT,pmiCT,imu)

         pmrCTGk = - pmiCTG
         pmiCTGk =   pmrCTG

         do id=1,nd
            Ccurlmur(:,:,:,:,imu) = Ccurlmur(:,:,:,:,imu) + pmrCTGk(:,:,:,:,id,id)
            Ccurlmui(:,:,:,:,imu) = Ccurlmui(:,:,:,:,imu) + pmiCTGk(:,:,:,:,id,id)
         end do

      end do

      Ccurlmur = (1.0d0/4.0d0) * Ccurlmur
      Ccurlmui = (1.0d0/4.0d0) * Ccurlmui

      return
      end subroutine curlCmu
c
c     ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     author: Frederic D.R. Bonnet, date 7th of June 2001.
c     subroutine to calculate the spinor trace of the quark propagator
c
      subroutine spintrace(pmrSCT,pmiSCT,pmrCT,pmiCT)
      implicit none
      include 'latticeSize.h'
      include 'LatParamtrans.h'

c     global variables

      integer,parameter                                     :: nmx=-nx/2+1,npx=nx/2
      integer,parameter                                     :: nmy=-ny/2+1,npy=ny/2
      integer,parameter                                     :: nmz=-nz/2+1,npz=nz/2
      integer,parameter                                     :: nmt=-nt/2+1,npt=nt/2
      double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt)    :: pmrSCT,pmiSCT
cmf$  layout pmrSCT(:news,:news,:news,:news)
cmf$  layout pmiSCT(:news,:news,:news,:news)
      double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt,nd,nd):: pmrCT,pmiCT
cmf$  layout pmrCT(:news,:news,:news,:news,:serial,:serial)
cmf$  layout pmiCT(:news,:news,:news,:news,:serial,:serial)

c     local variables

      integer                                                    :: id

c     start of the excution commands.

      pmrSCT = 0.0d0
      pmiSCT = 0.0d0

      do id=1,nd
         pmrSCT = pmrSCT + pmrCT(:,:,:,:,id,id)
         pmiSCT = pmiSCT + pmiCT(:,:,:,:,id,id)
      end do

      pmrSCT = (1.0d0/4.0d0) * pmrSCT
      pmiSCT = (1.0d0/4.0d0) * pmiSCT

      return
      end subroutine spintrace
c
c     ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Author: Frederic D.R. Bonnet; date: 4th of June 2001.
c     subroutine to write out the curli C and curli B to disk.
c     the routine writes out the curli on a configuration basis
c     with a selection of the which array it needs to do.
c     the switch for the curli B is 0; and the curli C_\mu is 1
c
      subroutine CurliCandBout(Bcurlr,Bcurli,Ccurlmur,Ccurlmui,qpk,
     &                         xkappa,dataf,ikappa,bcx,bcy,bcz,bct)
      implicit none
      include 'latticeSize.h'
      include 'LatParamtrans.h'

c     global variables.

      integer,parameter                                          :: nmx=-nx/2+1,npx=nx/2
      integer,parameter                                          :: nmy=-ny/2+1,npy=ny/2
      integer,parameter                                          :: nmz=-nz/2+1,npz=nz/2
      integer,parameter                                          :: nmt=-nt/2+1,npt=nt/2

      integer                                                    :: ikappa

      double precision                                           :: bcx,bcy,bcz,bct
      double precision,dimension(nkappa)                      :: xkappa
cmf$  layout xkappa(:serial)
      character(len=100),dimension(10)                        :: dataf
cmf$  layout dataf(:serial)
      character(len=5),dimension(nkappa)                      :: qpk
cmf$  layout qpk(:serial)

      double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt)    :: Bcurlr,Bcurli
cmf$  layout Bcurlr(:news,:news,:news,:news)
cmf$  layout Bcurli(:news,:news,:news,:news)
      double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt,mu)
     &                                                        :: Ccurlmur,Ccurlmui
cmf$  layout Ccurlmur(:news,:news,:news,:news,:serial)
cmf$  layout Ccurlmui(:news,:news,:news,:news,:serial)

c     local variables.

      character(len=100)                                      :: datafile

      integer                                                 :: imu

      interface
       function strlen(string)
        implicit none
        character*(*) string
        integer                                               :: strlen
        integer                                               :: i, blank
       end function strlen
      end interface

c     start of the execution commands.

      datafile = dataf(1)
      datafile = datafile(1:strlen(datafile))//qpk(ikappa)

      open(201,file=datafile,form='unformatted',status='new',action='write')

      write(201) xkappa(ikappa),bcx,bcy,bcz,bct

      write(201) Bcurlr(:,:,:,:)
      write(201) Bcurli(:,:,:,:)

      close(201)

      datafile = dataf(2)
      datafile = datafile(1:strlen(datafile))//qpk(ikappa)

      open(301,file=datafile,form='unformatted',status='new',action='write')

      write(301) xkappa(ikappa),bcx,bcy,bcz,bct

      do imu=1,mu
```

```
           write(301) Ccurlmur(:,:,:,:,imu)
           write(301) Ccurlmui(:,:,:,:,imu)
        end do

        close(301)

        return
        end subroutine CurliCandBout
c
c     ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Author: Frederic D.R. Bonnet; date: 29th of June 2000.
c     this subroutine reads in the quark propagators.
c
      subroutine ReadProp(qp,pr,pi,flq,xkappa,ikappa,jx,jy,jz,jt)
      implicit none
      include 'latticeSize.h'
      include 'LatParamtrans.h'

c     global variables.

      integer,parameter                                  :: nmx=-nx/2+1,npx=nx/2
      integer,parameter                                  :: nmy=-ny/2+1,npy=ny/2
      integer,parameter                                  :: nmz=-nz/2+1,npz=nz/2
      integer,parameter                                  :: nmt=-nt/2+1,npt=nt/2

      integer                                            :: ikappa
      integer                                            :: flq
      integer                                            :: jx,jy,jz,jt
      character(len=100),dimension(nkappa)               :: qp
      double precision,dimension(nkappa)                 :: xkappa
cmf$  layout xkappa(:serial)

      double precision,dimension(nx*ny*nz,nt,nc,nd,nc,nd)     :: pr,pi
cmf$  layout pr(:news,:news,:serial,:serial,:serial,:serial)
cmf$  layout pi(:news,:news,:serial,:serial,:serial,:serial)

c     local variables.

      integer,parameter                                  :: nreal=2
      integer,parameter                                  :: ngamma=3
      integer,parameter                                  :: nxyz = nx*ny*nz

      logical                                            :: uexists=.true.
      character(len=80)                                  :: quarkprop

c     overlap stuff

      double precision,dimension(nt)                     :: pionm,pionpr,pionpi
cmf$  layout pionm(:serial)
cmf$  layout pionpr(:serial)
cmf$  layout pionpi(:serial)
      double precision,dimension(nd,nd)                  :: deltafunc,gam5
```

```
cmf$  layout deltafunc(:serial,:serial)
cmf$  layout gam5(:serial,:serial)
      double precision,dimension(nd,nd,ngamma,nreal)          :: gammai,gamma5i
cmf$  layout  gammai(:serial,:serial,:serial,:serial)
cmf$  layout gamma5i(:serial,:serial,:serial,:serial)
      double precision,dimension(nx*ny*nz*nt,nc,nd,nc,nd)     :: gr,gi
cmf$  layout gr(:news,:serial,:serial,:serial,:serial)
cmf$  layout gi(:news,:serial,:serial,:serial,:serial)
      double precision,dimension(nx*ny*nz,nt,nc,nd,nc,nd)     :: p2r,p2i
cmf$  layout p2r(:news,:news,:serial,:serial,:serial,:serial)
cmf$  layout p2i(:news,:news,:serial,:serial,:serial,:serial)

      integer                                                 :: ixyz
      integer                                                 :: it
      integer                                                 :: id,jd,ic

      interface
         subroutine readZprop(quarkprop,gr,gi)
          implicit none
          include 'latticeSize.h'
          include 'LatParamtrans.h'
          character(len=80)                                     :: quarkprop
          double precision,dimension(nx*ny*nz*nt,nc,nd,nc,nd)   :: gr,gi
cmf$  layout gr(:news,:serial,:serial,:serial,:serial)
cmf$  layout gi(:news,:serial,:serial,:serial,:serial)
         end subroutine readZprop
         subroutine pion(pr,pi,pionm,pionpr,pionpi,deltafunc,gam5,gammai,gamma5i,it)
          implicit none
          include 'latticeSize.h'
          include 'LatParamtrans.h'
          integer                                               :: it
          integer,parameter                                     :: nreal=2
          integer,parameter                                     :: ngamma=3
          double precision,dimension(nt)                        :: pionm,pionpr,pionpi
cmf$  layout pionm(:serial)
cmf$  layout pionpr(:serial)
cmf$  layout pionpi(:serial)
          double precision,dimension(nd,nd)                     :: deltafunc,gam5
cmf$  layout deltafunc(:serial,:serial)
cmf$  layout gam5(:serial,:serial)
          double precision,dimension(nd,nd,ngamma,nreal)        :: gammai,gamma5i
cmf$  layout  gammai(:serial,:serial,:serial,:serial)
cmf$  layout gamma5i(:serial,:serial,:serial,:serial)
          double precision,dimension(nx*ny*nz,nt,nc,nd,nc,nd)   :: pr,pi
cmf$  layout pr(:news,:news,:serial,:serial,:serial,:serial)
cmf$  layout pi(:news,:news,:serial,:serial,:serial,:serial)
         end subroutine pion
         subroutine deltaNDgam(deltafunc,gam5,gammai,gamma5i)
          implicit none
          integer,parameter                                     :: nd=4 !direction
          integer,parameter                                     :: nreal=2
          integer,parameter                                     :: ngamma=3
          double precision,dimension(nd,nd)                    :: deltafunc,gam5
cmf$  layout deltafunc(:serial,:serial)
cmf$  layout gam5(:serial,:serial)
          double precision,dimension(nd,nd,ngamma,nreal)       :: gammai,gamma5i
cmf$  layout  gammai(:serial,:serial,:serial,:serial)
cmf$  layout gamma5i(:serial,:serial,:serial,:serial)
         end subroutine deltaNDgam
      end interface

c     start of the execution commands.

c     initializing the arrays before allocation.

      pr = 0.0d0
      pi = 0.0d0

      if ( flq == 0 .or. flq == 1 ) then

         if ( flq == 0 ) then
         open(200,file='ptQ'//qp(ikappa),form='unformatted',
     &        status='old',action='read')
         elseif ( flq == 1 ) then
         open(200,file='ptFLQ'//qp(ikappa),form='unformatted',
     &        status='old',action='read')
         end if

         do it = 1,nt
            read(200) pr(:,it,:,:,:,:)
            read(200) pi(:,it,:,:,:,:)
         end do

         close(200)

      elseif ( flq == 2 ) then

c     checkin the existence of the propagators.

         quarkprop = 'ptQ'//qp(ikappa)

         inquire(file=quarkprop,exist=uexists)

         if(uexists) then
            write(*,'(a,2x,a40,x,a)')'the quark propagator:',quarkprop,'exists'
            write(*,'(a,2x,a40)')'we are now proceeding with the reading of:',
     &                           quarkprop
         elseif(.not. uexists ) then
            write(*,'(a,2x,a40,x,a)')'the quark propagator:',
     &                               quarkprop,'does not exists'
            stop
         end if

c     initializing the pion propagators
```

```
         write(1,'(a,12x,a,11x,a)')'time','C(t)','mt=ln(|C(t-1)/C(t)|)'

         pionm = 0.0d0           !pi0+ prop function \bar{\psi}\psi
         pionpr = 0.0d0          !pi0- prop function \bar{\psi}\gamma_5\psi
         pionpi = 0.0d0

         call deltaNDgam(deltafunc,gam5,gammai,gamma5i)

         call readZprop(quarkprop,gr,gi)

         do it=1,nt

            do ixyz=1,nxyz
               p2r(ixyz,it,:,:,:,:) = gr(ixyz+(it-1)*nxyz,:,:,:,:)
               p2i(ixyz,it,:,:,:,:) = gi(ixyz+(it-1)*nxyz,:,:,:,:)
            end do

            call pion(p2r,p2i,pionm,pionpr,pionpi,deltafunc,gam5,gammai,gamma5i,it)

         end do

         do jd=1,2
            pr(:,:,:,:,:,jd) =   (1.0d0/sqrt(2.0d0)) *
     &                           ( p2r(:,:,:,:,:,jd) + p2r(:,:,:,:,:,jd+2) )
            pi(:,:,:,:,:,jd) =   (1.0d0/sqrt(2.0d0)) *
     &                           ( p2i(:,:,:,:,:,jd) + p2i(:,:,:,:,:,jd+2) )
         end do

         do jd=3,4
            pr(:,:,:,:,:,jd) = - (1.0d0/sqrt(2.0d0)) *
     &                           ( p2r(:,:,:,:,:,jd-2) - p2r(:,:,:,:,:,jd) )
            pi(:,:,:,:,:,jd) = - (1.0d0/sqrt(2.0d0)) *
     &                           ( p2i(:,:,:,:,:,jd-2) - p2i(:,:,:,:,:,jd) )
         end do

      end if

      return
      end subroutine ReadProp
c
c     cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Returns the significant length of a string.
c     Every character is significant with the
c     exception of:
c     blank     (32)
c     null      (0)
c     reqd. routines - NONE
c
      function strlen(string)
      implicit none
```

```
c     global variables.

      character*(*)                                         :: string
      integer                                               :: strlen

c     local variables.

      integer                                               :: i, blank

c     start of the execution commands.

      blank = ichar(' ')

      strlen = len(string)
      i = ichar(string(strlen:strlen))
      do while ((i.eq.blank .or. i.eq.0) .and. strlen.gt.0)
         strlen = strlen - 1
         i = ichar(string(strlen:strlen))
      end do
      return
      end
c
c     ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     author: frederic D.R. Bonnet, date: 27th of October 2000.
c     subroutine to calculate pion correlation function and effective mass.
c
      subroutine pion(pr,pi,pionm,pionpr,pionpi,deltafunc,gam5,gammai,gamma5i,it)
      implicit none
      include 'latticeSize.h'
      include 'LatParamtrans.h'

c     global variables.

      integer,parameter                                     :: nreal=2
      integer,parameter                                     :: ngamma=3

      double precision,dimension(nt)                        :: pionm,pionpr
      double precision,dimension(nt)                        :: pionpi,rho,scalar1m
cmf$  layout pionm(:serial)
cmf$  layout pionpr(:serial)
cmf$  layout pionpi(:serial)
cmf$  layout rho(:serial)
cmf$  layout scalar1m(:serial)
      double precision,dimension(nd,nd)                     :: deltafunc,gam5
cmf$  layout deltafunc(:serial,:serial)
cmf$  layout gam5(:serial,:serial)
      double precision,dimension(nd,nd,ngamma,nreal)        :: gammai,gamma5i
cmf$  layout  gammai(:serial,:serial,:serial,:serial)
cmf$  layout gamma5i(:serial,:serial,:serial,:serial)
      double precision,dimension(nx*ny*nz,nt,nc,nd,nc,nd)   :: pr,pi
cmf$  layout pr(:news,:news,:serial,:serial,:serial,:serial)
cmf$  layout pi(:news,:news,:serial,:serial,:serial,:serial)
```

```
      integer                                                 :: it

c     local variables.

      double precision                                        :: trof0,tiof0

      double precision,dimension(nt)                          :: pionMss,pionCt
cmf$  layout pionMss(:serial)
cmf$  layout pionCt(:serial)
      double precision,dimension(nx*ny*nz)                    :: v1r,v1i
cmf$  layout v1r(:news)
cmf$  layout v1i(:news)

      double precision,dimension(nx*ny*nz,nc,nd,nc,nd)        :: pmesr,pmesi
cmf$  layout pmesr(:news,:serial,:serial,:serial,:serial)
cmf$  layout pmesi(:news,:serial,:serial,:serial,:serial)

      integer                                                 :: ic,jc
      integer                                                 :: jd
      integer                                                 :: ialp,ibet,igam,idel

c     start of the execution commands.

c     first initiallise the variables

      pionCt(it) = 0.0d0

c     we are now calculating the pion mass and the effective mass of the pion.

      pionCt(it) = pionCt(it) + ( sum(pr(:,it,:,:,:,:)**2) +
     &                            sum(pi(:,it,:,:,:,:)**2) )

      pionMss(it) = 0.0d0
      if ( it >= 2 ) then
         pionMss(it) = log ( abs( pionCt(it-1) / pionCt(it) ) )
      end if

      write(1,'(i3,2x,f21.16,2x,f21.16)')it,pionCt(it),pionMss(it)

      return
      end subroutine pion
c
c     ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     author: Frederic D.R. Bonnet 2/11/00.
c     subroutine to create a delta function.
c
      subroutine deltaNDgam(deltafunc,gam5,gammai,gamma5i)
      implicit none

c     global variables
```

```
      integer,parameter                                          :: nd=4
      integer,parameter                                          :: nreal=2
      integer,parameter                                          :: ngamma=3

      double precision,dimension(nd,nd)                          :: deltafunc,gam5
cmf$  layout deltafunc(:serial,:serial)

      double precision,dimension(nd,nd,ngamma,nreal)             :: gammai,gamma5i
cmf$  layout  gammai(:serial,:serial,:serial,:serial)
cmf$  layout gamma5i(:serial,:serial,:serial,:serial)

c     local variables

      integer                                                    :: id,jd

c     start of the execution commands

      gam5 = 0.0d0
      deltafunc = 0.0d0

      do id=1,nd
         deltafunc(id,id) = 1.0d0
      end do

      do id=1,2
         do jd=1,2
            gam5(id,jd+2) = deltafunc(id,jd)
            gam5(id+2,jd) = deltafunc(id,jd)
         end do
      end do

      gammai = 0.0d0

c     \gamma 1

      gammai(1,4,1,2) = - deltafunc(1,1)
      gammai(2,3,1,2) = - deltafunc(2,2)
      gammai(3,2,1,2) =   deltafunc(3,3)
      gammai(4,1,1,2) =   deltafunc(4,4)

c     \gamma 2

      gammai(1,4,2,1) = - deltafunc(1,1)
      gammai(2,3,2,1) =   deltafunc(2,2)
      gammai(3,2,2,1) =   deltafunc(3,3)
      gammai(4,1,2,1) = - deltafunc(4,4)

c     \gamma 3

      gammai(1,3,3,2) = - deltafunc(1,1)
      gammai(2,4,3,2) =   deltafunc(2,2)
      gammai(3,1,3,2) =   deltafunc(3,3)
      gammai(4,2,3,2) = - deltafunc(4,4)

c     now constracting the product of \gamma_5*( \gamma_1,\gamma_2,\gamma_3 )

      gamma5i = 0.0d0

c     \gamma_5*\gamma_1

      gamma5i(1,2,1,2) = - deltafunc(1,1)
      gamma5i(2,1,1,2) = - deltafunc(2,2)
      gamma5i(3,4,1,2) =   deltafunc(3,3)
      gamma5i(4,3,1,2) =   deltafunc(4,4)

c     \gamma_5*\gamma_2

      gamma5i(1,2,2,1) = - deltafunc(1,1)
      gamma5i(2,1,2,1) =   deltafunc(2,2)
      gamma5i(3,4,2,1) =   deltafunc(3,3)
      gamma5i(4,3,2,1) = - deltafunc(4,4)

c     \gamma_5*\gamma_3

      gamma5i(1,3,3,2) = - deltafunc(1,1)
      gamma5i(2,4,3,2) =   deltafunc(2,2)
      gamma5i(3,1,3,2) =   deltafunc(3,3)
      gamma5i(4,2,3,2) = - deltafunc(4,4)

      return
      end subroutine deltaNDgam
c
c     ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     author: Frederic D.R. Bonnet, date 13th of June 2001.
c     subroutine to create a delta function at all lattice sites.
c
      subroutine getdelta(deltar)
      implicit none
      include 'latticeSize.h'
      include 'LatParamtrans.h'

c     global variables

      double precision,dimension(nx*ny*nz,nt,nc,nd,nc,nd)     :: deltar
cmf$  layout deltar(:news,:news,:serial,:serial,:serial,:serial)

c     local variables

      integer                                                  :: ic,id

c     start of the execution commands

      deltar = 0.0d0
      do id=1,nd
         do ic=1,nc
            deltar(:,:,ic,id,ic,id) = 1.0d0
         end do
      end do

      return
      end subroutine getdelta
c
c     cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     author: Frederic D.R. Bonnet 1/11/00.
c     routine to read the overlap fermions quark propagators imported by Jianbo Zhang.
c
      subroutine readZprop(quarkprop,gr,gi)
      implicit none
      include 'latticeSize.h'
      include 'LatParamtrans.h'

c     global variables.

      character(len=80)                                       :: quarkprop

c                               sink dirac ------     -------- source dirac
c                                               |     |
c
      double precision,dimension(nx*ny*nz*nt,nc,nd,nc,nd)     :: gr,gi
cmf$  layout gr(:news,:serial,:serial,:serial,:serial)
cmf$  layout gi(:news,:serial,:serial,:serial,:serial)

c     local variables.

      integer,parameter                                       :: nreal=2

      integer,parameter                                       :: nxyzt = nx*ny*nz*nt

      real(8),dimension(nxyzt)                                :: propa
cmf$  layout propa(:news)

      integer                                                 :: n1
      integer                                                 :: ic,jc,ireal,i
      integer                                                 :: id,jd
      integer                                                 :: islot
      integer                                                 :: jdsrc,jcsrc

c     start of the execution commands.

      inquire(iolength=n1)propa

c     n1=8*nxyzt

      open(9,file=quarkprop, status='unknown',
     &     form='unformatted',access='direct',recl=n1)
```

```
      do jdsrc=1,nd
         do jcsrc=1,nc

            do jd =1,nd
               do ireal=1,nreal

                  if ( ireal == 1 ) then
                     do jc =1,nc

                        islot=jc+nc*(jd-1+nd*(ireal-1+nreal*(jcsrc-1+nc*(jdsrc-1))))
                        read(9,rec=islot)(propa(i),i=1,nxyzt)

                        do i=1,nxyzt
                           gr(i,jc,jd,jcsrc,jdsrc)=propa(i)
                        enddo
                     end do
                  elseif( ireal == 2 ) then

                     do jc =1,nc

                        islot=jc+nc*(jd-1+nd*(ireal-1+nreal*(jcsrc-1+nc*(jdsrc-1))))
                        read(9,rec=islot)(propa(i),i=1,nxyzt)

                        do i=1,nxyzt
                           gi(i,jc,jd,jcsrc,jdsrc)=propa(i)
                        enddo
                     end do
                  end if

               end do
            end do

         end do
      end do

      close(9)

      return
      end subroutine readZprop
c
c     cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Author: Frederic D.R. Bonnet; date: 29th of August 2000.
c     subroutine to tranform the quark propagators in coordinate space before any tracing
c     but after the propagator has been calculated by fermion matrix inverter with
c     gauge fixed link variable.
c
      subroutine GTransSofx(lastconfig,pr,pi,pgr,pgi,jx,jy,jz,jt)
      implicit none
      include 'latticeSize.h'
      include 'LatParamtrans.h'
```

```
c     global variables

      integer,parameter                                       :: nmx=-nx/2+1,npx=nx/2
      integer,parameter                                       :: nmy=-ny/2+1,npy=ny/2
      integer,parameter                                       :: nmz=-nz/2+1,npz=nz/2
      integer,parameter                                       :: nmt=-nt/2+1,npt=nt/2

      integer                                                 :: jx,jy,jz,jt
      character(len=80)                                       :: lastconfig

      double precision,dimension(nx*ny*nz,nt,nc,nd,nc,nd)     :: pr,pi
cmf$  layout pr(:news,:news,:serial,:serial,:serial,:serial)
cmf$  layout pi(:news,:news,:serial,:serial,:serial,:serial)

      double precision,dimension(nx*ny*nz,nt,nc,nd,nc,nd)     :: pgr,pgi
cmf$  layout pgr(:news,:news,:serial,:serial,:serial,:serial)
cmf$  layout pgi(:news,:news,:serial,:serial,:serial,:serial)

c     local variables

      double precision                                        :: beta

      double precision,dimension(nx,ny,nz,nt,nc,nc)           :: tgr, tgi
cmf$  layout tgr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout tgi(:news,:news,:news,:news,:serial,:serial)
      double precision,dimension(nx*ny*nz,nt,nc,nc)           :: tcgr, tcgi
cmf$  layout tcgr(:news,:news,:serial,:serial)
cmf$  layout tcgi(:news,:news,:serial,:serial)
      double precision,dimension(nx*ny*nz,nt,nc,nd,nc,nd)     :: p1r,p1i
cmf$  layout p1r(:news,:news,:serial,:serial,:serial,:serial)
cmf$  layout p1i(:news,:news,:serial,:serial,:serial,:serial)

      integer                                                 :: counter
      integer                                                 :: ix,iy,iz,icounter
      integer                                                 :: ic,jc,kc,id,jd

      interface
         subroutine Gaugein(lastconfig,tgr,tgi,beta)
          implicit none
          include 'latticeSize.h'
          include 'LatParamtrans.h'
          double precision                                    :: beta
          character(len=80)                                   :: lastconfig
          double precision,dimension(nx,ny,nz,nt,nc,nc)       :: tgr, tgi
cmf$  layout tgr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout tgi(:news,:news,:news,:news,:serial,:serial)
         end subroutine Gaugein
      end interface

c     start of the execution commands

c     calling the fully gauge fix variables created from gauge fixing.
```

```
      call Gaugein(lastconfig,tgr,tgi,beta)

c     conpressing the spacial into one lattice site.

      tcgr = 0.0d0
      tcgi = 0.0d0

      counter = 0
      do iz=1,nz
         do iy=1,ny
            do ix=1,nx
               counter = counter + 1
               tcgr(counter,:,:,:) = tgr(ix,iy,iz,:,:,:)
               tcgi(counter,:,:,:) = tgi(ix,iy,iz,:,:,:)
            end do
         end do
      end do

c     Now multiplying the propagators by the fully gauge transformed variable
c     the compressed tgr,tgi : tcgr, tcgi. In color space.
c     S'(x,0) = V(x)S(x,0)V^{\dagger}(0)

c     the acting the gauge transformatin on the right.

      do id=1,nd
         do jd=1,nd

            p1r(:,:,:,id,:,jd) = 0.0d0
            p1i(:,:,:,id,:,jd) = 0.0d0

            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     p1r(:,:,ic,id,jc,jd) = p1r(:,:,ic,id,jc,jd) +
     &           (   pr(:,:,ic,id,kc,jd) * tcgr(jx+(jy-1)*nx+(jz-1)*nx*ny,jt,jc,kc) +
     &               pi(:,:,ic,id,kc,jd) * tcgi(jx+(jy-1)*nx+(jz-1)*nx*ny,jt,jc,kc) )
                     p1i(:,:,ic,id,jc,jd) = p1i(:,:,ic,id,jc,jd) +
     &           ( - pr(:,:,ic,id,kc,jd) * tcgi(jx+(jy-1)*nx+(jz-1)*nx*ny,jt,jc,kc) +
     &               pi(:,:,ic,id,kc,jd) * tcgr(jx+(jy-1)*nx+(jz-1)*nx*ny,jt,jc,kc) )
                  end do
               end do
            end do

            pgr(:,:,:,id,:,jd) = 0.0d0
            pgi(:,:,:,id,:,jd) = 0.0d0

            do ic=1,nc
               do jc=1,nc
                  do kc=1,nc
                     pgr(:,:,ic,id,jc,jd) = pgr(:,:,ic,id,jc,jd) +
     &                    ( tcgr(:,:,ic,kc) * p1r(:,:,kc,id,jc,jd) -
     &                      tcgi(:,:,ic,kc) * p1i(:,:,kc,id,jc,jd) )
                     pgi(:,:,ic,id,jc,jd) = pgi(:,:,ic,id,jc,jd) +
     &                    ( tcgr(:,:,ic,kc) * p1i(:,:,kc,id,jc,jd) +
     &                      tcgi(:,:,ic,kc) * p1r(:,:,kc,id,jc,jd) )
                  end do
               end do
            end do

         end do
      end do

      return
      end subroutine GTransSofx
c
c     ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Author: Frederic D.R. Bonnet; date: 29th of August 2000.
c     subroutine to write out the fixed links after a total gauge
c     fixing.

      subroutine Gaugein(lastconfig,tgr,tgi,beta)
      implicit none
      include 'latticeSize.h'
      include 'LatParamtrans.h'

c     global variables

      double precision                                         :: beta
      character(len=80)                                        :: lastconfig

      double precision,dimension(nx,ny,nz,nt,nc,nc)            :: tgr, tgi
cmf$  layout tgr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout tgi(:news,:news,:news,:news,:serial,:serial)

c     local variables

      integer                                                  :: nxf,nyf,nzf,ntf
      integer                                                  :: ic
      logical                                                  :: uexists=.true.
      character(len=84)                                        :: gin

      interface
       subroutine fixsgauge(tgr,tgi)
        implicit none
        include 'latticeSize.h'
        include 'LatParamtrans.h'
        double precision,dimension(nx,ny,nz,nt,nc,nc)          :: tgr,tgi
cmf$  layout tgr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout tgi(:news,:news,:news,:news,:serial,:serial)
       end subroutine fixsgauge
       function strlen(string)
        implicit none
        character*(*) string
        integer                                                 :: strlen
        integer                                                 :: i, blank
       end function strlen
      end interface

c     start of the execution commands.

      gin = lastconfig(1:strlen(lastconfig))//'.gag'

      inquire(file=gin,exist=uexists)
      if(uexists) then
         write(*,'(a,2x,a25,x,a)')'the gauge transformed configuration:',gin,'exists'
         write(*,'(a,2x,a25)')'we are now proceeding with the reading of:',gin
      elseif(.not. uexists ) then
         write(*,'(a,2x,a25,x,a)')'the gauge transformed configuration:',gin,
     &                            'does not exists'
         stop
      end if

      open(29,file=gin,form='unformatted',status='old',action='read')

      read(29) beta, nxf, nyf, nzf, ntf

      do ic=1,nc-1
         read(29) tgr(:,:,:,:,ic,:)
         read(29) tgi(:,:,:,:,ic,:)
      end do

      close(29)

      call fixsgauge(tgr,tgi)

      return
      end subroutine Gaugein
c
c     cccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Author: Frederic D.R. Bonnet; date: 29th of June 1998.
c     subroutine that fixes the gauge links. This subroutine needs to
c     be called after a the transformation the
c     purpose being to keep the links within the SU(3) algebra. This
c     routine reconstruct the su3 element by doing the row by row
c     orthonormailzation method. Unlike fixsu3 the direction mu is not
c     taken into consideration.
c
      subroutine fixsgauge(tgr,tgi)
      implicit none
      include 'latticeSize.h'
      include 'LatParamtrans.h'

c     global variables

      double precision,dimension(nx,ny,nz,nt,nc,nc)          :: tgr,tgi
```

```
cmf$  layout tgr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout tgi(:news,:news,:news,:news,:serial,:serial)

c     local variables

      integer,dimension(5)                                    :: yvector
cmf$  layout yvector(:serial)
      double precision,dimension(nx,ny,nz,nt)                 :: normr,normi
cmf$  layout normr(:news,:news,:news,:news)
cmf$  layout normi(:news,:news,:news,:news)
      integer                                                 :: jc,imu

c
c     First create an array to be looped over below
c     y(1) = 1 , y(2) = 2 , y(3) = 3 , y(4) = 1 , y(5) = 2

      do jc=1,nc
         yvector(jc) = jc
      end do
      do jc=4,5
         yvector(jc) = jc - 3
      end do

c     We'll do a loop here to save memory demands

c
c     first normalise first row
c
      normr = sqrt( tgr(:,:,:,:,1,1)**2 + tgr(:,:,:,:,1,2)**2 +
     &              tgi(:,:,:,:,1,1)**2 + tgi(:,:,:,:,1,2)**2 +
     &              tgr(:,:,:,:,1,3)**2 + tgi(:,:,:,:,1,3)**2 )

      do jc=1,nc
         tgr(:,:,:,:,1,jc) = tgr(:,:,:,:,1,jc) / normr
         tgi(:,:,:,:,1,jc) = tgi(:,:,:,:,1,jc) / normr
      end do
c
c     now compute row2 - (row2 dot row1)*row1
c
      normr = tgr(:,:,:,:,2,1) * tgr(:,:,:,:,1,1) +
     &        tgi(:,:,:,:,2,1) * tgi(:,:,:,:,1,1) +
     &        tgr(:,:,:,:,2,2) * tgr(:,:,:,:,1,2) +
     &        tgi(:,:,:,:,2,2) * tgi(:,:,:,:,1,2) +
     &        tgr(:,:,:,:,2,3) * tgr(:,:,:,:,1,3) +
     &        tgi(:,:,:,:,2,3) * tgi(:,:,:,:,1,3)

      normi = tgi(:,:,:,:,2,1) * tgr(:,:,:,:,1,1) -
     &        tgr(:,:,:,:,2,1) * tgi(:,:,:,:,1,1) +
     &        tgi(:,:,:,:,2,2) * tgr(:,:,:,:,1,2) -
     &        tgr(:,:,:,:,2,2) * tgi(:,:,:,:,1,2) +
     &        tgi(:,:,:,:,2,3) * tgr(:,:,:,:,1,3) -
     &        tgr(:,:,:,:,2,3) * tgi(:,:,:,:,1,3)
```

```
       do jc=1,nc
          tgr(:,:,:,:,2,jc) = tgr(:,:,:,:,2,jc) -
     &                       (  normr * tgr(:,:,:,:,1,jc) -
     &                          normi * tgi(:,:,:,:,1,jc) )
          tgi(:,:,:,:,2,jc) = tgi(:,:,:,:,2,jc) -
     &                       (  normr * tgi(:,:,:,:,1,jc) +
     &                          normi * tgr(:,:,:,:,1,jc) )
       end do

c      Now normalise the second row

       normr = sqrt( tgr(:,:,:,:,2,1)**2 + tgi(:,:,:,:,2,1)**2 +
     &               tgr(:,:,:,:,2,2)**2 + tgi(:,:,:,:,2,2)**2 +
     &               tgr(:,:,:,:,2,3)**2 + tgi(:,:,:,:,2,3)**2 )

       do jc=1,nc
          tgr(:,:,:,:,2,jc) = tgr(:,:,:,:,2,jc) / normr
          tgi(:,:,:,:,2,jc) = tgi(:,:,:,:,2,jc) / normr
       end do

c      now generate row3 = row1 cross row2

       do jc=1,nc
       tgr(:,:,:,:,3,jc) =   tgr(:,:,:,:,1,yvector(jc+1)) *
     &                       tgr(:,:,:,:,2,yvector(jc+2)) -
     &                       tgi(:,:,:,:,1,yvector(jc+1)) *
     &                       tgi(:,:,:,:,2,yvector(jc+2)) -
     &                       tgr(:,:,:,:,1,yvector(jc+2)) *
     &                       tgr(:,:,:,:,2,yvector(jc+1)) +
     &                       tgi(:,:,:,:,1,yvector(jc+2)) *
     &                       tgi(:,:,:,:,2,yvector(jc+1))
       tgi(:,:,:,:,3,jc) = - tgr(:,:,:,:,1,yvector(jc+1)) *
     &                       tgi(:,:,:,:,2,yvector(jc+2)) -
     &                       tgi(:,:,:,:,1,yvector(jc+1)) *
     &                       tgr(:,:,:,:,2,yvector(jc+2)) +
     &                       tgr(:,:,:,:,1,yvector(jc+2)) *
     &                       tgi(:,:,:,:,2,yvector(jc+1)) +
     &                       tgi(:,:,:,:,1,yvector(jc+2)) *
     &                       tgr(:,:,:,:,2,yvector(jc+1))
       end do

       return
       end subroutine fixsgauge
c
c      cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c      Author: Frederic D.R. Bonnet; date: 29th of June 2000.
c      This subroutine is to calculate the fourier transform of the n-point functions.
c      This subroputine maps a variable from cartesian coordinates space to a variable
c      in momentum space. it takes in the color trace function and returns it in
c      momentum space for all momentum value in the Brillouin zone [-pi/a,pi/a].
c
```

```
      subroutine fourier(prTr,piTr,pmrTr,pmiTr,jx,jy,jz,jt)
      implicit none
      include 'latticeSize.h'
      include 'LatParamtrans.h'

c     global variables

      integer,parameter                                     :: nmx=-nx/2+1,npx=nx/2
      integer,parameter                                     :: nmy=-ny/2+1,npy=ny/2
      integer,parameter                                     :: nmz=-nz/2+1,npz=nz/2
      integer,parameter                                     :: nmt=-nt/2+1,npt=nt/2

      integer                                               :: jx,jy,jz,jt

      double precision,dimension(nx,ny,nz,nt,nd,nd)         :: prTr,piTr
cmf$  layout prTr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout piTr(:news,:news,:news,:news,:serial,:serial)
      double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt,nd,nd):: pmrTr,pmiTr
cmf$  layout pmrTr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout pmiTr(:news,:news,:news,:news,:serial,:serial)

c     local variables

      integer                                               :: px,py,pz,pt
      double precision                                      :: pie
      double precision                                      :: qx,qy,qz,qt

      double precision,dimension(nx,ny,nz,nt)               :: phaser,phasei
cmf$  layout phaser(:news,:news,:news,:news)
cmf$  layout phasei(:news,:news,:news,:news)

      integer                                               :: id,jd
      integer                                               :: ix,iy,iz,it

c     calculating the fourier transform of the traced quark propagator.
c     The two point function (G2) or three point function (G3).
c     The inverse Fourier transform is defined as
c     S(p)=\sum_{n}\exp[(2i{\pi}/n)n{\cdot}p]S(n)
c     for p_\mu=(2-n_\mu)/2,..,n_\mu/2.

      pie = 4.0d0 * atan(1.0d0)

      do pt=nmt,npt
         do pz=nmz,npz
            do py=nmy,npy
               do px=nmx,npx
                  qx = ( ( 2.0d0 * pie ) / nx ) * px
                  qy = ( ( 2.0d0 * pie ) / ny ) * py
                  qz = ( ( 2.0d0 * pie ) / nz ) * pz
                  qt = ( ( 2.0d0 * pie ) / nt ) * ( pt - 1.0d0/2.0d0 )
                  forall( ix=1:nx,iy=1:ny,iz=1:nz,it=1:nt )
     &     phaser(ix,iy,iz,it) = cos( qx*(ix-jx) + qy*(iy-jy) +
     &                                qz*(iz-jz) + qt*(it-jt) )
                forall( ix=1:nx,iy=1:ny,iz=1:nz,it=1:nt )
     &     phasei(ix,iy,iz,it) = sin( qx*(ix-jx) + qy*(iy-jy) +
     &                                qz*(iz-jz) + qt*(it-jt) )
                do id=1,nd
                   do jd=1,nd
                      pmrTr(px,py,pz,pt,id,jd) =
     &                sum( prTr(:,:,:,:,id,jd) * phaser -
     &                     piTr(:,:,:,:,id,jd) * phasei )
                      pmiTr(px,py,pz,pt,id,jd) =
     &                sum( prTr(:,:,:,:,id,jd) * phasei +
     &                     piTr(:,:,:,:,id,jd) * phaser )
                   end do
                end do
             end do
          end do
       end do
      end do

      return
      end subroutine fourier
c
c     cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Author: Frederic D.R. Bonnet; date: 29th of June 2000.
c     This subroutine is to multiply the quark propagators, by an arbitrary
c     gamma matrix in the Sakurai representation.
c
      subroutine gammaXSp(pmrTrG,pmiTrG,pmrTr,pmiTr,imu)
      implicit none
      include 'latticeSize.h'
      include 'LatParamtrans.h'

c     global variables.

      integer,parameter                                     :: nmx=-nx/2+1,npx=nx/2
      integer,parameter                                     :: nmy=-ny/2+1,npy=ny/2
      integer,parameter                                     :: nmz=-nz/2+1,npz=nz/2
      integer,parameter                                     :: nmt=-nt/2+1,npt=nt/2

      integer                                               :: imu

      double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt,nd,nd):: pmrTr,pmiTr
cmf$  layout pmrTr(:news,:news,:news,:news,:serial,:serial)
cmf$  layout pmiTr(:news,:news,:news,:news,:serial,:serial)
      double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt,nd,nd)
     &                                                      :: pmrTrG,pmiTrG
cmf$  layout pmrTrG(:news,:news,:news,:news,:serial,:serial)
cmf$  layout pmiTrG(:news,:news,:news,:news,:serial,:serial)

c     local variables.

      double precision                                      :: gammaf
      integer                                                   :: id,jd,kd

c     start of the execution commands.

c     in the x-direction.

      if ( imu == 1 ) then

         do id=1,nd

            if( id == 1 ) then
               kd = 4
               gammaf = - 1.0d0
            else if( id == 2 ) then
               kd = 3
               gammaf = - 1.0d0
            else if( id == 3 ) then
               kd = 2
               gammaf =   1.0d0
            else if( id == 4 ) then
               kd = 1
               gammaf =   1.0d0
            end if

            do jd=1,nd
               pmrTrG(:,:,:,:,id,jd) = - gammaf * pmiTr(:,:,:,:,kd,jd)
               pmiTrG(:,:,:,:,id,jd) =   gammaf * pmrTr(:,:,:,:,kd,jd)
            end do
         end do

c     in the y-direction.

      else if ( imu == 2 ) then

         do id=1,nd

            if( id == 1 ) then
               kd = 4
               gammaf = - 1.0d0
            else if( id == 2 ) then
               kd = 3
               gammaf =   1.0d0
            else if( id == 3 ) then
               kd = 2
               gammaf =   1.0d0
            else if( id == 4 ) then
               kd = 1
               gammaf = - 1.0d0
            end if

            do jd=1,nd
               pmrTrG(:,:,:,:,id,jd) =   gammaf * pmrTr(:,:,:,:,kd,jd)
```

```
              pmiTrG(:,:,:,:,id,jd) =   gammaf * pmiTr(:,:,:,:,kd,jd)
           end do
        end do

c     in the z-direction.

      else if ( imu == 3 ) then

        do id=1,nd

           if( id == 1 ) then
              kd = 3
              gammaf = - 1.0d0
           else if( id == 2 ) then
              kd = 4
              gammaf =   1.0d0
           else if( id == 3 ) then
              kd = 1
              gammaf =   1.0d0
           else if( id == 4 ) then
              kd = 2
              gammaf = - 1.0d0
           end if

           do jd=1,nd
              pmrTrG(:,:,:,:,id,jd) = - gammaf * pmiTr(:,:,:,:,kd,jd)
              pmiTrG(:,:,:,:,id,jd) =   gammaf * pmrTr(:,:,:,:,kd,jd)
           end do
        end do

c     in the t-direction.

      else if ( imu == 4 ) then

        do id=1,nd

           if( id == 1 ) then
              kd = 1
              gammaf =   1.0d0
           else if( id == 2 ) then
              kd = 2
              gammaf =   1.0d0
           else if( id == 3 ) then
              kd = 3
              gammaf = - 1.0d0
           else if( id == 4 ) then
              kd = 4
              gammaf = - 1.0d0
           end if

           do jd=1,nd
              pmrTrG(:,:,:,:,id,jd) =   gammaf * pmrTr(:,:,:,:,kd,jd)
              pmiTrG(:,:,:,:,id,jd) =   gammaf * pmiTr(:,:,:,:,kd,jd)
           end do
        end do

     end if

     return
     end subroutine gammaXSp
```

E.24 Program to extract $M(q^2)$ and $Z(q^2)$ with Jackknife analysis

```
c    program to first read in gauge field configuration and quark
c    propagators. The program will then calcultate the color trace
c    fourier transform the data and (optional) will calculate the invert
c    of the 4x4 matrix in the Dirac space. Then calculate the full color trace
c    multiply by an arbitrary gamma matrix the inverted propagator to give the
c    Z(p) function.
c    ---------------------------------------------------------------------------
c        Quark-gluon vertex utility for SU(3) Gauge configuration
c    ---------------------------------------------------------------------------
c    Author: Frederic D.R. Bonnet; date: 14 of April 2000.
c                                    22 of November 2000.
c                                     Included the Overlap fermion routines
c                                     and front end ReadZlinks,ReadZprop
c                                    10th of June 2001.
c                                     Read in the curli B and curli C_\mu.
c                                    20th of June 2001.
c                                     Reproduce the results for the
c                                     Wilson fermion
c                                     for M^c(p) and Z^c(p).
c                                    25th of June 2001.
c                                     Insertion of the complement sets.
c                                    29th of June 2001.
c                                     Include the Jacknife analysis.
c                                    2nd  of Jully 2001.
c                                     Extrapolation to the chiral quark mass.
c
c    output files:
c    are propagator.
c    Transform.log.
c
c    To compile
c
c    f95 -fast -extend_source -convert big_endian -assume byterecl overlap.f
c          Newroutine.f transform.f -o
c    f95 -fast -extend_source overlap.f Newroutine.f Transform.f -o non
c          overlap fermions.
c    cmf -cm5 -vu -extend_source -f90syntax Transform.fcm -o outputfile
c    ---------------------------------------------------------------------------
```

```
      program JackMassAndZ
      implicit none
      include 'latticeSize.h'
      include 'LatParamtrans.h'

c     global variables.

      integer,parameter                                      :: ncon=nprop-1

      integer,parameter                                      :: nmx=-nx/2+1,npx=nx/2
      integer,parameter                                      :: nmy=-ny/2+1,npy=ny/2
      integer,parameter                                      :: nmz=-nz/2+1,npz=nz/2
      integer,parameter                                      :: nmt=-nt/2+1,npt=nt/2

      double precision,parameter                             :: rwil=1.0d0

      double precision                                       :: bcx,bcy,bcz,bct
      double precision                                       :: uzero=0.87208643d0
      double precision                                       :: mq
      double precision                                       :: kc
      double precision                                       :: ksim
      double precision                                       :: kappa
      double precision                                       :: Zpsi,Z0psi
      double precision                                       :: beta

c     local variables.

      logical                                                :: uexists=.true.

      integer                                                :: nkappaQP
      integer                                                :: ncount
      integer                                                :: jx,jy,jz,jt
      integer                                                :: flq
      character(len=4)                                       :: fixname
      character(len=4)                                       :: cfg
      character(len=80)                                      :: quarkprop
      character(len=80)                                      :: lastconfig,corename

      character(len=3),dimension(nprop)                      :: confnum
      character(len=80),dimension(nprop)                     :: confs
cmf$  layout confs(:serial)
cmf$  layout confnum(:serial)

      character(len=100)                                     :: datafile
      character(len=100),dimension(10)                       :: dataf
      character(len=5),dimension(nkappa)                     :: qpk
      character(len=100),dimension(nkappa)                   :: qp
cmf$  layout dataf(:serial)
cmf$  layout qpk(:serial)
cmf$  layout qp(:serial)

      double precision,dimension(mu)                         :: pmu
```

```
cmf$  layout pmu(:serial)

      double precision,dimension(nkappa/2)                   :: mqk      !mq(ikappa)
cmf$  layout mqk(:serial)
      double precision,dimension(nkappa)                     :: xkappa
cmf$  layout xkappa(:serial)

      double precision,dimension((npy-1)*npx*npt*(npz-1))     :: A0curlamr,A0curlami
      double precision,dimension((npy-1)*npx*npt*(npz-1))     :: B0curlamr,B0curlami
cmf$  layout A0curlamr(:news)
cmf$  layout A0curlami(:news)
cmf$  layout B0curlamr(:news)
cmf$  layout B0curlami(:news)
      double precision,dimension((npy-1)*npx*npt*(npz-1))     :: Acurlamr,Acurlami
      double precision,dimension((npy-1)*npx*npt*(npz-1))     :: Bcurlamr,Bcurlami
cmf$  layout Acurlamr(:news)
cmf$  layout Acurlami(:news)
cmf$  layout Bcurlamr(:news)
cmf$  layout Bcurlami(:news)

      double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt)
     &                                :: A0curlr,A0curli,Acurlr,Acurli
cmf$  layout A0curlr(:news,:news,:news,:news)
cmf$  layout A0curli(:news,:news,:news,:news)
cmf$  layout Acurlr(:news,:news,:news,:news)
cmf$  layout Acurli(:news,:news,:news,:news)
      double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt)
     &                                :: B0curlr,B0curli,Bcurlr,Bcurli
cmf$  layout B0curlr(:news,:news,:news,:news)
cmf$  layout B0curli(:news,:news,:news,:news)
cmf$  layout Bcurlr(:news,:news,:news,:news)
cmf$  layout Bcurli(:news,:news,:news,:news)
      double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt,mu)
     &                                :: C0curlmur,C0curlmui,Ccurlmur,Ccurlmui
cmf$  layout C0curlmur(:news,:news,:news,:news,:serial)
cmf$  layout C0curlmui(:news,:news,:news,:news,:serial)
cmf$  layout Ccurlmur(:news,:news,:news,:news,:serial)
cmf$  layout Ccurlmui(:news,:news,:news,:news,:serial)

      double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt)   :: qmusq
cmf$  layout qmusq(:news,:news,:news,:news)
      double precision,dimension((npy-1)*npx*npt*(npz-1))        :: qsqam,psqam,tempam
cmf$  layout qsqam(:news)
cmf$  layout psqam(:news)
cmf$  layout tempam(:news)

      double precision,dimension(0:npx-1,0:npy-1,0:npz-1,0:npt-1):: qZ2sq,qZ3sq,tempZ3
cmf$  layout qZ2sq(:news,:news,:news,:news)
cmf$  layout qZ3sq(:news,:news,:news,:news)
cmf$  layout tempZ3(:news,:news,:news,:news)

      double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt,mu):: qmur,qmui
```

```
cmf$  layout qmur(:news,:news,:news,:news,:serial)
cmf$  layout qmui(:news,:news,:news,:news,:serial)

      double precision,dimension(nkappa/2,(npy-1)*npx*npt*(npz-1))
    &                                                :: Br,Bi,B0r,B0i,B1r,B1i
cmf$  layout Br(:serial,:news)
cmf$  layout Bi(:serial,:news)
cmf$  layout B0r(:serial,:news)
cmf$  layout B0i(:serial,:news)
cmf$  layout B1r(:serial,:news)
cmf$  layout B1i(:serial,:news)
      double precision,dimension(nkappa/2,(npy-1)*npx*npt*(npz-1))
    &                                                :: Ar,Ai,A0r,A0i,A1r,A1i
cmf$  layout Ar(:serial,:news)
cmf$  layout Ai(:serial,:news)
cmf$  layout A0r(:serial,:news)
cmf$  layout A0i(:serial,:news)
cmf$  layout A1r(:serial,:news)
cmf$  layout A1i(:serial,:news)

      double precision,dimension(nkappa/2,(npy-1)*npx*npt*(npz-1))  :: Bcr,Bci
cmf$  layout Bcr(:serial,:news)
cmf$  layout Bci(:serial,:news)
      double precision,dimension(nkappa/2,(npy-1)*npx*npt*(npz-1))  :: Acr,Aci
cmf$  layout Acr(:serial,:news)
cmf$  layout Aci(:serial,:news)

      double precision,dimension(nkappa/2,(npy-1)*npx*npt*(npz-1))  :: Mcr,Zcr
cmf$  layout Mcr(:serial,:news)
cmf$  layout Zcr(:serial,:news)

      double precision,dimension(nkappa/2,(npy-1)*npx*npt*(npz-1))  :: normA0
cmf$  layout normA0(:serial,:news)

      double precision,dimension(nkappa/2,(npy-1)*npx*npt*(npz-1))
    &                                           :: A0curlcfgr,A0curlcfgi
      double precision,dimension(nkappa/2,(npy-1)*npx*npt*(npz-1))
    &                                           :: B0curlcfgr,B0curlcfgi
cmf$  layout A0curlcfgr(:serial,:news)
cmf$  layout A0curlcfgi(:serial,:news)
cmf$  layout B0curlcfgr(:serial,:news)
cmf$  layout B0curlcfgi(:serial,:news)

      double precision,dimension(nkappa/2,(npy-1)*npx*npt*(npz-1))
     &                                          :: Acurlcfgaver,Acurlcfgavei
cmf$  layout Acurlcfgaver(:serial,:news)
cmf$  layout Acurlcfgavei(:serial,:news)
      double precision,dimension(nkappa/2,(npy-1)*npx*npt*(npz-1))
     &                                          :: Bcurlcfgaver,Bcurlcfgavei
cmf$  layout Bcurlcfgaver(:serial,:news)
cmf$  layout Bcurlcfgavei(:serial,:news)
```

```
c     the jacknife ananlysis variables.

      double precision,dimension(nkappa/2,0:ncon+2,0:ncon+1)  :: Acurlgr,Acurlgi
cmf$  layout Acurlgr(:serial,:serial,:serial)
cmf$  layout Acurlgi(:serial,:serial,:serial)
      double precision,dimension(nkappa/2,0:ncon+2,0:ncon+1)  :: Bcurlgr,Bcurlgi
cmf$  layout Bcurlgr(:serial,:serial,:serial)
cmf$  layout Bcurlgi(:serial,:serial,:serial)
      double precision,dimension(nkappa/2,0:ncon+2,0:ncon+1)  :: Bcgr,Bcgi
cmf$  layout Bcgr(:serial,:serial,:serial)
cmf$  layout Bcgi(:serial,:serial,:serial)
      double precision,dimension(nkappa/2,0:ncon+2,0:ncon+1)  :: Acgr,Acgi
cmf$  layout Acgr(:serial,:serial,:serial)
cmf$  layout Acgi(:serial,:serial,:serial)
      double precision,dimension(nkappa/2,0:ncon+2,0:ncon+1)  :: Mcgr,Zcgr
cmf$  layout Mcgr(:serial,:serial,:serial)
cmf$  layout Zcgr(:serial,:serial,:serial)

      double precision,dimension(nkappa/2,ncon,(npy-1)*npx*npt*(npz-1))
     &                                                        :: Acurlcfgr,Acurlcfgi
      double precision,dimension(nkappa/2,ncon,(npy-1)*npx*npt*(npz-1))
     &                                                        :: Bcurlcfgr,Bcurlcfgi
cmf$  layout Acurlcfgr(:serial,:serial,:news)
cmf$  layout Acurlcfgi(:serial,:serial,:news)
cmf$  layout Bcurlcfgr(:serial,:serial,:news)
cmf$  layout Bcurlcfgi(:serial,:serial,:news)

c     writing out to disk the bin files.

      double precision,dimension(nkappa/2,0:ncon+2,0:ncon+1,(npy-1)*npx*npt*(npz-1))
     &                                                        :: Mcgrallq,Zcgrallq
cmf$  layout Mcgrallq(:serial,:serial,:serial,:news)
cmf$  layout Zcgrallq(:serial,:serial,:serial,:news)

      double precision,dimension((npy-1)*npx*npt*(npz-1),0:ncon+2)    :: Mextallq
cmf$  layout Mextallq(:news,:serial)

c     output variables for the mass and z function, and the etrapolated values.

      double precision,dimension(1,0:ncon+2)                          :: mcr1,mcr2
cmf$  layout mcr1(:serial,:serial)
cmf$  layout mcr2(:serial,:serial)
      double precision,dimension(2,(npy-1)*npx*npt*(npz-1))           :: Mext,Zext
cmf$  layout Mext(:serial,:news)
cmf$  layout Zext(:serial,:news)
      double precision,dimension(nkappa/2,2,(npy-1)*npx*npt*(npz-1))  :: Mcrq
cmf$  layout Mcrq(:serial,:serial,:news)
      double precision,dimension(nkappa/2,2,(npy-1)*npx*npt*(npz-1))  :: Zcrq
cmf$  layout Zcrq(:serial,:serial,:news)
      double precision,dimension(nkappa/2,2,(npy-1)*npx*npt*(npz-1))  :: Acrq
cmf$  layout Acrq(:serial,:serial,:news)
      double precision,dimension(nkappa/2,2,(npy-1)*npx*npt*(npz-1))  :: Bcrq
```

```
cmf$  layout Bcrq(:serial,:serial,:news)

      double precision                                         :: Acurlsqgr,Acurlsqgi
      double precision                                         :: A0curlsqgr,A0curlsqgi
      double precision                                         :: fgr,fgi,f0gr,f0gi
      double precision                                         :: normg,norm0g
      double precision                                         :: Agr,Agi,A0gr,A0gi
      double precision                                         :: Bgr,Bgi,B0gr,B0gi

      integer                                                  :: icon,jcon,ido
      integer                                                  :: ikappa,iprop,imu
      integer                                                  :: ifile,incount
      integer                                                  :: counter,cfgcount
      integer                                                  :: px,py,pz,pt

      INTERFACE
         subroutine CurliCandBin(Bcurlr,Bcurli,Ccurlmur,Ccurlmui,qpk,xkappa,
     &                           dataf,ikappa,bcx,bcy,bcz,bct)
          implicit none
          include 'latticeSize.h'
          include 'LatParamtrans.h'
          integer,parameter                                    :: nmx=-nx/2+1,npx=nx/2
          integer,parameter                                    :: nmy=-ny/2+1,npy=ny/2
          integer,parameter                                    :: nmz=-nz/2+1,npz=nz/2
          integer,parameter                                    :: nmt=-nt/2+1,npt=nt/2
          integer                                              :: ikappa
          double precision                                     :: bcx,bcy,bcz,bct
          double precision,dimension(nkappa)                   :: xkappa
cmf$  layout xkappa(:serial)
          character(len=100),dimension(10)                     :: dataf
cmf$  layout dataf(:serial)
          character(len=5),dimension(nkappa)                   :: qpk
cmf$  layout qpk(:serial)
          double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt):: Bcurlr,Bcurli
cmf$  layout Bcurlr(:news,:news,:news,:news)
cmf$  layout Bcurli(:news,:news,:news,:news)
          double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt,mu)
     &                                                         :: Ccurlmur,Ccurlmui
cmf$  layout Ccurlmur(:news,:news,:news,:news,:serial)
cmf$  layout Ccurlmui(:news,:news,:news,:news,:serial)
         end subroutine CurliCandBin
         subroutine Momentum(Cmur,Cmui,Ccurlmur,Ccurlmui,Bcurlr,Bcurli)
          implicit none
          include 'latticeSize.h'
          include 'LatParamtrans.h'
          integer,parameter                                    :: nmx=-nx/2+1,npx=nx/2
          integer,parameter                                    :: nmy=-ny/2+1,npy=ny/2
          integer,parameter                                    :: nmz=-nz/2+1,npz=nz/2
          integer,parameter                                    :: nmt=-nt/2+1,npt=nt/2
          double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt):: Bcurlr,Bcurli
cmf$  layout Bcurlr(:news,:news,:news,:news)
cmf$  layout Bcurli(:news,:news,:news,:news)
```

```
          double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt,mu)
     &                                                          :: Ccurlmur,Ccurlmui
cmf$  layout Ccurlmur(:news,:news,:news,:news,:serial)
cmf$  layout Ccurlmui(:news,:news,:news,:news,:serial)
          double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt,mu):: Cmur,Cmui
cmf$  layout Cmur(:news,:news,:news,:news,:serial)
cmf$  layout Cmui(:news,:news,:news,:news,:serial)
         end subroutine Momentum

         subroutine qZ2sqave(qZ2sq,qmusq)
          implicit none
          include 'latticeSize.h'
          include 'LatParamtrans.h'
          integer,parameter                                 :: nmx=-nx/2+1,npx=nx/2
          integer,parameter                                 :: nmy=-ny/2+1,npy=ny/2
          integer,parameter                                 :: nmz=-nz/2+1,npz=nz/2
          integer,parameter                                 :: nmt=-nt/2+1,npt=nt/2
          double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt)    :: qmusq
cmf$  layout qmusq(:news,:news,:news,:news)
          double precision,dimension(0:npx-1,0:npy-1,0:npz-1,0:npt-1)    :: qZ2sq
cmf$  layout qZ2sq(:news,:news,:news,:news)
          end subroutine qZ2sqave
         subroutine qZ3sqave(qZ3sq,qZ2sq)
          implicit none
          include 'latticeSize.h'
          include 'LatParamtrans.h'
          integer,parameter                                 :: nmx=-nx/2+1,npx=nx/2
          integer,parameter                                 :: nmy=-ny/2+1,npy=ny/2
          integer,parameter                                 :: nmz=-nz/2+1,npz=nz/2
          integer,parameter                                 :: nmt=-nt/2+1,npt=nt/2
          double precision,dimension(0:npx-1,0:npy-1,0:npz-1,0:npt-1)    :: qZ3sq
cmf$  layout qZ3sq(:news,:news,:news,:news)
          double precision,dimension(0:npx-1,0:npy-1,0:npz-1,0:npt-1)    :: qZ2sq
cmf$  layout qZ2sq(:news,:news,:news,:news)
         end subroutine qZ3sqave
         subroutine compression(ncount,psqam,amr,ami,AZ3r,AZ3i)
          implicit none
          include 'latticeSize.h'
          include 'LatParamtrans.h'
          integer,parameter                                 :: nmx=-nx/2+1,npx=nx/2
          integer,parameter                                 :: nmy=-ny/2+1,npy=ny/2
          integer,parameter                                 :: nmz=-nz/2+1,npz=nz/2
          integer,parameter                                 :: nmt=-nt/2+1,npt=nt/2
          integer                                           :: ncount
          double precision,dimension((npy-1)*npx*npt*(npz-1)) :: psqam
cmf$  layout psqam(:news)
          double precision,dimension((npy-1)*npx*npt*(npz-1)) :: amr,ami
cmf$  layout amr(:news)
cmf$  layout ami(:news)
          double precision,dimension(0:npx-1,0:npy-1,0:npz-1,0:npt-1):: AZ3r,AZ3i
cmf$  layout AZ3r(:news,:news,:news,:news)
cmf$  layout AZ3i(:news,:news,:news,:news)
```

```
         end subroutine compression
         subroutine AcurlNdBam(ncount,Acurlamr,Acurlami,Bcurlamr,
     &                      Bcurlami,qmur,Ccurlmur,Ccurlmui,Bcurlr,Bcurli)
          implicit none
          include 'latticeSize.h'
          include 'LatParamtrans.h'
          integer,parameter                                  :: nmx=-nx/2+1,npx=nx/2
          integer,parameter                                  :: nmy=-ny/2+1,npy=ny/2
          integer,parameter                                  :: nmz=-nz/2+1,npz=nz/2
          integer,parameter                                  :: nmt=-nt/2+1,npt=nt/2
          integer                                            :: ncount
          double precision,dimension((npy-1)*npx*npt*(npz-1)) :: Acurlamr,Acurlami
          double precision,dimension((npy-1)*npx*npt*(npz-1)) :: Bcurlamr,Bcurlami
cmf$  layout Acurlamr(:news)
cmf$  layout Acurlami(:news)
cmf$  layout Bcurlamr(:news)
cmf$  layout Bcurlami(:news)
          double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt,mu):: qmur
cmf$  layout qmur(:news,:news,:news,:news,:serial)
          double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt):: Bcurlr,Bcurli
cmf$  layout Bcurlr(:news,:news,:news,:news)
cmf$  layout Bcurli(:news,:news,:news,:news)
          double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt,mu)
     &                                                       :: Ccurlmur,Ccurlmui
cmf$  layout Ccurlmur(:news,:news,:news,:news,:serial)
cmf$  layout Ccurlmui(:news,:news,:news,:news,:serial)
         end subroutine AcurlNdBam
         subroutine AandB(ncount,Ar,Ai,Br,Bi,Acurlaver,Acurlavei,
     &                 Bcurlaver,Bcurlavei,qsqam,ikappa)
          implicit none
          include 'latticeSize.h'
          include 'LatParamtrans.h'
          integer,parameter                                  :: nmx=-nx/2+1,npx=nx/2
          integer,parameter                                  :: nmy=-ny/2+1,npy=ny/2
          integer,parameter                                  :: nmz=-nz/2+1,npz=nz/2
          integer,parameter                                  :: nmt=-nt/2+1,npt=nt/2
          integer                                            :: ikappa
          integer                                            :: ncount
          double precision,dimension(nkappa/2,(npy-1)*npx*npt*(npz-1)):: Br,Bi
cmf$  layout Br(:serial,:news)
cmf$  layout Bi(:serial,:news)
          double precision,dimension(nkappa/2,(npy-1)*npx*npt*(npz-1))   :: Ar,Ai
cmf$  layout Ar(:serial,:news)
cmf$  layout Ai(:serial,:news)
          double precision,dimension(nkappa/2,(npy-1)*npx*npt*(npz-1))
     &                                                       :: Bcurlaver,Bcurlavei
cmf$  layout Bcurlaver(:serial,:news)
cmf$  layout Bcurlavei(:serial,:news)
          double precision,dimension(nkappa/2,(npy-1)*npx*npt*(npz-1))
     &                                                       :: Acurlaver,Acurlavei
cmf$  layout Acurlaver(:serial,:news)
cmf$  layout Acurlavei(:serial,:news)
```

```
          double precision,dimension((npy-1)*npx*npt*(npz-1)) :: qsqam
cmf$  layout qsqam(:news)
         end subroutine AandB

         subroutine complement(gin,g)
          implicit none
          include 'latticeSize.h'
          include 'LatParamtrans.h'
          integer,parameter                                  :: ncon=nprop-1
          double precision,dimension(ncon)                   :: gin
cmf$  layout gin(:serial)
          double precision,dimension(0:ncon+2,0:ncon+1)       :: g
cmf$  layout g(:serial,:serial)
         end subroutine complement
         subroutine jack1(g,nlb,nle,lb,le)
          implicit none
          include 'latticeSize.h'
          include 'LatParamtrans.h'
          integer,parameter                                  :: ncon=nprop-1
          integer                                            :: nlb,nle,lb,le,l,icon
          double precision,dimension(nlb:nle,0:ncon+2)        :: g
cmf$  layout g(:serial,:serial)
          end subroutine jack1
          subroutine jack2(g,nlb,nle,lb,le)
           implicit none
           include 'latticeSize.h'
           include 'LatParamtrans.h'
           integer,parameter                                 :: ncon=nprop-1
           integer                                           :: nlb,nle,lb,le
           double precision,dimension(nlb:nle,0:ncon+2,0:ncon+1)  :: g
cmf$  layout g(:serial,:serial,:serial)
          end subroutine jack2

          subroutine extrapjack(m,mcr1,mqk)
           implicit none
           include 'latticeSize.h'
           include 'LatParamtrans.h'
           integer,parameter                                 :: ncon=nprop-1
           double precision,dimension(nkappa/2)              :: mqk
cmf$  layout mqk(:serial)
           double precision,dimension(1,0:ncon+2)            :: mcr1
cmf$  layout mcr1(:serial,:serial)
           double precision,dimension(nkappa/2,0:ncon+2,0:ncon+1):: m
cmf$  layout m(:serial,:serial,:serial)
          end subroutine extrapjack

          subroutine writeMbin(Mcgrallq,Zcgrallq,qpk,ncount,ikappa)
           implicit none
           include 'latticeSize.h'
           include 'LatParamtrans.h'
           integer,parameter                                 :: ncon=nprop-1
           integer,parameter                                 :: nmx=-nx/2+1,npx=nx/2
           integer,parameter                                :: nmy=-ny/2+1,npy=ny/2
           integer,parameter                                :: nmz=-nz/2+1,npz=nz/2
           integer,parameter                                :: nmt=-nt/2+1,npt=nt/2
           integer                                          :: ncount
           integer                                          :: ikappa
           character(len=5),dimension(nkappa)               :: qpk
cmf$  layout qpk(:serial)
         double precision,
     &   dimension(nkappa/2,0:ncon+2,0:ncon+1,(npy-1)*npx*npt*(npz-1))
     &                                                     :: Mcgrallq,Zcgrallq
cmf$  layout Mcgrallq(:serial,:serial,:serial,:news)
cmf$  layout Zcgrallq(:serial,:serial,:serial,:news)
          end subroutine writeMbin

          function strlen(string)
           implicit none
           character*(*) string
           integer                                           :: strlen
           integer                                           :: i, blank
          end function strlen
      end interface

c     start of the execution commands.

c     first read in the datafile ParamTrans.dat

      open(11,file='ParamTrans_Jack.dat',form='formatted',status='old',action='read')

      read (11,*) nkappaQP

      if (nkappaQP /= nkappa) pause 'mismatch in nkappa'
      do ikappa=1,nkappa
         read (11,*) xkappa(ikappa)
         read (11,'(a5)') qpk(ikappa)
      end do

      read (11,*) jx,jy,jz,jt
      read (11,*) bcx,bcy,bcz,bct
      close(11)

c     first start by sreen input.

      write(*,*)
      write(*,*)'Please enter a beta value.'
      read(*,*) beta

       write(*,*) beta

      write(*,*)'Would you to examine fat link action:'
      write(*,*)'                      0:no  fat link'
      write(*,*)'                      1:yes fat link.'
      write(*,*)'                      2:overlap Fermions.'
      read(*,'(i3)') flq

       write(*,*) flq

      write(*,*)'Please enter the core name of the configuration:'
      write(*,*)'Ex: SU3B600S6T18C'
      read(*,'(a80)') corename
      write(*,*)

       write(*,'(a)') corename

      write(*,*)'Please enter the fix name name of the configuration:'
      write(*,*)'Ex: if 0.078 then enter .078. If no fixname leave blank'
      read (*,'(a4)') fixname
      write(*,*)

         write(*,'(a)') fixname

      do iprop=1,nprop

         write(*,*)
         write(*,*)'Which Gauge field configuration are we reading from?'
         write(*,*)'With its associated quark propagator.'
         read (*,'(a3)') confnum(iprop)

         write(*,'(a)') confnum(iprop)

         lastconfig = corename(1:strlen(corename))//confnum(iprop)
         confs(iprop) = lastconfig(1:strlen(lastconfig))//fixname

      end do

      write(*,*) confs(1),confs(2)

      open(1,file='Jacknife.log',status='unknown',position='append')

      write(1,*)
      write(1,*)'=========================================================
     &           ========================'
      write(1,'(a)')'    The mass function for the Wilson Fermions for
     &                   SU(3) Gauge configuration    '
      write(1,*)'=========================================================
     &           ========================'

      write(1,'(4(a,i3))')'lattice size = ',nx,'x',ny,'x',nz,'x',nt
      write(1,'(a,i2)')'The number of **kappa** value that are
     &                  to be considered is =',nkappa
      write(1,'(a,4i3)')  'The source position x, y, z, t is =',jx,jy,jz,jt
      write(1,'(a,4f8.4)')'Boundary condition for fermion fields are=',bcx,bcy,bcz,bct

c     initializing the variables.
```

```
      A0curlcfgr = 0.0d0
      A0curlcfgi = 0.0d0
      B0curlcfgr = 0.0d0
      B0curlcfgi = 0.0d0

      Acurlcfgaver = 0.0d0
      Acurlcfgavei = 0.0d0
      Bcurlcfgaver = 0.0d0
      Bcurlcfgavei = 0.0d0

c     constructing the quark propagator filename.

      if ( beta == 6.00d0 ) then
         kc = 0.15699d0 * uzero
      elseif( beta == 4.60d0 ) then
         kappa = 0.19d0
         if ( flq == 0 .or. flq == 1 ) then
            kc = 0.1390d0
         else if ( flq == 2 ) then
            ksim = 0.1390d0
            kc = 0.125d0
         end if
      elseif( beta == 4.286d0 ) then
         kappa = 0.19d0
         if ( flq == 2 ) then
            ksim = 0.1464d0
            kc = 0.125d0
         end if
      end if

      do ikappa = 1,nkappa/2

         if ( flq == 0 .or. flq == 1 ) then
            mq  = ( 1.0d0/2.0d0 ) * ( ( 1.0d0/xkappa(ikappa) )   - ( 1.0d0 / kc  ) )
            mqk(ikappa) = mq
         elseif ( flq == 2 ) then
c            Zpsi = (2.0d0/uzero)*( (-1.0d0/2.0d0)*( ( 1.0d0/kappa )
c     &                    - ( 1.0d0 / kc  ) ) - 4.0d0*(1.0d0-uzero) )
            Zpsi = ( ( 1.0d0 / ksim ) - ( 1.0d0/kappa ) )
            Z0psi = ( ( 1.0d0 / kc ) - ( 1.0d0/kappa ) )
            mq = xkappa(ikappa) * Zpsi
            mqk(ikappa) = mq
         end if

         write(*,*)
         write(1,'(a,f25.16)')    'the input kappa in the quark propagator code is
     &                             k = ',xkappa(ikappa)
         write(1,'(a,f25.16)')    'The Desy corresponding kappa value,
     &                             k/uzero = ',xkappa(ikappa)/uzero
         write(1,'(a,f25.16)')    'The wilson coefficient r,
     &                             r = ',rwil
```

```
         write(1,'(a,f25.16)')    'the corrected kappa critical from Kc=.125 is
     &                             kc = ',kc
         write(1,'(a,f25.16)')    'the Desy kappa critical is
     &                             kc_Desy = kc / uzero = ',kc/uzero
         write(1,'(a,f25.16,x,a)')'Lattice bare q mass
     &                             (1/2a)*(1/k-1/kc)*197.327(aMeV),
     &                             mq = ',mq*(197.33270d0),'(MeV)*a'

c     get the tree level curli B.

         lastconfig = confs(1)
         if ( flq == 0 .or. flq == 1 ) then
            qp(ikappa) = lastconfig(1:strlen(lastconfig))//
     &                   '.fix'//qpk(ikappa+nkappa/2)
         elseif ( flq == 2 ) then
            qp(ikappa) = lastconfig(1:strlen(lastconfig))//'.'//qpk(ikappa+nkappa/2)
         end if

         write(*,*)
         write(*,'(a,3x,a100)')'We are now reading from:',qp(ikappa+nkappa/2)
         write(1,'(a,3x,a100)')'We are now reading from:',qp(ikappa+nkappa/2)

c     checkin the existence of the propagators.

         quarkprop = 'ptQ'//qp(ikappa)

         inquire(file=quarkprop,exist=uexists)

         if(uexists) then
            write(*,'(a,2x,a40,x,a)')'the quark propagator:',quarkprop,'exists'
            write(*,'(a,2x,a40)')'we are now proceeding with the reading of:',
     &                            quarkprop
         elseif(.not. uexists ) then
            write(*,'(a,2x,a40,x,a)')'the quark propagator:',
     &                                quarkprop,'does not exists'
            stop
         end if

c     now get the tree level culi C and B from disk

         datafile = 'Bcurl'
         datafile = datafile(1:strlen(datafile))//corename
         datafile = datafile(1:strlen(datafile))//confnum(1)
         dataf(1) = datafile(1:strlen(datafile))//fixname

         datafile = 'Ccurl'
         datafile = datafile(1:strlen(datafile))//corename
         datafile = datafile(1:strlen(datafile))//confnum(1)
         dataf(2) = datafile(1:strlen(datafile))//fixname

         call CurliCandBin(B0curlr,B0curli,C0curlmur,C0curlmui,qpk,
     &                     xkappa,dataf,ikappa+nkappa/2,bcx,bcy,bcz,bct)
```

```
         if ( flq == 2 ) then
            C0curlmur = C0curlmur * ( Z0psi**(-1) )
            C0curlmui = C0curlmui * ( Z0psi**(-1) )
            B0curlr   = B0curlr   * ( Z0psi**(-1) )
            B0curli   = B0curli   * ( Z0psi**(-1) )
         end if

c     now constrcutig the lattice momentum q_\mu

         call Momentum(qmur,qmui,C0curlmur,C0curlmui,B0curlr,B0curli)

         qmusq = 0.0d0
         do imu=1,mu
            qmusq(:,:,:,:) = qmusq(:,:,:,:) + ( qmur(:,:,:,:,imu)**2 )
         end do

         call qZ2sqave(qZ2sq,qmusq)
         call qZ3sqave(qZ3sq,qZ2sq)

         tempZ3 = 0.0d0
         tempam = 0.0d0

         call compression(ncount,psqam,qsqam,tempam,qZ3sq,tempZ3)

c     now constraucting the Z2 and Z3 averaged curli A and B curli

         call AcurlNdBam(ncount,A0curlamr,A0curlami,B0curlamr,B0curlami,
     &                   qmur,C0curlmur,C0curlmui,B0curlr,B0curli)

         A0curlcfgr(ikappa, :) = A0curlamr(:)
         A0curlcfgi(ikappa, :) = A0curlami(:)
         B0curlcfgr(ikappa, :) = B0curlamr(:)
         B0curlcfgi(ikappa, :) = B0curlami(:)

         write(*,*) ncount

c     now repeating the same procedure for all the configurations.

         cfgcount = 0
         do iprop=2,nprop

            cfgcount = cfgcount + 1
            icon = iprop - 1

            datafile = 'Bcurl'
            datafile = datafile(1:strlen(datafile))//corename
            datafile = datafile(1:strlen(datafile))//confnum(iprop)
            dataf(1) = datafile(1:strlen(datafile))//fixname

            datafile = 'Ccurl'
            datafile = datafile(1:strlen(datafile))//corename
```

```
          datafile = datafile(1:strlen(datafile))//confnum(iprop)
          dataf(2) = datafile(1:strlen(datafile))//fixname

          call CurliCandBin(Bcurlr,Bcurli,Ccurlmur,Ccurlmui,qpk,
     &                        xkappa,dataf,ikappa,bcx,bcy,bcz,bct)

          if ( flq == 2 ) then
             Ccurlmur = Ccurlmur * ( Zpsi**(-1) )
             Ccurlmui = Ccurlmui * ( Zpsi**(-1) )
             Bcurlr   = Bcurlr   * ( Zpsi**(-1) )
             Bcurli   = Bcurli   * ( Zpsi**(-1) )
          end if

          call AcurlNdBam(ncount,Acurlamr,Acurlami,Bcurlamr,Bcurlami,
     &                     qmur,Ccurlmur,Ccurlmui,Bcurlr,Bcurli)

          Acurlcfgr(ikappa, icon, :) = Acurlamr(:)
          Acurlcfgi(ikappa, icon, :) = Acurlami(:)
          Bcurlcfgr(ikappa, icon, :) = Bcurlamr(:)
          Bcurlcfgi(ikappa, icon, :) = Bcurlami(:)

          Acurlcfgaver(ikappa,:) = Acurlcfgaver(ikappa,:) +
     &                             Acurlcfgr(ikappa, icon, :)
          Acurlcfgavei(ikappa,:) = Acurlcfgavei(ikappa,:) +
     &                             Acurlcfgi(ikappa, icon, :)
          Bcurlcfgaver(ikappa,:) = Bcurlcfgaver(ikappa,:) +
     &                             Bcurlcfgr(ikappa, icon, :)
          Bcurlcfgavei(ikappa,:) = Bcurlcfgavei(ikappa,:) +
     &                             Bcurlcfgi(ikappa, icon, :)

          write(*,*) ncount

       end do

       Acurlcfgaver(ikappa,:) = Acurlcfgaver(ikappa,:) / cfgcount
       Acurlcfgavei(ikappa,:) = Acurlcfgavei(ikappa,:) / cfgcount
       Bcurlcfgaver(ikappa,:) = Bcurlcfgaver(ikappa,:) / cfgcount
       Bcurlcfgavei(ikappa,:) = Bcurlcfgavei(ikappa,:) / cfgcount

c     opening the ouput files first.

c     opening the ouput files first.

       dataf(1) = 'Bcor.'
       dataf(2) = 'Acor.'
       dataf(3) = 'Mcor.'
       dataf(4) = 'Zcor.'

       do ifile=1,4
          datafile = dataf(ifile)
          datafile = datafile(1:strlen(datafile))//qpk(ikappa)//'.dat'
          open(20+ifile+ikappa,file=datafile,status='unknown',position='append')
```

```
         end do

c     now constructing the A and B. then performing the correction.

         call AandB(ncount,  Ar, Ai , Br , Bi , Acurlcfgaver, Acurlcfgavei,
     &              Bcurlcfgaver, Bcurlcfgavei, qsqam,ikappa)
         if ( flq == 0 .or. flq == 1 ) then
            call AandB(ncount, A0r, A0i, B0r, B0i, A0curlcfgr  , A0curlcfgi  ,
     &                 B0curlcfgr  , B0curlcfgi  , qsqam,ikappa)
         else if ( flq == 2 ) then
            call AandB(ncount, A1r, A1i, B1r, B1i, A0curlcfgr  , A0curlcfgi  ,
     &                 B0curlcfgr  , B0curlcfgi  , qsqam,ikappa)
c           write(100,'(f21.15)') A1r(ikappa,:)
c           write(101,'(f21.15)') A1i(ikappa,:)
c           write(102,'(f21.15)') B1r(ikappa,:)
c           write(103,'(f21.15)') B1i(ikappa,:)
            A0r = 1.0d0
            A0i = 0.0d0
            B0r = 1.0d0
            B0i = 0.0d0
         end if

c     now performing the tree level correction.

c     the tree level correction for the B(p) function.

         Bcr(ikappa,:) = 0.0d0
         Bci(ikappa,:) = 0.0d0

         normA0(ikappa,1:ncount) =
     &                    ( B0r(ikappa,1:ncount)**2 + B0i(ikappa,1:ncount)**2 )

         Bcr(ikappa,1:ncount) = (   Br(ikappa,1:ncount) * B0r(ikappa,1:ncount) +
     &                              Bi(ikappa,1:ncount) * B0i(ikappa,1:ncount) )
     &                              / normA0(ikappa,1:ncount)
         Bci(ikappa,1:ncount) = ( - Br(ikappa,1:ncount) * B0i(ikappa,1:ncount) +
     &                              Bi(ikappa,1:ncount) * B0r(ikappa,1:ncount) )
     &                              / normA0(ikappa,1:ncount)

         if ( flq == 0 .or. flq == 1 ) then
            Bcr(ikappa,1:ncount) = Bcr(ikappa,1:ncount) * mq
            Bci(ikappa,1:ncount) = Bci(ikappa,1:ncount) * mq
         end if

         do incount=1,ncount
            write(21+ikappa,'(x,f25.16,2(2x,f25.16))') psqam(incount),
     &                                 qsqam(incount),Bcr(ikappa,incount)
         end do

c     tree level correction for the C_{\mu}(p)

         Acr(ikappa,:) = 0.0d0
```

```
            Aci(ikappa,:) = 0.0d0

            normA0(ikappa,1:ncount) = A0r(ikappa,1:ncount)**2 + A0i(ikappa,1:ncount)**2

            Acr(ikappa,1:ncount) = (   Ar(ikappa,1:ncount) * A0r(ikappa,1:ncount) +
     &                                 Ai(ikappa,1:ncount) * A0i(ikappa,1:ncount) )
     &                                 / normA0(ikappa,1:ncount)
            Aci(ikappa,1:ncount) = ( - Ar(ikappa,1:ncount) * A0i(ikappa,1:ncount) +
     &                                 Ai(ikappa,1:ncount) * A0r(ikappa,1:ncount) )
     &                                 / normA0(ikappa,1:ncount)

            Mcr(ikappa,1:ncount) = Bcr(ikappa,1:ncount) / Acr(ikappa,1:ncount)
            Zcr(ikappa,1:ncount) = 1.0d0 / Acr(ikappa,1:ncount)

            do incount=1,ncount
               write(22+ikappa,'(x,f25.16,2(2x,f25.16))') psqam(incount),qsqam(incount),
     &                                                    Acr(ikappa,incount)
               write(23+ikappa,'(x,f25.16,2(2x,f25.16))') psqam(incount),qsqam(incount),
     &                                                    Mcr(ikappa,incount)
               write(24+ikappa,'(x,f25.16,2(2x,f25.16))') psqam(incount),qsqam(incount),
     &                                                    Zcr(ikappa,incount)
               write(104+ikappa,'(x,f25.16,2(2x,f25.16))') psqam(incount),qsqam(incount),
     &                                                     Bcurlcfgaver(ikappa,incount)
            end do

            do ifile=1,4
               close(20+ifile+ikappa)
            end do

         end do

c        now calculating the errors on the ensemble averaged functions.

         Acurlgr = 0.0d0
         Acurlgi = 0.0d0
         Bcurlgr = 0.0d0
         Bcurlgi = 0.0d0

         Bcgr = 0.0d0
         Bcgi = 0.0d0
         Acgr = 0.0d0
         Acgi = 0.0d0
         Mcgr = 0.0d0
         Zcgr = 0.0d0

         do incount=1,ncount

c        first get the complement sets for the variables, for each kappa and momentum
c        value.

            do ikappa=1,nkappa/2
               call complement(Acurlcfgr(ikappa,:,incount),Acurlgr(ikappa,:,:))
               call complement(Acurlcfgi(ikappa,:,incount),Acurlgi(ikappa,:,:))

               call complement(Bcurlcfgr(ikappa,:,incount),Bcurlgr(ikappa,:,:))
               call complement(Bcurlcfgi(ikappa,:,incount),Bcurlgi(ikappa,:,:))

               do icon=0,ncon
                  do jcon=0,ncon

                     ido = 0

                     if( icon == jcon .and. icon /= 0 ) then
                        ido = 1
                     elseif ( icon == 0 .and. jcon /= 0 ) then
                        ido = 1
                     end if

                     if ( ido /= 1 ) then

                        Acurlsqgr =   ( Acurlgr(ikappa,icon,jcon)**2 -
        &                               Acurlgi(ikappa,icon,jcon)**2 ) * qsqam(incount)
                        Acurlsqgi = ( 2.0d0 * Acurlgr(ikappa,icon,jcon) *
        &                             Acurlgi(ikappa,icon,jcon) ) * qsqam(incount)

                        A0curlsqgr =   ( A0curlcfgr(ikappa,incount)**2 -
        &                                A0curlcfgi(ikappa,incount)**2 ) * qsqam(incount)
                        A0curlsqgi = ( 2.0d0 * A0curlcfgr(ikappa,incount) *
        &                              A0curlcfgi(ikappa,incount) ) * qsqam(incount)

   c     Bsqcurlaver = Bcurlaver**2 - Bcurlavei**2
   c     Bsqcurlavei = 2.0d0 * Bcurlaver * Bcurlavei

                        fgr = Acurlsqgr + Bcurlgr(ikappa,icon,jcon)**2 -
        &                     Bcurlgi(ikappa,icon,jcon)**2
                        fgi = Acurlsqgi + 2.0d0 * Bcurlgr(ikappa,icon,jcon) *
        &                     Bcurlgi(ikappa,icon,jcon)

                        f0gr = A0curlsqgr + B0curlcfgr(ikappa,incount)**2 -
        &                      B0curlcfgi(ikappa,incount)**2
                        f0gi = A0curlsqgi + 2.0d0 * B0curlcfgr(ikappa,incount) *
        &                      B0curlcfgi(ikappa,incount)

                        normg = fgr**2 + fgi**2
                        norm0g = f0gr**2 + f0gi**2

                        Agr = (   Acurlgr(ikappa,icon,jcon) * fgr +
        &                         Acurlgi(ikappa,icon,jcon) * fgi ) / normg
                        Agi = ( - Acurlgr(ikappa,icon,jcon) * fgi +
        &                         Acurlgi(ikappa,icon,jcon) * fgr ) / normg
                        Bgr = (   Bcurlgr(ikappa,icon,jcon) * fgr +
        &                         Bcurlgi(ikappa,icon,jcon) * fgi ) / normg
                        Bgi = ( - Bcurlgr(ikappa,icon,jcon) * fgi +
        &                          Bcurlgi(ikappa,icon,jcon) * fgr ) / normg

                        if ( flq == 0 .or. flq == 1 ) then
                           A0gr = (   A0curlcfgr(ikappa,incount) * f0gr +
        &                            A0curlcfgi(ikappa,incount) * f0gi ) / norm0g
                           A0gi = ( - A0curlcfgr(ikappa,incount) * f0gi +
        &                            A0curlcfgi(ikappa,incount) * f0gr ) / norm0g
                           B0gr = (   B0curlcfgr(ikappa,incount) * f0gr +
        &                            B0curlcfgi(ikappa,incount) * f0gi ) / norm0g
                           B0gi = ( - B0curlcfgr(ikappa,incount) * f0gi +
        &                            B0curlcfgi(ikappa,incount) * f0gr ) / norm0g
                        else if ( flq == 2 ) then
                           A0gr = 1.0d0
                           A0gi = 0.0d0
                           B0gr = 1.0d0
                           B0gi = 0.0d0
                        end if

   c     the tree level correction for the B(p) function.

                        normg = B0gr**2 + B0gi**2

                        if ( flq == 0 .or. flq == 1 ) then
                           Bcgr(ikappa,icon,jcon) = ( (   Bgr * B0gr + Bgi * B0gi )
        &                                          / normg ) * mqk(ikappa)
                           Bcgi(ikappa,icon,jcon) = ( ( - Bgr * B0gi + Bgi * B0gr )
        &                                          / normg ) * mqk(ikappa)
                        else if ( flq == 2 ) then
                           Bcgr(ikappa,icon,jcon) = ( (   Bgr * B0gr + Bgi * B0gi )
        &                                         / normg )
                           Bcgi(ikappa,icon,jcon) = ( ( - Bgr * B0gi + Bgi * B0gr )
        &                                         / normg )
                        end if

   c     tree level correction for the C_{\mu}(p)

                        normg = A0gr**2 + A0gi**2

                        Acgr(ikappa,icon,jcon) = (   Agr * A0gr + Agi * A0gi ) / normg
                        Acgi(ikappa,icon,jcon) = ( - Agr * A0gi + Agi * A0gr ) / normg

                        Mcgr(ikappa,icon,jcon) = Bcgr(ikappa,icon,jcon) /
        &                                       Acgr(ikappa,icon,jcon)
                        Zcgr(ikappa,icon,jcon) = 1.0d0 / Acgr(ikappa,icon,jcon)

                     end if

                  end do
               end do

            end do
```

```
c     storing the complement sets for the mass function into one
c     array for all momemtum values

         Mcgrallq(:,:,:,incount) = Mcgr(:,:,:)
         Zcgrallq(:,:,:,incount) = Zcgr(:,:,:)

c     here get the Jacknife methods Jack21

         call jack2(Acgr,1,nkappa/2,1,nkappa/2)
         call jack2(Bcgr,1,nkappa/2,1,nkappa/2)
         call jack2(Mcgr,1,nkappa/2,1,nkappa/2)
         call jack2(Zcgr,1,nkappa/2,1,nkappa/2)

         call jack1(Acgr(:,0:ncon+2,0),1,nkappa/2,1,nkappa/2)
         call jack1(Bcgr(:,0:ncon+2,0),1,nkappa/2,1,nkappa/2)
         call jack1(Mcgr(:,0:ncon+2,0),1,nkappa/2,1,nkappa/2)
         call jack1(Zcgr(:,0:ncon+2,0),1,nkappa/2,1,nkappa/2)

         Acrq(:,0,incount) = Acgr(:,0,0)
         Acrq(:,1,incount) = Acgr(:,ncon+1,0)
         Bcrq(:,0,incount) = Bcgr(:,0,0)
         Bcrq(:,1,incount) = Bcgr(:,ncon+1,0)

         Mcrq(:,0,incount) = Mcgr(:,0,0)
         Mcrq(:,1,incount) = Mcgr(:,ncon+1,0)
         Zcrq(:,0,incount) = Zcgr(:,0,0)
         Zcrq(:,1,incount) = Zcgr(:,ncon+1,0)

         call extrapjack(Mcgr,mcr1,mqk)
         Mext(0,incount) = mcr1(1,0)
         Mext(1,incount) = mcr1(1,ncon+1)

         Mextallq(incount,0:ncon+2) = mcr1(1,0:ncon+2)

         call extrapjack(Zcgr,mcr2,mqk)
         Zext(0,incount) = mcr2(1,0)
         Zext(1,incount) = mcr2(1,ncon+1)

      end do

c     now printing out the values for all the momentum values

      dataf(7)  = 'Mcor_err.'
      dataf(8)  = 'Zcor_err.'
      dataf(9)  = 'Acor_err.'
      dataf(10) = 'Bcor_err.'

      do ikappa=1,nkappa/2

         do ifile=7,10
            datafile = dataf(ifile)
            datafile = datafile(1:strlen(datafile))//qpk(ikappa)//'.dat'
            open(1000+ifile+ikappa,file=datafile,status='unknown',
     &           position='append',action='write')
         end do

         do incount=1,ncount

            write(1007+ikappa,'(x,f25.16,4(2x,f25.16))')
     &           psqam(incount),qsqam(incount),Mcr(ikappa,incount),
     &           Mcrq(ikappa,0,incount),Mcrq(ikappa,1,incount)
            write(1008+ikappa,'(x,f25.16,4(2x,f25.16))')
     &           psqam(incount),qsqam(incount),Zcr(ikappa,incount),
     &           Zcrq(ikappa,0,incount),Zcrq(ikappa,1,incount)
            write(1009+ikappa,'(x,f25.16,4(2x,f25.16))')
     &           psqam(incount),qsqam(incount),Acr(ikappa,incount),
     &           Acrq(ikappa,0,incount),Acrq(ikappa,1,incount)
            write(1010+ikappa,'(x,f25.16,4(2x,f25.16))')
     &           psqam(incount),qsqam(incount),Bcr(ikappa,incount),
     &           Bcrq(ikappa,0,incount),Bcrq(ikappa,1,incount)

         end do

         do ifile=7,10
            close(1000+ifile+ikappa)
         end do

         call writeMbin(Mcgrallq,Zcgrallq,qpk,ncount,ikappa)

      end do

      datafile = 'Mextra.'
      datafile = datafile(1:strlen(datafile))//'dat'
      open(1011,file=datafile,status='unknown',position='append',action='write')
      datafile = 'Zextra.'
      datafile = datafile(1:strlen(datafile))//'dat'
      open(1012,file=datafile,status='unknown',position='append',action='write')

      datafile = 'Mextra_bin.'
      datafile = datafile(1:strlen(datafile))//'.dat'
      open(1013,file=datafile,form='unformatted',status='replace',action='write')

      do icon=0,ncon+2
         write(1013) Mextallq(1:ncount,icon)
      end do
      close(1013)

      do incount=1,ncount
         write(1011,'(x,f25.16,3(2x,f25.16))') psqam(incount),qsqam(incount),
     &                                         Mext(0,incount),Mext(1,incount)
         write(1012,'(x,f25.16,3(2x,f25.16))') psqam(incount),qsqam(incount),
     &                                         Zext(0,incount),Zext(1,incount)
      end do
      close(1011)
      close(1012)

      close(1)

      end program JackMassAndZ
c
c     cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Author: Frederic D.R. Bonnet; date: 18th of June 2001.
c     subroutine to calculate the uncorrected C_\mu(p) and and B(p).
c
      subroutine Z2averaging(AZ2r,AZ2i,Ar,Ai)
      implicit none
      include 'latticeSize.h'
      include 'LatParamtrans.h'

c     global variables.

      integer,parameter                                       :: nmx=-nx/2+1,npx=nx/2
      integer,parameter                                       :: nmy=-ny/2+1,npy=ny/2
      integer,parameter                                       :: nmz=-nz/2+1,npz=nz/2
      integer,parameter                                       :: nmt=-nt/2+1,npt=nt/2

      double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt)    :: Ar,Ai
cmf$  layout Ar(:news,:news,:news,:news)
cmf$  layout Ai(:news,:news,:news,:news)
      double precision,dimension(0:npx-1,0:npy-1,0:npz-1,0:npt-1)    :: AZ2r,AZ2i
cmf$  layout AZ2r(:news,:news,:news,:news)
cmf$  layout AZ2i(:news,:news,:news,:news)

c     local variables.

      integer                                                 :: px,py,pz,pt

c     start of the execution commands.

c     first calculating the CurcC_{\mu}=(i/4)Tr[\gamma_\mu*S(p)]

      AZ2r = 0.0d0
      AZ2i = 0.0d0

      do pt=0,npt-1
         do pz=0,npz-1
            do py=0,npy-1
               do px=0,npx-1
                  AZ2r(px,py,pz,pt) = Ar( px, py, pz, pt+1) + Ar(-px, py, pz, pt+1) +
     &                                Ar( px,-py, pz, pt+1) + Ar(-px,-py, pz, pt+1) +
     &                                Ar( px, py,-pz, pt+1) + Ar(-px, py,-pz, pt+1) +
     &                                Ar( px,-py,-pz, pt+1) + Ar(-px,-py,-pz, pt+1) +
     &                                Ar( px, py, pz,  -pt) + Ar(-px, py, pz,  -pt) +
     &                                Ar( px,-py, pz,  -pt) + Ar(-px,-py, pz,  -pt) +
     &                                Ar( px, py,-pz,  -pt) + Ar(-px, py,-pz,  -pt) +
     &                                Ar( px,-py,-pz,  -pt) + Ar(-px,-py,-pz,  -pt)
                AZ2i(px,py,pz,pt) = Ai( px, py, pz, pt+1) + Ai(-px, py, pz, pt+1) +
     &                                Ai( px,-py, pz, pt+1) + Ai(-px,-py, pz, pt+1) +
     &                                Ai( px, py,-pz, pt+1) + Ai(-px, py,-pz, pt+1) +
     &                                Ai( px,-py,-pz, pt+1) + Ai(-px,-py,-pz, pt+1) +
     &                                Ai( px, py, pz,  -pt) + Ai(-px, py, pz,  -pt) +
     &                                Ai( px,-py, pz,  -pt) + Ai(-px,-py, pz,  -pt) +
     &                                Ai( px, py,-pz,  -pt) + Ai(-px, py,-pz,  -pt) +
     &                                Ai( px,-py,-pz,  -pt) + Ai(-px,-py,-pz,  -pt)
               end do
            end do
         end do
      end do

      AZ2r = (1.0d0/16.0d0) * AZ2r
      AZ2i = (1.0d0/16.0d0) * AZ2i

      return
      end subroutine Z2averaging
c
c     cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Author: Frederic D.R. Bonnet; date: 18th of June 2001.
c     subroutine to calculate the uncorrected C_\mu(p) and and B(p).
c
      subroutine Z3averaging(AZ3r,AZ3i,AZ2r,AZ2i)
      implicit none
      include 'latticeSize.h'
      include 'LatParamtrans.h'

c     global variables.

      integer,parameter                                      :: nmx=-nx/2+1,npx=nx/2
      integer,parameter                                      :: nmy=-ny/2+1,npy=ny/2
      integer,parameter                                      :: nmz=-nz/2+1,npz=nz/2
      integer,parameter                                      :: nmt=-nt/2+1,npt=nt/2

      double precision,dimension(0:npx-1,0:npy-1,0:npz-1,0:npt-1)    :: AZ3r,AZ3i
cmf$  layout AZ3r(:news,:news,:news,:news)
cmf$  layout AZ3i(:news,:news,:news,:news)
      double precision,dimension(0:npx-1,0:npy-1,0:npz-1,0:npt-1)    :: AZ2r,AZ2i
cmf$  layout AZ2r(:news,:news,:news,:news)
cmf$  layout AZ2i(:news,:news,:news,:news)

c     local variables.

      integer                                                :: px,py,pz,pt

c     start of the execution commands.

c     Calculating the Z3 averaging for the given array.

      AZ3r = 0.0d0
      AZ3i = 0.0d0

      do px = 0,npx-1
         do py = 0,px
            do pz=0,py
               do pt=0,npt-1
                  AZ3r(px,py,pz,pt) =  AZ2r(px,py,pz,pt) + AZ2r(px,pz,py,pt) +
     &                                 AZ2r(pz,px,py,pt) + AZ2r(pz,py,px,pt) +
     &                                 AZ2r(py,pz,px,pt) + AZ2r(py,px,pz,pt)
                  AZ3i(px,py,pz,pt) =  AZ2i(px,py,pz,pt) + AZ2i(px,pz,py,pt) +
     &                                 AZ2i(pz,px,py,pt) + AZ2i(pz,py,px,pt) +
     &                                 AZ2i(py,pz,px,pt) + AZ2i(py,px,pz,pt)
               end do
            end do
         end do
      end do

      AZ3r = AZ3r / 6.0d0
      AZ3i = AZ3i / 6.0d0

      return
      end subroutine Z3averaging
c
c     ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Author: Frederic D.R. Bonnet; date: 19th of June 2001.
c     subroutine to calculate the uncorrected C_\mu(p) and and B(p).
c
      subroutine qZ3sqave(qZ3sq,qZ2sq)
      implicit none
      include 'latticeSize.h'
      include 'LatParamtrans.h'

c     global variables.

      integer,parameter                                       :: nmx=-nx/2+1,npx=nx/2
      integer,parameter                                       :: nmy=-ny/2+1,npy=ny/2
      integer,parameter                                       :: nmz=-nz/2+1,npz=nz/2
      integer,parameter                                       :: nmt=-nt/2+1,npt=nt/2

      double precision,dimension(0:npx-1,0:npy-1,0:npz-1,0:npt-1)    :: qZ3sq
cmf$  layout qZ3sq(:news,:news,:news,:news)
      double precision,dimension(0:npx-1,0:npy-1,0:npz-1,0:npt-1)    :: qZ2sq
cmf$  layout qZ2sq(:news,:news,:news,:news)

c     local variables.

      integer                                                 :: px,py,pz,pt

c     start of the execution commands.

c     Calculating the Z3 averaging for the given array.
```

```
      qZ3sq(:,:,:,:) = 0.0d0

      do px = 0,npx-1
         do py = 0,px
            do pz=0,py
               do pt=0,npt-1
                  qZ3sq(px,py,pz,pt) =  qZ2sq(px,py,pz,pt) + qZ2sq(px,pz,py,pt) +
     &                                  qZ2sq(pz,px,py,pt) + qZ2sq(pz,py,px,pt) +
     &                                  qZ2sq(py,pz,px,pt) + qZ2sq(py,px,pz,pt)
               end do
            end do
         end do
      end do

      qZ3sq(:,:,:,:) = qZ3sq(:,:,:,:) / 6.0d0

      return
      end subroutine qZ3sqave
c
c     cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Author: Frederic D.R. Bonnet; date: 19th of June 2001.
c     subroutine to calculate the uncorrected C_\mu(p) and and B(p).
c
      subroutine qZ2sqave(qZ2sq,qmusq)
      implicit none
      include 'latticeSize.h'
      include 'LatParamtrans.h'

c     global variables.

      integer,parameter                                      :: nmx=-nx/2+1,npx=nx/2
      integer,parameter                                      :: nmy=-ny/2+1,npy=ny/2
      integer,parameter                                      :: nmz=-nz/2+1,npz=nz/2
      integer,parameter                                      :: nmt=-nt/2+1,npt=nt/2

      double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt)    :: qmusq
cmf$  layout qmusq(:news,:news,:news,:news)
      double precision,dimension(0:npx-1,0:npy-1,0:npz-1,0:npt-1)    :: qZ2sq
cmf$  layout qZ2sq(:news,:news,:news,:news)

c     local variables.

      integer                                                :: px,py,pz,pt

c     start of the execution commands.

c     calculating the Z2 averaging for the qmu arrays.

      qZ2sq(:,:,:,:) = 0.0d0

      do pt=0,npt-1
         do pz=0,npz-1
```

```
             do py=0,npy-1
                do px=0,npx-1
                   qZ2sq(px,py,pz,pt) = qmusq( px, py, pz, pt+1) +
     &                                  qmusq(-px, py, pz, pt+1) +
     &                                  qmusq( px,-py, pz, pt+1) +
     &                                  qmusq(-px,-py, pz, pt+1) +
     &                                  qmusq( px, py,-pz, pt+1) +
     &                                  qmusq(-px, py,-pz, pt+1) +
     &                                  qmusq( px,-py,-pz, pt+1) +
     &                                  qmusq(-px,-py,-pz, pt+1) +
     &                                  qmusq( px, py, pz,  -pt) +
     &                                  qmusq(-px, py, pz,  -pt) +
     &                                  qmusq( px,-py, pz,  -pt) +
     &                                  qmusq(-px,-py, pz,  -pt) +
     &                                  qmusq( px, py,-pz,  -pt) +
     &                                  qmusq(-px, py,-pz,  -pt) +
     &                                  qmusq( px,-py,-pz,  -pt) +
     &                                  qmusq(-px,-py,-pz,  -pt)
                end do
             end do
          end do
       end do

       qZ2sq(:,:,:,:) = (1.0d0/16.0d0) * qZ2sq(:,:,:,:)

       return
       end subroutine qZ2sqave
c
c      cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c      Author: Frederic D.R. Bonnet; date: 2nd of February 2001.
c      subroutine to calculate the uncorrected C_\mu(p) and and B(p).
c
       subroutine Momentum(Cmur,Cmui,Ccurlmur,Ccurlmui,Bcurlr,Bcurli)
       implicit none
       include 'latticeSize.h'
       include 'LatParamtrans.h'

c      global variables.

       integer,parameter                                       :: nmx=-nx/2+1,npx=nx/2
       integer,parameter                                       :: nmy=-ny/2+1,npy=ny/2
       integer,parameter                                       :: nmz=-nz/2+1,npz=nz/2
       integer,parameter                                       :: nmt=-nt/2+1,npt=nt/2

       double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt)   :: Bcurlr,Bcurli
cmf$   layout Bcurlr(:news,:news,:news,:news)
cmf$   layout Bcurli(:news,:news,:news,:news)
       double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt,mu)
     &                                                         :: Ccurlmur,Ccurlmui
cmf$   layout Ccurlmur(:news,:news,:news,:news,:serial)
cmf$   layout Ccurlmui(:news,:news,:news,:news,:serial)
       double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt,mu):: Cmur,Cmui
```

```
cmf$  layout Cmur(:news,:news,:news,:news,:serial)
cmf$  layout Cmui(:news,:news,:news,:news,:serial)

c     local variables.

      double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt):: norm,fr,fi
cmf$  layout norm(:news,:news,:news,:news)
cmf$  layout fr(:news,:news,:news,:news)
cmf$  layout fi(:news,:news,:news,:news)

      double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt):: Ccurlsqr,Ccurlsqi
cmf$  layout Ccurlsqr(:news,:news,:news,:news)
cmf$  layout Ccurlsqi(:news,:news,:news,:news)

      integer                                                 :: id,imu

c     start of the execution commands.

c     start of the execution commands.

      Cmur = 0.0d0
      Cmui = 0.0d0

c     now calculating the normal C_{\mu}(p) as defined by 1.
c     similarly for the normal B(p) as defined by 2.

      Ccurlsqr = 0.0d0
      Ccurlsqi = 0.0d0

      do imu=1,mu

         Ccurlsqr = Ccurlsqr + ( Ccurlmur(:,:,:,:,imu)**2 - Ccurlmui(:,:,:,:,imu)**2 )
         Ccurlsqi = Ccurlsqi +
     &              ( 2.0d0 * Ccurlmur(:,:,:,:,imu) * Ccurlmui(:,:,:,:,imu) )

      end do

c      Bsqcurlr = Bcurlr**2 - Bcurli**2
c      Bsqcurli = 2.0d0 * Bcurlr * Bcurli

      fr = Ccurlsqr + Bcurlr**2 - Bcurli**2
      fi = Ccurlsqi + 2.0d0 * Bcurlr * Bcurli

      norm = fr**2 + fi**2

      do imu=1,mu

         Cmur(:,:,:,:,imu) = (   Ccurlmur(:,:,:,:,imu) * fr +
     &                           Ccurlmui(:,:,:,:,imu) * fi ) / norm
         Cmui(:,:,:,:,imu) = ( - Ccurlmur(:,:,:,:,imu) * fi +
     &                           Ccurlmui(:,:,:,:,imu) * fr ) / norm
```

```
      end do

      return
      end subroutine Momentum
c
c     cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Author: Frederic D.R. Bonnet; date: 20th of June 2001.
c     subroutine to get the compressed write curli A and B on a
c     configuration basis for each kappa the information will be inserted
c     after routine call.
c
      subroutine AcurlNdBam(ncount,Acurlamr,Acurlami,Bcurlamr,Bcurlami,
     &                     qmur,Ccurlmur,Ccurlmui,Bcurlr,Bcurli)
      implicit none
      include 'latticeSize.h'
      include 'LatParamtrans.h'

c     global variables.

      integer,parameter                                      :: nmx=-nx/2+1,npx=nx/2
      integer,parameter                                      :: nmy=-ny/2+1,npy=ny/2
      integer,parameter                                      :: nmz=-nz/2+1,npz=nz/2
      integer,parameter                                      :: nmt=-nt/2+1,npt=nt/2

      integer                                                :: ncount

      double precision,dimension((npy-1)*npx*npt*(npz-1))     :: Acurlamr,Acurlami
      double precision,dimension((npy-1)*npx*npt*(npz-1))     :: Bcurlamr,Bcurlami
cmf$  layout Acurlamr(:news)
cmf$  layout Acurlami(:news)
cmf$  layout Bcurlamr(:news)
cmf$  layout Bcurlami(:news)

      double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt,mu):: qmur
cmf$  layout qmur(:news,:news,:news,:news,:serial)
      double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt)   :: qsq
cmf$  layout qsq(:news,:news,:news,:news)

      double precision,dimension((npy-1)*npx*npt*(npz-1))           :: psqam
cmf$  layout psqam(:news)

      double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt)   :: Bcurlr,Bcurli
cmf$  layout Bcurlr(:news,:news,:news,:news)
cmf$  layout Bcurli(:news,:news,:news,:news)
      double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt,mu)
     &                                                       :: Ccurlmur,Ccurlmui
cmf$  layout Ccurlmur(:news,:news,:news,:news,:serial)
cmf$  layout Ccurlmui(:news,:news,:news,:news,:serial)

c     local variables

      double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt):: Acurlr,Acurli
```

```
cmf$  layout Acurlr(:news,:news,:news,:news)
cmf$  layout Acurli(:news,:news,:news,:news)

      double precision,dimension(0:npx-1,0:npy-1,0:npz-1,0:npt-1):: AZ2curlr,AZ2curli
cmf$  layout AZ2curlr(:news,:news,:news,:news)
cmf$  layout AZ2curli(:news,:news,:news,:news)
      double precision,dimension(0:npx-1,0:npy-1,0:npz-1,0:npt-1):: AZ3curlr,AZ3curli
cmf$  layout AZ3curlr(:news,:news,:news,:news)
cmf$  layout AZ3curli(:news,:news,:news,:news)
      double precision,dimension(0:npx-1,0:npy-1,0:npz-1,0:npt-1):: BZ2curlr,BZ2curli
cmf$  layout BZ2curlr(:news,:news,:news,:news)
cmf$  layout BZ2curli(:news,:news,:news,:news)
      double precision,dimension(0:npx-1,0:npy-1,0:npz-1,0:npt-1):: BZ3curlr,BZ3curli
cmf$  layout BZ3curlr(:news,:news,:news,:news)
cmf$  layout BZ3curli(:news,:news,:news,:news)

      integer                                                 :: px,py,pz,pt
      integer                                                 :: counter
      integer                                                 :: imu

      interface
         subroutine Z2averaging(AZ2r,AZ2i,Ar,Ai)
          implicit none
          include 'latticeSize.h'
          include 'LatParamtrans.h'
          integer,parameter                                    :: nmx=-nx/2+1,npx=nx/2
          integer,parameter                                    :: nmy=-ny/2+1,npy=ny/2
          integer,parameter                                    :: nmz=-nz/2+1,npz=nz/2
          integer,parameter                                    :: nmt=-nt/2+1,npt=nt/2
          double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt)    :: Ar,Ai
cmf$  layout Ar(:news,:news,:news,:news)
cmf$  layout Ai(:news,:news,:news,:news)
          double precision,dimension(0:npx-1,0:npy-1,0:npz-1,0:npt-1)    :: AZ2r,AZ2i
cmf$  layout AZ2r(:news,:news,:news,:news)
cmf$  layout AZ2i(:news,:news,:news,:news)
         end subroutine Z2averaging
         subroutine Z3averaging(AZ3r,AZ3i,AZ2r,AZ2i)
          implicit none
          include 'latticeSize.h'
          include 'LatParamtrans.h'
          integer,parameter                                    :: nmx=-nx/2+1,npx=nx/2
          integer,parameter                                    :: nmy=-ny/2+1,npy=ny/2
          integer,parameter                                    :: nmz=-nz/2+1,npz=nz/2
          integer,parameter                                    :: nmt=-nt/2+1,npt=nt/2
          double precision,dimension(0:npx-1,0:npy-1,0:npz-1,0:npt-1)    :: AZ3r,AZ3i
cmf$  layout AZ3r(:news,:news,:news,:news)
cmf$  layout AZ3i(:news,:news,:news,:news)
          double precision,dimension(0:npx-1,0:npy-1,0:npz-1,0:npt-1)    :: AZ2r,AZ2i
cmf$  layout AZ2r(:news,:news,:news,:news)
cmf$  layout AZ2i(:news,:news,:news,:news)
         end subroutine Z3averaging
         subroutine compression(ncount,psqam,amr,ami,AZ3r,AZ3i)
```

```
          implicit none
          include 'latticeSize.h'
          include 'LatParamtrans.h'
          integer,parameter                             :: nmx=-nx/2+1,npx=nx/2
          integer,parameter                             :: nmy=-ny/2+1,npy=ny/2
          integer,parameter                             :: nmz=-nz/2+1,npz=nz/2
          integer,parameter                             :: nmt=-nt/2+1,npt=nt/2
          integer                                                 :: ncount
          double precision,dimension((npy-1)*npx*npt*(npz-1))     :: psqam
cmf$  layout psqam(:news)
          double precision,dimension((npy-1)*npx*npt*(npz-1))     :: amr,ami
cmf$  layout amr(:news)
cmf$  layout ami(:news)
          double precision,dimension(0:npx-1,0:npy-1,0:npz-1,0:npt-1)   :: AZ3r,AZ3i
cmf$  layout AZ3r(:news,:news,:news,:news)
cmf$  layout AZ3i(:news,:news,:news,:news)
         end subroutine compression
      end interface

c     start of the execution commands

c     now constrcutig the lattice momentum q_\mu

      Acurlr = 0.0d0
      Acurli = 0.0d0
      qsq = 0.0d0

      do imu=1,mu

         Acurlr(:,:,:,:) = Acurlr(:,:,:,:) +
     &                     ( qmur(:,:,:,:,imu) * Ccurlmur(:,:,:,:,imu) )
         Acurli(:,:,:,:) = Acurli(:,:,:,:) +
     &                     ( qmur(:,:,:,:,imu) * Ccurlmui(:,:,:,:,imu) )

         qsq = qsq + qmur(:,:,:,:,imu)**2

      end do

      Acurlr = (1.0d0/qsq) * Acurlr
      Acurli = (1.0d0/qsq) * Acurli

c     now the Z2 averaging.

      call Z2averaging(AZ2curlr,AZ2curli,Acurlr,Acurli)
      call Z2averaging(BZ2curlr,BZ2curli,Bcurlr,Bcurli)

c     Z3 averaging.

      call Z3averaging(AZ3curlr,AZ3curli,AZ2curlr,AZ2curli)
      call Z3averaging(BZ3curlr,BZ3curli,BZ2curlr,BZ2curli)

c     now compresing the arrays.
```

```
c     insert cuts inside the loops.

      call compression(ncount,psqam,Acurlamr,Acurlami,AZ3curlr,AZ3curli)
      call compression(ncount,psqam,Bcurlamr,Bcurlami,BZ3curlr,BZ3curli)

      return
      end subroutine AcurlNdBam
c
c     ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Author: Frederic D.R. Bonnet; date: 2nd of February 2001.
c     subroutine to calculate the uncorrected C_\mu(p) and and B(p).
c
      subroutine AandB(ncount,Ar,Ai,Br,Bi,Acurlaver,Acurlavei,Bcurlaver,Bcurlavei,
     &                 qsqam,ikappa)
      implicit none
      include 'latticeSize.h'
      include 'LatParamtrans.h'

c     global variables.

      integer,parameter                                        :: nmx=-nx/2+1,npx=nx/2
      integer,parameter                                        :: nmy=-ny/2+1,npy=ny/2
      integer,parameter                                        :: nmz=-nz/2+1,npz=nz/2
      integer,parameter                                        :: nmt=-nt/2+1,npt=nt/2

      integer                                                  :: ikappa
      integer                                                  :: ncount

      double precision,dimension((npy-1)*npx*npt*(npz-1))            :: qsqam
cmf$  layout qsqam(:news)

      double precision,dimension(nkappa/2,(npy-1)*npx*npt*(npz-1))   :: Br,Bi
cmf$  layout Br(:serial,:news)
cmf$  layout Bi(:serial,:news)
      double precision,dimension(nkappa/2,(npy-1)*npx*npt*(npz-1))   :: Ar,Ai
cmf$  layout Ar(:serial,:news)
cmf$  layout Ai(:serial,:news)

      double precision,dimension(nkappa/2,(npy-1)*npx*npt*(npz-1))
     &                                                         :: Bcurlaver,Bcurlavei
cmf$  layout Bcurlaver(:serial,:news)
cmf$  layout Bcurlavei(:serial,:news)
      double precision,dimension(nkappa/2,(npy-1)*npx*npt*(npz-1))
     &                                                         :: Acurlaver,Acurlavei
cmf$  layout Acurlaver(:serial,:news)
cmf$  layout Acurlavei(:serial,:news)

c     local variables.

      double precision,dimension(nkappa/2,(npy-1)*npx*npt*(npz-1)):: normj,f1r,f1i
cmf$  layout normj(:serial,:news)
cmf$  layout f1r(:serial,:news)
```

```
cmf$  layout f1i(:serial,:news)
      double precision,dimension(nkappa/2,(npy-1)*npx*npt*(npz-1)):: Acurlsqr,Acurlsqi
cmf$  layout Acurlsqr(:serial,:news)
cmf$  layout Acurlsqi(:serial,:news)

c     start of the execution commands.

      Ar(ikappa,:) = 0.0d0
      Ai(ikappa,:) = 0.0d0

      Br(ikappa,:) = 0.0d0
      Bi(ikappa,:) = 0.0d0

c     now calculating the normal C_{\mu}(p) as defined by 1.
c     similarly for the normal B(p) as defined by 2.

      Acurlsqr(ikappa,1:ncount) = ( Acurlaver(ikappa,1:ncount)**2 -
     &                              Acurlavei(ikappa,1:ncount)**2 ) * qsqam(1:ncount)
      Acurlsqi(ikappa,1:ncount) = ( 2.0d0 * Acurlaver(ikappa,1:ncount) *
     &                              Acurlavei(ikappa,1:ncount) ) * qsqam(1:ncount)

c     Bsqcurlaver = Bcurlaver**2 - Bcurlavei**2
c     Bsqcurlavei = 2.0d0 * Bcurlaver * Bcurlavei

      f1r(ikappa,1:ncount) = Acurlsqr(ikappa,1:ncount) +
     &                       Bcurlaver(ikappa,1:ncount)**2 -
     &                       Bcurlavei(ikappa,1:ncount)**2
      f1i(ikappa,1:ncount) = Acurlsqi(ikappa,1:ncount) +
     &                       2.0d0 * Bcurlaver(ikappa,1:ncount) *
     &                       Bcurlavei(ikappa,1:ncount)

      normj(ikappa,1:ncount) = f1r(ikappa,1:ncount)**2 + f1i(ikappa,1:ncount)**2

      Ar(ikappa,1:ncount) = (   Acurlaver(ikappa,1:ncount) * f1r(ikappa,1:ncount) +
     &                          Acurlavei(ikappa,1:ncount) * f1i(ikappa,1:ncount) )
     &                          / normj(ikappa,1:ncount)
      Ai(ikappa,1:ncount) = ( - Acurlaver(ikappa,1:ncount) * f1i(ikappa,1:ncount) +
     &                          Acurlavei(ikappa,1:ncount) * f1r(ikappa,1:ncount) )
     &                          / normj(ikappa,1:ncount)

      Br(ikappa,1:ncount) = (   Bcurlaver(ikappa,1:ncount) * f1r(ikappa,1:ncount) +
     &                          Bcurlavei(ikappa,1:ncount) * f1i(ikappa,1:ncount) )
     &                          / normj(ikappa,1:ncount)
      Bi(ikappa,1:ncount) = ( - Bcurlaver(ikappa,1:ncount) * f1i(ikappa,1:ncount) +
     &                          Bcurlavei(ikappa,1:ncount) * f1r(ikappa,1:ncount) )
     &                          / normj(ikappa,1:ncount)

      return
      end subroutine AandB
c
c     cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Author: Frederic D.R. Bonnet; date: 25th of June 2001.
```

```
c     subroutine to get the compressed Z3 averaged functions. It also
c     returns the momentum values.
c     The cuts are to be inserted in this section of the code.
c
      subroutine compression(ncount,psqam,amr,ami,AZ3r,AZ3i)
      implicit none
      include 'latticeSize.h'
      include 'LatParamtrans.h'

c     global variables.

      integer,parameter                                    :: nmx=-nx/2+1,npx=nx/2
      integer,parameter                                    :: nmy=-ny/2+1,npy=ny/2
      integer,parameter                                    :: nmz=-nz/2+1,npz=nz/2
      integer,parameter                                    :: nmt=-nt/2+1,npt=nt/2

      integer                                              :: ncount

      double precision,dimension((npy-1)*npx*npt*(npz-1))     :: psqam
cmf$  layout psqam(:news)
      double precision,dimension((npy-1)*npx*npt*(npz-1))     :: amr,ami
cmf$  layout amr(:news)
cmf$  layout ami(:news)

      double precision,dimension(0:npx-1,0:npy-1,0:npz-1,0:npt-1)   :: AZ3r,AZ3i
cmf$  layout AZ3r(:news,:news,:news,:news)
cmf$  layout AZ3i(:news,:news,:news,:news)

c     loacal variables.

      double precision                                     :: pie
      double precision                                     :: psq=1.0d0,deltap

      double precision,dimension(mu)                       :: pmu
cmf$  layout pmu(:serial)

      integer                                              :: px,py,pz,pt
      integer                                              :: counter

c     start of the executions.

c     insert the cuts in the do loop.

      pie= 4.0d0 * atan(1.0d0)
      counter = 0
      do pt = 0, npt-1
         do px = 0, npx-1
            do py = 0, px
               do pz = 0, py

                  pmu(1) = ( ( 2.0d0 * pie ) / nx ) * px
                  pmu(2) = ( ( 2.0d0 * pie ) / ny ) * py
```

```
                 pmu(3) = ( ( 2.0d0 * pie ) / nz ) * pz
                 pmu(4) = ( ( 2.0d0 * pie ) / nt ) * ( pt + 1.0d0/2.0d0 )

                 psq = pmu(1)**2+pmu(2)**2+pmu(3)**2+pmu(4)**2
c                 deltap = sqrt(psq) * sin( acos( (pmu(1)+pmu(2)+pmu(3)+pmu(4))
c    &                     / (2.0d0*sqrt(psq)) ) )

c     performing the cuts half momentum and cylinder cut.

c               if ( deltap < (4.0d0*pie/nx) .and. deltap > (-4.0d0*pie/nx) ) then
c                 if ( deltap < (2.0d0*pie/nx) .and. deltap > (-2.0d0*pie/nx) ) then
c                   if ( pmu(4) <= pie/2.0d0 .and. pmu(4) >= -pie/2.0d0) then

c                 if ( pmu(1) <= pie/2.0d0 .and. pmu(1) >= -pie/2.0d0 ) then
c                    if ( pmu(2) <= pie/2.0d0 .and. pmu(2) >= -pie/2.0d0) then
c                       if ( pmu(3) <= pie/2.0d0 .and. pmu(3) >= -pie/2.0d0) then
c                          if ( pmu(4) <= pie/2.0d0 .and. pmu(4) >= -pie/2.0d0) then

                            counter = counter + 1
                            psqam(counter) = pmu(1)**2+pmu(2)**2+pmu(3)**2+pmu(4)**2
                            amr(counter) = AZ3r(px,py,pz,pt)
                            ami(counter) = AZ3i(px,py,pz,pt)

c                          end if
c                       end if
c                    end if
c                 end if

              end do
           end do
        end do
     end do

     ncount = counter

     return
     end subroutine compression
c
c    cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c    routine moified to only do second order Jacknife analysis.
c    Author Frederic D.R. Bonnet, date 29th June 2001.
c    configurations, arranged according to :
c    g(0,     0)      average over all configurations
c    g(ncon+1,0)      will be jackknifed error for g(0,0)
c    g(ncon+2,0)      will be gavg - g(0,0)
c    g(i,     0)      average over all but 'i'th conf.,
c                             (i=1,ncon)
c    g(i,ncon+1)      will be jackknifed error for g(i,0)
c    g(i,     j)      average over all but 'i'th and 'j'th conf.
c                             (i,j=1,ncon), g(i,i,0)=g(i,0)
c    g(i,     j)      will be jackknifed error for g(i,j)
c    g(i,     j)      av. over all but 'i'th, 'j'th conf.
```

```
c                             g(i,j)=g(i,j)
c
      subroutine complement(gin,g)
      implicit none
      include 'latticeSize.h'
      include 'LatParamtrans.h'

c     global variables.

      integer,parameter                                       :: ncon=nprop-1

      double precision,dimension(ncon)                        :: gin
cmf$  layout gin(:serial)

      double precision,dimension(0:ncon+2,0:ncon+1) :: g
cmf$  layout g(:serial,:serial)

c     local variables.

      integer                                                 :: it,icon,jcon

c     start of the execution commands.

      g(0,0)=0.0d0

      do icon=1,ncon
         g(0,0) = g(0,0) + gin(icon)
      end do

      do icon=1,ncon
         g(icon,0) = g(0,0) - gin(icon)
         do jcon=1,ncon
            g(icon,jcon) = g(icon,0) - gin(jcon)
         end do
      end do

c     handling of normalization for first two levels of Jackknife

      do icon=1,ncon

         do jcon=1,ncon
            g(icon,jcon) = g(icon,jcon) / ( ncon-2 )
         end do

         g(icon,0) = g(icon,0) / ( ncon-1 )

      end do

      g(0,0) = g(0,0) / ncon

      return
      end subroutine complement
```

```
c
c     ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     subroutine jack1 to calculate the first order Jacknife.
c     routine modified (from correlation.fcm) to f90 style and to suit only second
c     order Jacknife.
c     Author: Frederic D.R. Bonnet, date: 29th of June 2001.
c     computes a first-order jackknife
c
      subroutine jack1(g,nlb,nle,lb,le)
      implicit none
      include 'latticeSize.h'
      include 'LatParamtrans.h'

c     global variables.

      integer,parameter                                       :: ncon=nprop-1

      integer                                                 :: nlb,nle,lb,le,l,icon

      double precision,dimension(nlb:nle,0:ncon+2)            :: g
cmf$  layout g(:serial,:serial)

c     loacal variables.

      double precision                                        :: gavg

c     start of the execution commands.

      do l=lb,le
        g(l,ncon+1)=0.0d0
        gavg=0.0d0
        do icon=1,ncon
          gavg=gavg+g(l,icon)
        end do
        gavg=gavg/ncon
        do icon=1,ncon
          g(l,ncon+1)=g(l,ncon+1)+(g(l,icon)-gavg)**2
        end do
        g(l,ncon+1)=sqrt(g(l,ncon+1)*(ncon-1)/ncon)
        g(l,ncon+2)=gavg - g(l,0)
      end do

      return
      end subroutine jack1
c
c     cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     subroutine jack2 to calculate the first order Jacknife.
c     routine modified (from correlation.fcm) to f90 style and to suit only second
c     order Jacknife.
c     Author: Frederic D.R. Bonnet, date: 29th of June 2001.
c     computes a second-order jackknife
c
```

```
      subroutine jack2(g,nlb,nle,lb,le)
      implicit none
      include 'latticeSize.h'
      include 'LatParamtrans.h'

c     global variables.

      integer,parameter                                        :: ncon=nprop-1

      integer                                                  :: nlb,nle,lb,le

      double precision,dimension(nlb:nle,0:ncon+2,0:ncon+1)    :: g
cmf$  layout g(:serial,:serial,:serial)

c     loacal variables.

      integer                                                  :: l,icon,jcon
      double precision                                         :: gavg

c     start of the execution commands.

      do l=lb,le
        do icon=1,ncon
          g(l,icon,ncon+1)=0.0d0
          gavg=0.0d0
          do jcon=1,ncon
            if (jcon.ne.icon) then
              gavg=gavg+g(l,icon,jcon)
            end if
          end do
          gavg=gavg/(ncon-1)
          do jcon=1,ncon
            if (jcon.ne.icon) then
              g(l,icon,ncon+1)=g(l,icon,ncon+1) + (g(l,icon,jcon)-gavg)**2
            end if
          end do

          g(l,icon,ncon+1)=sqrt(g(l,icon,ncon+1)*(ncon-2)/(ncon-1))

        end do
      end do

      return
      end subroutine jack2
c
c     cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     extrapolates m to kappa_crit,
c     jackknifes m(kappa_crit).
c     subroutine adapted to f90 style by
c     author: Frederic D.R. Bonnet; date: 30th of June 2001.
c
      subroutine extrapjack(m,mcr1,mqk)
      implicit none
      include 'latticeSize.h'
      include 'LatParamtrans.h'

c     global variables

      integer,parameter                                    :: ncon=nprop-1

      double precision,dimension(nkappa/2)                 :: mqk
cmf$  layout mqk(:serial)
      double precision,dimension(1,0:ncon+2)               :: mcr1
cmf$  layout mcr1(:serial,:serial)

      double precision,dimension(nkappa/2,0:ncon+2,0:ncon+1)  :: m
cmf$  layout m(:serial,:serial,:serial)

c     local variables

      integer                                              :: icon,jcon,ikappa

      double precision,dimension(0:ncon+2)                 :: mcr
cmf$  layout mcr(:serial)

      interface
         subroutine jack1(g,nlb,nle,lb,le)
          implicit none
          include 'latticeSize.h'
          include 'LatParamtrans.h'
          integer,parameter                                :: ncon=nprop-1
          integer                                          :: nlb,nle,lb,le,l,icon
          double precision,dimension(nlb:nle,0:ncon+2)     :: g
cmf$  layout g(:serial,:serial)
         end subroutine jack1
         subroutine extrap(m,merr,mcr,mqk)
          implicit none
          include 'latticeSize.h'
          include 'LatParamtrans.h'
          double precision                                 :: mcr,kapcr
          double precision,dimension(nkappa/2)             :: m,merr
cmf$  layout m(:serial)
cmf$  layout merr(:serial)
          double precision,dimension(nkappa)               :: mqk
cmf$  layout mqk(:serial)
         end subroutine extrap
      end interface

c     start of the exceution commands

c     first initialize the variables.

      do icon=1,ncon
         CALL extrap(m(1:nkappa/2,icon,0),m(1:nkappa/2,icon,ncon+1),mcr1(1,icon),mqk)
      end do
      CALL extrap(m(1:nkappa/2,0,0),m(1:nkappa/2,ncon+1,0),mcr1(1,0),mqk)
      CALL jack1(mcr1(:,0:ncon+2),1,1,1,1)

      return
      end subroutine extrapjack
c
c     cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Least-squares fit of m versus 1/kappa
c     extrapolates m linearly in 1/kappa to get m(1/kappa_crit).
c     subroutine adapted to f90 style by
c     author: Frederic D.R. Bonnet; date: 30th of June 2001.
c
c     Solution by use of the Normal Equations.
c
c     y_i=a_1 * R_1 (x_i) + a_2 * R_2 (x_i) + a_3 * R_3 (x_i)
c
c     y_i=mcr
c     x_i=xm
c
c     R_1=1
c     R_2=1/xkappa_i
c     R_3=(1/xkappa_i - 1/kapcr)**d
c
c     See Numerical Receipes Pages 509 to 511 (2nd edition 665-668)
c     for reference.
c     Note: r==alpha ; b == beta.
c
      subroutine extrap(m,merr,mcr,mqk)
      implicit none
      include 'latticeSize.h'
      include 'LatParamtrans.h'

c     global variables

      double precision                                            :: mcr

      double precision,dimension(nkappa/2)                        :: m,merr
cmf$  layout m(:serial)
cmf$  layout merr(:serial)

      double precision,dimension(nkappa/2)                        :: mqk
cmf$  layout mqk(:serial)

c     local variables

      double precision                                            :: varinv,x,y,det

      double precision,dimension(2)                               :: a,b
cmf$  layout a(:serial)
cmf$  layout b(:serial)
      double precision,dimension(2,2)                             :: r,ri
```

```
cmf$  layout r(:serial,:serial)
cmf$  layout ri(:serial,:serial)

      integer                                                   :: ikappa,i,j

      b = 0.0d0
      r = 0.0d0

c      do i=1,3
c         b(i)=0.0d0
c         do j=1,3
c            r(i,j)=0.0d0
c         end do
c      end do

      do ikappa=1,nkappa/2
         varinv=(1.0d0/(merr(ikappa)))**2
         x=mqk(ikappa)
         y=m(ikappa)
         r(1,1)=r(1,1)+varinv
         r(1,2)=r(1,2)+x*varinv
         r(2,2)=r(2,2)+x**2*varinv
         b(1)=b(1)+y*varinv
         b(2)=b(2)+x*y*varinv
      end do
      r(2,1)=r(1,2)

c     ri=r**-1

      det=r(1,1)*r(2,2)-r(1,2)*r(2,1)
      ri(1,1)=r(2,2)/det
      ri(1,2)=-r(2,1)/det
      ri(2,2)=r(1,1)/det
      ri(2,1)=ri(1,2)

c     best fit is y=a1+a2*x where a=(r**-1)b

      a(1)=ri(1,1)*b(1)+ri(1,2)*b(2)
      a(2)=ri(2,1)*b(1)+ri(2,2)*b(2)

c     determine m(1/kappa_crit)

      mcr=a(1)

      return
      end subroutine extrap
c
c     ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Author: Frederic D.R. Bonnet; date: 11th of June 2001.
c     subroutine to write out the curli C and curli B to disk.
c     the routine writes out the curli on a configuration basis
c     with a selection of the which array it needs to do.
```

```
c     the switch for the curli B is 0; and the curli C_\mu is 1
c
      subroutine CurliCandBin(Bcurlr,Bcurli,Ccurlmur,Ccurlmui,
     &                        qpk,xkappa,dataf,ikappa,bcx,bcy,bcz,bct)
      implicit none
      include 'latticeSize.h'
      include 'LatParamtrans.h'

c     global variables.

      integer,parameter                                      :: nmx=-nx/2+1,npx=nx/2
      integer,parameter                                      :: nmy=-ny/2+1,npy=ny/2
      integer,parameter                                      :: nmz=-nz/2+1,npz=nz/2
      integer,parameter                                      :: nmt=-nt/2+1,npt=nt/2

      integer                                                :: ikappa

      double precision                                       :: bcx,bcy,bcz,bct

      double precision,dimension(nkappa)                     :: xkappa
cmf$  layout xkappa(:serial)
      character(len=100),dimension(10)                       :: dataf
cmf$  layout dataf(:serial)
      character(len=5),dimension(nkappa)                     :: qpk
cmf$  layout qpk(:serial)

      double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt)   :: Bcurlr,Bcurli
cmf$  layout Bcurlr(:news,:news,:news,:news)
cmf$  layout Bcurli(:news,:news,:news,:news)
      double precision,dimension(nmx:npx,nmy:npy,nmz:npz,nmt:npt,mu)
     &                                                       :: Ccurlmur,Ccurlmui
cmf$  layout Ccurlmur(:news,:news,:news,:news,:serial)
cmf$  layout Ccurlmui(:news,:news,:news,:news,:serial)

c     local variables.

      logical                                                :: uexists=.true.
      character(len=100)                                     :: datafile

      integer                                                :: imu

      interface
       function strlen(string)
        implicit none
        character*(*) string
        integer                                              :: strlen
        integer                                              :: i, blank
       end function strlen
      end interface

c     start of the execution commands.
```

```
      datafile = dataf(1)
      datafile = datafile(1:strlen(datafile))//qpk(ikappa)

      inquire(file=datafile,exist=uexists)

      if(uexists) then
         write(*,'(a,2x,a40,x,a)')'the Bcurli:',datafile,'exists'
         write(*,'(a,2x,a40)')'we are now proceeding with the reading of:',datafile
      elseif(.not. uexists ) then
         write(*,'(a,2x,a40,x,a)')'the Ccurlimu:',datafile,'does not exists'
         stop
      end if

      open(201,file=datafile,form='unformatted',status='old',action='read')

      read(201) xkappa(ikappa),bcx,bcy,bcz,bct

      read(201) Bcurlr(:,:,:,:)
      read(201) Bcurli(:,:,:,:)

      close(201)

      datafile = dataf(2)
      datafile = datafile(1:strlen(datafile))//qpk(ikappa)

      inquire(file=datafile,exist=uexists)

      if(uexists) then
         write(*,'(a,2x,a40,x,a)')'the Ccurlimu:',datafile,'exists'
         write(*,'(a,2x,a40)')'we are now proceeding with the reading of:',datafile
      elseif(.not. uexists ) then
         write(*,'(a,2x,a40,x,a)')'the Ccurlimu:',datafile,'does not exists'
         stop
      end if

      open(301,file=datafile,form='unformatted',status='old',action='read')

      read(301) xkappa(ikappa),bcx,bcy,bcz,bct

      do imu=1,mu
         read(301) Ccurlmur(:,:,:,:,imu)
         read(301) Ccurlmui(:,:,:,:,imu)
      end do

      close(301)

      return
      end subroutine CurliCandBin
c
c     cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     subroutine to write out the jack1 to calculate the first order Jacknife.
c     Author: Frederic D.R. Bonnet, date: 18th of October 2001.
```

```
c
      subroutine writeMbin(Mcgrallq,Zcgrallq,qpk,ncount,ikappa)
      implicit none
      include 'latticeSize.h'
      include 'LatParamtrans.h'

c     global variables.

      integer,parameter                                        :: ncon=nprop-1

      integer,parameter                                        :: nmx=-nx/2+1,npx=nx/2
      integer,parameter                                        :: nmy=-ny/2+1,npy=ny/2
      integer,parameter                                        :: nmz=-nz/2+1,npz=nz/2
      integer,parameter                                        :: nmt=-nt/2+1,npt=nt/2

      integer                                                  :: ncount
      integer                                                  :: ikappa

      character(len=5),dimension(nkappa)                       :: qpk
cmf$  layout qpk(:serial)

      double precision,dimension(nkappa/2,0:ncon+2,0:ncon+1,(npy-1)*npx*npt*(npz-1))
     &                                                         :: Mcgrallq,Zcgrallq
cmf$  layout Mcgrallq(:serial,:serial,:serial,:news)
cmf$  layout Zcgrallq(:serial,:serial,:serial,:news)

c     local variables.

      character(len=100)                                       :: datafile

      interface
          function strlen(string)
           implicit none
           character*(*) string
           integer                                              :: strlen
           integer                                              :: i, blank
          end function strlen
      end interface

c     start of the execution commands.

      datafile = 'Mcor_bin.'
      datafile = datafile(1:strlen(datafile))//qpk(ikappa)//'.dat'
      open(2000+ikappa,file=datafile,form='unformatted',
     &     status='replace',action='write')

      write(2000+ikappa) Mcgrallq(ikappa,:,:,1:ncount)

      close(2000+ikappa)

      return
      end subroutine writeMbin
```

```
c
c     ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Returns the significant length of a string.
c     Every character is significant with the
c     exception of:
c     blank     (32)
c     null      (0)
c     reqd. routines - NONE
c
      function strlen(string)
      implicit none

c     global variables.

      character*(*)                                           :: string
      integer                                                 :: strlen

c     local variables.

      integer                                                 :: i, blank

c     start of the execution commands.

      blank = ichar(' ')

      strlen = len(string)
      i = ichar(string(strlen:strlen))
      do while ((i.eq.blank .or. i.eq.0) .and. strlen.gt.0)
         strlen = strlen - 1
         i = ichar(string(strlen:strlen))
      end do
      return
      end
```

Bibliography

[1] M. E. Peskin and D. V. Schroeder, *An Introduction to Quantum Field Theory* (Addison-Wesley, New York, 1995).

[2] T. Muta, Foundations of quantum chromodynamics: An introduction to perturbative methods in gauge theories, in *World Scientific Lecture Notes In Physics, 5* (World Scientific, Singapore 1987).

[3] Jeffrey E. Mandula, *The Gluon Propagator* [hep-lat/9907020].

[4] C. D. Roberts and A. G. Williams, *Prog. Part. Nucl. Phys.* **33**, 477 (1994) [hep-ph/9403224].

[5] H. J. Rothe, *Lattice Gauge Theories: An Introduction* (World Scientific, Singapore, 1992).

[6] I. Montvay and G. Munster, *Quantum Fields on a Lattice*, (Cambridge Univ. Press, Cambridge, UK, 1994).

[7] D. Schutte, W. Zheng, and C. J. Hamer, *Phys. Rev. D* **55**, 2974 (1997) [hep-lat/9603026].

[8] S. J. Brodsky, H.-C. Pauli, and S. S. Pinsky, *Phys. Rept.* **301**, 299 (1998) [hep-ph/9705477]; S. J. Brodsky, G. McCartor, H.-C. Pauli, and S. S. Pinsky, *Part. World* **3**, 109 (1993); S. Glazek, A. Harindranath, S. Pinsky, J. Shigemitsu, and Kenneth Wilson, *Phys. Rev. D* **47**, 1599 (1993).

[9] Lattice 99, *Nucl. Phys.* (Proc. Suppl.) **83**, (2000).

[10] G. P. Lepage and D. B. Mackenzie, *On the viability of lattice perturbation theory*, *Phys. Rev. D* **48**, 2250 (1993) [hep-lat/9209022].

[11] G. P. Lepage, *Redesigning lattice QCD*, in *Perturbative and Nonperturbative Aspects of Quantum Field Theory*, (Proc. of 35th International Universitätswochen für Ken–und Teichenphysik, Schladming, Austria, March 2–9, 1996), p. 1-45, (Springer–Verlag Berlin Heidelberg, 1997) [hep-lat/9607076].

[12] K. G. Wilson, *Confinement of quarks*, *Phys. Rev. D* **10**, 2445 (1974).

[13] P. Forcrand, M. G. Perez, and I.-O. Stamatescu, *Topology of the SU(2) vacuum: A lattice study using improved cooling*, *Nucl. Phys. B* **499**, 409 (1997) [hep-lat/9701012].

[14] K. G. Wilson, *Quarks and Strings on a Lattice*, in *New Phenomena in Subnuclear Physics*, ed. A. Zichichi (Plenum Press, New York), Part A, p. 69.

[15] CP-PACS Collaboration (S. Aoki *et al.*), *Quenched Light Hadron Spectrum Phys. Rev. Lett* **84**, 238-241 (2000) [hep-lat/9904012].

[16] M. Creutz, *Quarks, Gluons and Lattices*, *Cambridge University Press* (1983a).

[17] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller & E. Teller, *J. Chem. Phys.* **21**, 1087 (1953).

[18] F. D. R. Bonnet *et al*, "General Algorithm For Improved Lattice Actions On Parallel Computing Architectures.", *J. Comp. Phys.* **170**, 1 (2001) [hep-Lat/0001017].

[19] *Review of Particle Physics, Particle Data Group*, *Eur. Phys. J. C* **15**, 1-878 (2000), [Section 29.4.4 p203].

[20] M. Creutz, *Phys. Rev. D* **21**, 2308 (1979).

[21] N. Cabibbo and E. Marinari, *A new method for updating SU(N) matrices in computer simulations of gauge theories*, *Phys. Lett. B* **119**, 387 (1982).

[22] G. Curci, P. Menotti & G. Paffuti, *Phys. Lett. B* **130**, 205 (1983); Erratum ibid **B135**, 516 (1984).

[23] M. Lucher and P. Weisz, *Comm. Math. Phys.* **97**, 59 (1985).

[24] J. W. Negele, "Instantons, the QCD Vacuum, and Hadronic.Physics." *Nucl. Phys.* (Proc. Suppl.) **73**, 92 (1999) [hep-lat/9810053].

[25] M. Alford, W. Dimm, and P. Lepage, *Lattice QCD on small computers*, *Phys. Lett.* **B361**, 87 (1995) [hep-lat/9507010].

[26] F. X. Lee & D. B. Leinweber, *Phys. Rev. D* **59**, 074504 (1999) [hep-lat/9711044].

[27] H. R. Fiebig, R. M. Woloshyn, *Phys. Lett. B* **385**, 273 (1996) [hep-lat/9603001].

[28] F. D. R. Bonnet, D. B. Leinweber, A. G. Williams, and J. M. Zanotti, *Symanzik improvement in the static quark potential*, *Submitted in Phys. Rev. D* [hep-lat/9912044].

[29] F. D. R. Bonnet, P. O. Bowman, D. B. Leinweber, and A. G. Williams, *Infrared behavior of the gluon propagator on a large volume lattice, Phys. Rev. D* **62**, 051501, (2000); F. D. R. Bonnet, P. O. Bowman, D. B. Leinweber, A. G. Williams and, J. M. Zanotti, *Infinite volume and continuum limits of the landau gauge gluon propagator*, *Phys. Rev. D* **64** 034501 (2001) [hep-lat/0101013].

[30] S. Bilson-Thompson, F. D. R. Bonnet, D. B. Leinweber, and A. G. Williams, in preparation.

[31] Timothy R. Klassen, *Nucl. Phys. B* **533**, 557 (1998) [hep-lat/9803010].

[32] Mike J. Peardon & Colin J. Morningstar, *Phys. Rev. D* **56** 4043 (1997) [hep-lat/9704011].

[33] Colin Morningstar, *Nucl. Phys.* (Proc. Suppl.) **53** 914 (1997) [hep-lat/9608019].

[34] M. F. Atiyah & I. M. Singer, *Bull. Am. Math. Soc.* **69**, 422 (1963); *Ann. Math.* **87**, 484 (1968); *ibid* 546 (1968). M. F. Atiyah & I. M. Singer, (1971). "The Index of Elliptic Operators": *V, Ann. Math.* **93**, 139.

[35] E. Witten, *Nucl. Phys. B* **156**, 269(1979).

[36] G. Veneziano, *Nucl. Phys. B* **159**, 213 (1979).

[37] M. Campostrini, A. Di Giacomo, M. Maggiore, H. Panagopoulos & E. Vicari, *Phys. Lett. B* **225**, 403 (1989).

[38] M. Campostrini *et al*, *Phys. Lett. B* **212**, 206 (1988). M. Campostrini *et al*, *Nucl. Phys. B* **17**, 634 (1990). B. Alles, M. D'Elia & A. Di Giacomo, *Nucl. Phys. B* **494**, 281 (1997) [Hep-Lat/9605013].

[39] M. Lüscher, "Topology of lattice gauge fields". *Comm. Math. Phys.* **Vol 85**, 39 (1982). A. Phillips *et al.* "Lattice Gauge Fields, Principal Bundles and calculation of topological charge". *Comm. Math. Phys.* **Vol 103**, 599 (1986).

[40] M. Teper, *Phys. Lett. B* **180** 112 (1986); *Nucl. Phys. B* **288** 589 (1987).

[41] Falcioni M. Paciello M. Parisi G. Taglienti B. *Nucl. Phys. B* **251**, [FS13] 624 (1985). M. Albanese *et al*, *Phys. Lett. B* **192**, 163 (1987).

[42] A. Schwarz, *Quantum Field Theory and Topology.* A Series of Comprehensive Studies in Mathematics **307**.

[43] A. Schwarz, *Topology for Physicist.* A Series of Comprehensive Studies in Mathematics **308**.

[44] F. J. Yndurain, *The Theory of Quark and Gluon Interaction,* Chapter 8 section 1. Text and Monographs in Physics.

[45] A. A. Belavin, A. M. Polyakov, A. P. Schwartz & Y. S. Tyupkin, *Phys. Lett. B* **59**, 85 (1975).

[46] B. Sheikholeslami & R. Wohlert, *Improved Continuum Limit Lattice Action For Qcd With Wilson Fermions"*, *Nucl. Phys. B*, **259** 572 (1985).

[47] J. E. Hetrick & P. de Forcrand, *Nucl. Phys.* (Proc. Suppl.) **63**, 838 (1998) [hep-lat/9710003].

[48] T. DeGrand [MILC collaboration], *Phys. Rev. D* **60**, 094501 (1999) [hep-lat/9903006].

[49] D. B. Leinweber, J. I. Skullerud, A. G. Williams & C. Parrinello [UKQCD Collaboration], *Phys. Rev. D* **60**, 094507 (1999) [hep-lat/9811027]; D. B. Leinweber, J. I. Skullerud, A. G. Williams & C. Parrinello [UKQCD collaboration], *Phys. Rev. D* **58**, 031501 (1998) [hep-lat/9803015]; See also J. E. Mandula, *Phys. Rept.* **315** 273, (1999) and references therein.

[50] T. DeGrand, *et. al.*, *Nucl. Phys. B* **547**, 259 (1999). T. Blum, *et. al.*, *Phys. Rev. D* **55** R1133 (1997).

[51] C. Bernard & T. DeGrand, *Nucl. Phys.* (Proc. Suppl.) **83-84**, 845 (2000) [hep-lat/9909083].

[52] M. Garcia Perez *et al.*, *Nucl. Phys. B* **551**, 293 (1999) [hep-lat/9812006].

[53] F. D. R. Bonnet, P. Fitzhenry, D. B. Leinweber, M. R. Stanford & A. G. Williams, *Phys. Rev. D* **62**, 094509 (2000) [hep-lat/0001018].

[54] F. D. R. Bonnet, D. B. Leinweber, A. G. Williams & J. M. Zanotti, *submitted to Phys. Rev. D* [hep-lat/0106023].

[55] D. B. Leinweber, J-I. Skullerud, A. G. Williams & C. Parrinello, *Phys. Rev. D* **58**, 031501 (1998) [hep-lat/9803015]; *ibid, submitted to Phys. Rev. D* [hep-lat/9811027] and references therein.

[56] C. T. H. Davies *et. al.*, *Phys. Rev. D* **37**, 1581 (1988).

[57] L. Giusti, *Nucl. Phys.* **B 498**, 331 (1997); L. Giusti *et. al.*, *Phys. Lett.* **B 432**, 196 (1998) [hep-lat/9803021].

[58] A. Cucchieri & T. Mendes, *Phys. Rev. D* **57**, 3822 (1998) [hep-lat/9711047].

[59] J. P. Ma, *Mod. Phys .Lett. A* **15**, 229 (2000) [hep-lat/9903009].

[60] A. Cucchieri, *Phys. Rev. D* **60**, 034508 (1999) [hep-lat/9902023].

[61] C. Bernard, C. Paciello & A. Soni, *Nucl. Phys. B* (Proc. Suppl.) **30**, 535 (1993) [hep-lat/9211020].

[62] D. B. Leinweber, J-I. Skullerud, A. G. Williams & C. Parrinello, *Phys. Rev. D 58*, 031501 (1998); *ibid. D* **60**, 094507 (1999) [hep-lat/9811027]; *Erratum–ibid.* **D 61**, 079901 (2000).

[63] F. D. R. Bonnet, P. O. Bowman, D. B. Leinweber, D. G. Richards & A. G. Williams, *Aust. J. Phys.* **52**, 939 (1999) [hep-lat/9905006]; *Nucl. Phys.* **B** (Proc. Suppl.) 83-84, 905 (2000).

[64] A. Cucchieri & T. Mendes, *Phys. Rev. D* **57**, 3822 (1998) [hep-lat/9711047].

[65] D. Zwanziger, *Nucl. Phys. B* **364**, 127 (1991).

[66] D. Becirevic *et al.*, *Phys. Rev. D* **60**, 094509 (1999) [hep-ph/9903364]; *ibid. D***61**, 114508 (2000) [hep-ph/9910204].

[67] See for example, F. X. Lee & D. B. Leinweber, *Phys. Rev. D* **59**, 074504 (1999) [hep-lat/9711044], and references therein.

[68] Kazuyuki Kanaya, *Prog. Theor. Phys. Suppl.* **131**, 73 (1998) [hep-lat/9804006].

[69] K. Symanzik, *Nucl. Phys. B* **226**, 187 (1983); *ibid.* pp. 205ff.

[70] P. Weisz, *Nucl. Phys.* **B 212**, 1 (1983); P. Weisz & R. Wohlert, *Nucl. Phys.* **B 236**, 397 (1984); *Erratum–ibid.* **B 247**, 544 (1984).

[71] M. Lüscher and P. Weisz, *Commun. Math. Phys.* **97**, 59 (1985).

[72] F. D. R. Bonnet, P. O. Bowman, D. B. Leinweber & A. G. Williams, *Phys. Rev. D* **62**, 051501 (2000) [hep-lat/0002020].

[73] P. Marenzoni, G. Martinelli & N. Stella, *Nucl. Phys. B* **455**, 339 (1995) [hep-lat/9410011].

[74] J-I. Skullerud & A. G. Williams, *Phys. Rev. D* **63**, 054508 (2001) [hep-lat/0007028]; D. B. Leinweber, J-I. Skullerud & A. G. Williams, *Phys. Rev. D* **64**, 074508 (2001) [hep-lat/0102013].

[75] Philippe Boucaud, private communication.

[76] C. Alexandrou, Ph. de Forcrand & E. Follana, *Phys. Rev. D* **63**, 094504 (2001) [hep-lat/0008012].

[77] J-I. Skullerud & A. G. Williams, *Phys. Rev. D* **63**, 054508 (2001) [hep-lat/0007028].

[78] D. B. Leinweber, J-I. Skullerud & A. G. Williams, *Phys. Rev. D* **64**, 074508 (2001) [hep-lat/0102013].

[79] R. Alkofer & L-V. Smekal, *Phys. Rept.*, 353 (2001) [hep-ph/0007355].

[80] M. Alford, T. Klassen & P. Lepage, *Nucl. Phys.* (Proc. Suppl.) **53**, 861 (1997) [hep-lat/9608113]; *Nucl. Phys. B* **496** 377 (1997) [hep-lat/9611010]; *Nucl. Phys.* (Proc. Suppl.) **63**, 862 (1998) [hep-lat/9709126].

[81] H. Hamber and C. M. Wu, *Phys. Lett. B* **133**, 351 (1983); *Ibid B* **136**, 255 (1984).

[82] T. Eguchi and N. Kawamoto, *Nucl. Phys. B* **237**, 609 (1984).

[83] A. Bender, C. D. Roberts and L. V. Smekal, *Phys. Lett. B* **380**, 7 (1996) [nucl-th/9602012].

[84] H. Neuberger *et.al Nucl. Phys. B* **443**, 305 (1995) [hep-th/9411108].

[85] H. Neuberger, *Phys. Rev. D* **57**, 5417 (1998) [hep-lat/9710089].

[86] H. Neuberger, *Phys. Lett. B* **427**, 353 (1998) [hep-lat/9801031].

[87] P. Ginsparg & K. Wilson *Phys. Rev. D* **25**, 2649 (1982).

[88] S. J. Dong, F. X. Lee, K. F. Liu & J. B. Zhang, *Phys. Rev. Lett.* **85**, 5051 (2000) [hep-lat/0006004 v2].

[89] R. G. Edwards, U. M. Heller & R. Narayanan, *Nucl. Phys. B* **540**, 457 (1999) [hep-lat/9807017].

[90] R. G. Edwards & U. M. Heller, *Phys. Rev. D* **63**, 094505 (2001) [hep-lat/0005002].

[91] R. Edwards *et. al.*, *Phys. Rev. D* **59**, 094510 (1999) [hep-lat/9811030].

[92] C. Dawson *et. al.*, *Nucl. Phys.* (Proc. Suppl.) **63**, 877 (1998) [hep-lat/9710027].

[93] D. Becirevic, V. Gimenez, V. Lubicz & G. Martinelli, *Phys. Rev. D* **61**, 114507 (2000) [hep-lat/9909082].

[94] C. D. Roberts, [nucl-th/0007054], to appear in conference proceeding

[95] Peter C. Tandy, *For Proceedings of the Workshop on Lepton Scattering, Hadrons, and QCD*, Adelaide, Australia, March-April 2001; [nucl-th/0106031].

[96] P. Maris, C. D. Roberts & P. C. Tandy, *Phys. Lett. B* **420**, 267 (1998) [nucl-th/9707003].

[97] S. J. Dong, T. Draper, I. Horvath, F. X. Lee, K. F. Liu & J. B. Zhang, "Chiral Properties of Pseudoscalar Mesons on a Quenched 20^4 Lattice with Overlap Fermions", *Submitted to Phys. Rev. D* [hep-lat/0108020]; K. F. Liu, S. J. Dong, F. X. Lee & J. B. Zhang, *Nucl. Phys.* (Proc. Suppl.) **83**, 636-638 (2000).

[98] J. Fingberg, U. Heller & F. Karsh, *Nucl. Phys. B* **392**, 493 (1993) [hep-lat/9208012].

Appendix F

Articles Published in Journals and Publication in Progress

F.1 Published Works in Journals

- **Infinite Volume and Continuum Limits of the Landau-Gauge Gluon Propagator.**

  By Frédéric D.R. Bonnet, Patrick O. Bowman, Derek B. Leinweber, Anthony G. Williams and James M. Zanotti, *Phys. Rev. D* **64**, 034501 (2001) [hep-lat/0101013].

- **Infrared Behavior of the Gluon Propagator on a Large Volume Lattice.**

  By Frédéric D.R. Bonnet, Patrick O. Bowman, Derek B. Leinweber and Anthony G. Williams, *Phys. Rev. D* **62**, 051501 (2000) [hep-lat/0002020].

- **Calibration of Smearing and Cooling Algorithms in SU(3): Color Gauge Theory.**

  By Frédéric D.R. Bonnet, Patrick Fitzhenry, Derek B. Leinweber, Mark R. Stanford and Anthony G. Williams. *Phys. Rev. D* **62**, 094509 (2000) [hep-lat/0001018].

- **General Algorithm for Improved Lattice Actions on Parallel Computing Architectures.**

  By Frédéric D.R. Bonnet, Derek B. Leinweber and Anthony G. Williams, *J. Comput. Phys.* **170**, 1 (2001) [hep-lat/0001017].

- **Improved Landau Gauge Fixing and Discretization Errors.**

  By F.D.R. Bonnet, P.O. Bowman, D.B. Leinweber, D.G. Richards and A.G. Williams. *Nucl. Phys. Proc. Suppl.* **83**, 905 (2000) [hep-lat/9909110].

- **Discretization Errors in Landau Gauge on the Lattice.**

  By Frédéric Bonnet, D.R., Patrick O. Bowman, Derek B. Leinweber, Anthony G. Williams and David G. Richards. *Austral. J. Phys.* **52**, 939 (1999) [hep-lat/9905006].

F.2 Published Works in Proceeding

- **Gluon and Quark Propagators in Landau Gauge from the Lattice.** A. G. Williams, P. O. Bowman, F. D. R. Bonnet, D. B. Leinweber, J. I. Skullerud (Adelaide U. & DESY). Dec 1999. Published in *Adelaide 1999, Lightcone QCD and nonperturbative hadron physics* 159-164

F.3 Publications in Progress

- **Numerical Study of Lattice Index Theorem Using Improved Cooling and Overlap Fermions.**

  J. B. Zhang, S. O. Bilson-Thompson, F. D. R. Bonnet, D. B. Leinweber, A. G. Williams & J. M. Zanotti (Adelaide U.), submitted to *Phys. Rev. D* [hep-lat/0111060].

- **Improved Smoothing Algorithms for Lattice QCD.**

  By Frédéric D. R. Bonnet, Derek B. Leinweber, Anthony G. Williams and James M. Zanotti, submitted to *Phys. Rev. D*, [hep-lat/0106023].

- **Overlap Quark Propagator in Landau Gauge.**

  By Frédéric D. R. Bonnet, Patrick O. Bowman, Derek B. Leinweber and Anthony G. Williams, to be submitted to *Phys. Rev. D.*

- **Hadron Masses from Novel Fat–link Fermion Action.**

  By James M. Zanotti, S. Bilson–Thompson, Frédéric D. R. Bonnet, Paul D. Coddington, Derek B. Leinweber, Anthony G. Williams, Jianbo B. Zhang, Wally Melnitchouk and Frank X. Lee, submitted to *Phys. Rev. D.*

- **Symanzik Improvement in the Static Quark Potential.**

  By Frédéric D. R. Bonnet, Derek B. Leinweber, Anthony G. Williams and James M. Zanotti.submitted to *Phys. Rev. D*, [hep-lat/9912044].

- **Topological Study of the QCD Vacuum with an Over–Improved Action on Large Lattice Volumes.**

  By Frédéric D. R. Bonnet, Sundance Bilson–Thompson, Derek B. Leinweber and Anthony G. Williams.