Algorithms in Lattice QCD

Stephen M. Pickles

Doctor of Philosophy
The University of Edinburgh
1998

To Mandy.

Abstract

The enormous computing resources that large-scale simulations in Lattice QCD require will continue to test the limits of even the largest supercomputers into the foreseeable future. The efficiency of such simulations will therefore concern practitioners of lattice QCD for some time to come.

I begin with an introduction to those aspects of lattice QCD essential to the remainder of the thesis, and follow with a description of the Wilson fermion matrix M, an object which is central to my theme.

The principal bottleneck in Lattice QCD simulations is the solution of linear systems involving M, and this topic is treated in depth. I compare some of the more popular iterative methods, including Minimal Residual, Conjugate Gradient on the Normal Equation, Bi-Conjugate Gradient, QMR, BiCGSTAB and BiCGSTAB2, and then turn to a study of block algorithms, a special class of iterative solvers for systems with multiple right-hand sides. Included in this study are two block algorithms which had not previously been applied to lattice QCD.

The next chapters are concerned with a generalised Hybrid Monte Carlo algorithm (GHMC) for QCD simulations involving dynamical quarks. I focus squarely on the efficient and robust implementation of GHMC, and describe some tricks to improve its performance. A limited set of results from HMC simulations at various parameter values is presented.

A treatment of the non-hermitian Lanczos method and its application to the eigenvalue problem for M rounds off the theme of large-scale matrix computations.

Declaration

This thesis has been composed wholly by me and contains my own work, carried out as a member of the UKQCD collaboration.

The solver codes described in chapters 4 and 5 were built on earlier codes owned by the UKQCD collaboration; apart from MR, all the inversion algorithms described therein, and many new support routines, were implemented by me. Some of the material in chapter 5 has been presented in

- S. M. Pickles. *Nucl. Phys. B (Proc. Suppl.)* **63** (1998) 961-963.

The GHMC code in chapter 6 was designed and developed in collaboration with Zbigniew Sroczynski and Stephen Booth. It incorporated some pre-existing UKQCD code, and some new code written by Mauro Talevi and Bálint Joó; my own contribution to the coding effort is an estimated 7,500 lines. Some of the material in chapter 6 has been presented in

- Z. Sroczynski, S. M. Pickles and S. P. Booth. *Nucl. Phys. B (Proc. Suppl.)* **63** (1998) 949-951.

The Lanczos code in chapter 8 also incorporated some pre-existing UKQCD code; the source code for one key routine was supplied by Christine Davies and modified by me; the remaining ten new modules were written by me.

The studies in chapters 4 and 5 made use of quenched gauge field configurations generated by the UKQCD collaboration. The dynamical gauge field configurations used in chapter 7 were generated by Zbigniew Sroczynski and me.

(Stephen Pickles)

Acknowledgments

Thanks are due to Artan Boriçi, Graham Boyd, Christine Davies, Philippe de Forcrand, Roland Freund, Uwe Glässner, Joachim Hein, David Henty, Karl Jansen, Tony Kennedy, Tom Leonard, Harry Newton, Gero Ritzenhöfer, Hubert Simma, Peer Uberholz for helpful communications or stimulating discussions.

To Douglas Smith and Hubert Simma (again) for making their eigenvalue code available to me.

To Alan Irving, James Sexton, Hartmut Wittig, Chris Allton, Mauro Talevi for their involvement in the Dynamical Fermions Project.

To Bálint Joó for implementing the clover trick of appendix C, and Stephen Booth, for rewriting it in assembler.

To Joshua (R.I.P.) and Jericho for remaining cheerfully responsive.

To all the denizens of the "four-four corridor", including my supervisors Ken Bowler and David Richards.

Special thanks to Zbigniew Sroczynski and Stephen Booth. Our collaboration on the Generalised Hybrid Monte Carlo code is one I shall remember fondly.

And most of all, to my wife Mandy, who devoutly refuses to believe in quarks.

Contents

Chapter 1

Introduction

QCD (Quantum ChromoDynamics) is the gauge theory of the strong force. When expressed in the Feynman Path Integral formulation, the natural way to give meaning to the infinite-dimensional integrals therein is to regularise the theory on a discrete grid of space-time points; the resulting non-perturbative theory is known as lattice QCD.

The computing resources required to deal with even coarse lattices of small physical volumes are enormous.[1] If the goal of simulating near-continuum physics on the lattice is ever to be achieved, we will need to see steady increases in computing power as well as further improvements in lattice methods.

This thesis is concerned with the rôle of algorithms in lattice QCD.

I begin by giving, in chapter 2 a brief introduction to those aspects of lattice QCD essential to the present purpose. Readers seeking more detailed information could do worse than consult one of several textbooks on the subject [1, 2, 3].

The fermion matrix is of central importance to our theme. Hadronic physics can not be studied properly on the lattice without recourse to the quark propagator, which is nothing more than the inverse of the fermion matrix.[2] The QCD

[1]The words "small" and "coarse" are relative to what is achievable with the latest computing technology. A lattice described as "large" and "fine" by a practitioner a few years ago might be described as "small" and "coarse" today, and a "large, fine" lattice today might still seem absurdly "small" and "coarse" to the uninitiated.

[2]One might claim, tongue in cheek, that speeding up the computation of quark propagators by one third is worth several hundred thousand pounds per annum to the particle physics community in Britain alone!

partition function involves the determinant of the fermion matrix.[3] The eigenvalue spectrum of the fermion matrix is relevant to studies of phase transitions in QCD, and is related to the subject of topology. Thus all the classical problems of linear algebra are not just relevant but rather essential to simulating QCD on the lattice. Accordingly, chapter 3 is devoted to the fermion matrix and some of its properties.

A review and comparison of some of the more popular iterative methods for computing quark propagators is given in chapter 4. I touch on the subject of preconditioning, and describe briefly some alternative approaches.

In chapter 5, I consider a special class of iterative solvers for linear systems known as block algorithms. Often one requires solutions to a linear system with several right hand sides, but the same coefficient matrix, and this is indeed the case when computing quark propagators for a given configuration of gauge fields. By solving several systems at the same time, it is possible to use the information obtained from the solution of each system to help in the solution of the others, (hopefully) accelerating the convergence of all. Here I study five block algorithms for non-hermitian systems, including two which had not been applied previously to lattice QCD.

Lattice simulations with dynamical fermions involve Monte Carlo sampling of gauge configurations from a distribution which involves the fermionic determinant. In this thesis, our attention is confined to the Generalised Hybrid Monte Carlo method (GHMC), which is the subject of chapters 6 and 7. A description of the algorithm and details of our implementation and efforts to improve its performance are given in chapter 6. For a more detailed review of dynamical fermion algorithms the reader is referred to [4].

A limited set of results from HMC simulations at various parameter values is presented in chapter 7. Hadron spectrum calculations on these configurations

[3]Ignoring the weighting from the determinant is the *quenched* approximation, which is equivalent in effect to neglecting virtual quark loops.

are outwith the present scope, but details will be reported in [5].

The non-hermitian Lanczos method, by which the eigenspectrum of the fermion matrix may be extracted, is studied in chapter 8.

Finally, chapter 9 summarises the salient points of this thesis.

Algorithms can not be fairly compared *in vacuo*. The hardware on which a piece of software is to run is an important consideration. For example, the now universal requirement of scalability on parallel computers rules out some otherwise excellent methods, and an efficient implementation of an ordinary algorithm will frequently outperform an ordinary implementation of the theoretical favourite. Rather than devote a separate chapter to architecture-specific considerations, I will make remarks where they seem appropriate during the text. However, wherever relevant, it is to be understood that the methods discussed in this thesis are targeted at a specific kind of architecture: massively parallel supercomputers consisting of a large number of communicating scalar processors, as exemplified by the Cray-T3D or Cray-T3E.

Furthermore, the motivation behind much of the work underlying this thesis was to better enable the physics program of the UKQCD collaboration. Thus conclusions on the relative viability of various methods are made against the backdrop of clover-improved Wilson fermions and red-black preconditioning, as the collaboration is committed to the former and has made a large investment in a highly optimised assembler matrix multiply routine for the latter.

Chapter 2

Introduction to Lattice QCD

Lattice QCD may be viewed in several ways. As a prescription for calculating observables of QCD from first principles, lattice QCD is able to make predictions which test the correctness of the standard model at energy scales not accessible by perturbative methods. As a means of defining what is meant by the QCD functional integrals, lattice QCD serves to place the continuum theory on a sounder theoretical footing. Lattice QCD is also an interesting statistical mechanical system in its own right.

2.1 Continuum QCD

We begin with the QCD Lagrangian in Minkowski space-time.

$$\mathcal{L}_{\mathrm{QCD}} = \mathcal{L}_{\mathrm{YM}} + \mathcal{L}_{\mathrm{F}} \tag{2.1}$$

The Yang-Mills term $\mathcal{L}_{\mathrm{YM}}$

$$\mathcal{L}_{\mathrm{YM}} = -\frac{1}{2}\mathrm{Tr}F_{\mu\nu}F^{\mu\nu}. \tag{2.2}$$

which describes the purely gluonic sector of the theory, is constructed from $su(3)$ fields $A_\mu(x)$ (the gauge bosons of the theory), covariant derivatives D_μ and field tensor $F_{\mu\nu}$:

$$\begin{aligned} A_\mu(x) &\equiv A_\mu^a(x)\frac{\lambda_a}{2} & (2.3)\\ D_\mu &= \partial_\mu + igA_\mu, & (2.4)\end{aligned}$$

$$F_{\mu\nu} = -\frac{i}{g}\left[D_\mu, D_\nu\right] \tag{2.5}$$

where g is a coupling constant and the λ_a are the Gell-Mann matrices §(A.3).

The fermionic term $\mathcal{L}_{\mathrm{F}}$ is given by

$$\mathcal{L}_{\mathrm{F}} = \sum_f \bar{\psi}_f \left(i\gamma^\mu D_\mu - m_f\right)\psi_f \tag{2.6}$$

where the sum is over quark flavours, and the ψ_f are fermion fields.

The Lagrangian $\mathcal{L}_{\mathrm{QCD}}$ is invariant under local gauge transformations of the fields, the gauge group being $SU(3)$.

$$\begin{aligned}
\psi_f(x) &\rightarrow \exp\left(-ig\theta^a\frac{\lambda_a}{2}\right)\psi_f(x) && (2.7)\\
\bar{\psi}_f(x) &\rightarrow \bar{\psi}_f(x)\exp\left(ig\theta^a\frac{\lambda_a}{2}\right) && (2.8)\\
A_\mu(x) &\rightarrow \exp\left(-ig\theta^a\frac{\lambda_a}{2}\right)\left(A_\mu(x) - \frac{i}{g}\partial_\mu\right)\exp\left(ig\theta^a\frac{\lambda_a}{2}\right) && (2.9)
\end{aligned}$$

To quantise the theory in the Feynman path integral formalism one would now introduce a QCD partition function (ignoring gauge-fixing and Faddeev-Popov terms)[1]

$$Z = \int \mathcal{D}A\mathcal{D}\psi\mathcal{D}\bar{\psi}\ \exp\left(i\int d^4x\mathcal{L}_{\mathrm{QCD}}(x)\right) \tag{2.10}$$

and corresponding path integral. The fermion fields are now Grassmann-valued. Unfortunately, the functional integral is not well-defined. Moreover, the wildly oscillating integrand is unsuited to a numerical approach based on Monte Carlo methods. A common remedy, especially in lattice studies, is to reformulate the original theory in Euclidean, instead of Minkowski, space-time. The transcription of the theory onto Euclidean space is accomplished by an analytic continuation of the time variable in the complex plane, $x^0 \rightarrow -ix_4^E$. This *Wick rotation*

[1]Provided that one's interest is confined to gauge-invariant observables, it will not be necessary to fix the gauge on the lattice.

modifies the Dirac matrices, the metric and the integration measure, and yields the Euclidean partition function

$$Z_E = \int \mathcal{D}A\mathcal{D}\psi\mathcal{D}\bar{\psi}\ \exp\left(\int d^4x^E \mathcal{L}^E_{\mathrm{QCD}}(x)\right). \tag{2.11}$$

Hereafter, we shall assume that the transition to Euclidean space has been made, and we shall trust the reader to bear this in mind without any superscripts E to remind him.

Quantities such as masses and decay constants are invariant under Wick rotation. If necessary, Green's functions calculated in Euclidean space can be analytically continued back to Minkowski space.[2]

2.2 Lattice QCD

The lattice approach is to replace the continuum of Euclidean space by a discrete grid of space-time points (the lattice), derivatives by finite differences, and space-time integrals by sums over the lattice. We specialise immediately to isotropic, hypercubic lattices with lattice spacing a and lattice volume $V = L^3 \times L_T$, L being the number of lattice sites in each spatial direction and L_T being the number in the temporal direction. In this thesis, we impose periodic boundary conditions on the lattice, identifying the site $x + L_\mu\hat{\mu}$ with the site x. The lattice spacing acts as an ultraviolet momentum cut-off π/a and the volume acts as an infrared cut-off; in this sense the lattice is regularising the Euclidean path integral.

Various discretisations of the fields are possible, and in principle, any discretisation will do as long as the continuum action is recovered in an appropriate limiting procedure involving $a \to 0$ and $V \to \infty$. In practice, however, simulations 'near the continuum limit' will not be achievable for many years to come, and each discretisation scheme brings its own set of problems. One issue of great practical importance is the behaviour of discretisation errors as a function of a.

Throughout this thesis, we adopt Wilson's original discretisation of the gauge

[2]Subject to certain conditions, such as reflection positivity, which need not concern us here.

fields [6]. The manner in which we represent fermions on the lattice also traces its lineage to Wilson, but incorporates improvements of more recent vintage [7, 8, 9].

2.2.1 Gauge fields on the lattice

Wilson recognised that the way to preserve local gauge invariance on the lattice is to represent the gauge fields by $SU(3)$ matrices on the lattice links instead of by $su(3)$ matrices on the lattice sites. Thus, the $SU(3)$ variable $U(x, x+\hat{\mu}) \equiv U_{\mu x}$ is associated with the link joining lattice site x to its nearest neighbour in the μ-direction, $x+\hat{\mu}$. These link fields are the parallel transporters

$$U_{\mu x} = \exp\left(-igaA_{\mu}\left(x+\frac{1}{2}\hat{\mu}\right)\right) \approx \exp\left(ig\int_{x+\hat{\mu}}^{x} A_{\mu}(x)\,dx\right). \tag{2.12}$$

The ordered product of such link variables around the elementary square is called the plaquette $\Box_{\mu\nu}(x)$

$$\Box_{\mu\nu}(x) = U_{\mu x}U_{\nu x+\hat{\mu}}U^{\dagger}_{\mu x+\hat{\nu}}U^{\dagger}_{\nu x}. \tag{2.13}$$

Note that

$$U^{\dagger}_{\nu x} = U(x+\hat{\nu}, x). \tag{2.14}$$

Local gauge transformations $V(x) \in SU(3)$ are defined at the lattice sites

$$\begin{aligned} \psi_f(x) &\rightarrow V(x)\psi_f(x) && (2.15)\\ \bar{\psi}_f(x) &\rightarrow \bar{\psi}_f(x)V(x) && (2.16)\\ U_{\mu x} &\rightarrow V(x)U_{\mu x}V^{-1}(x+\hat{\mu}). && (2.17) \end{aligned}$$

The trace of the plaquette is gauge invariant by construction, as is the trace of any product of link variables around a closed path (Wilson loop). A path-ordered product of link variables capped by a fermion and an antifermion, such as

$$\bar{\psi}(x)U_{\mu}(x)U_{\nu}(x+\hat{\mu})\ldots U_{\rho}(y-\hat{\rho})\psi(y),$$

is also gauge invariant.

The pure gauge part of the lattice action is the sum

$$S_{pg}[U] = \beta \sum_{x,\mu>\nu} \left\{ 1 - \frac{1}{3} \mathrm{Re}\ \mathrm{Tr}\ \Box_{\mu\nu}(x) \right\}, \tag{2.18}$$

where $\beta = 6/g^2$. It differs from the continuum Yang-Mills action in Euclidean space by terms of order a^2. The constant term in equation (2.18) has no physical significance and is often omitted; we shall do this ourselves in chapter 6.

The lattice counterpart of the continuum functional integral $\int \mathcal{D}A$ is $\int [dU]$ where

$$[dU] \equiv \prod_{\mu,x} dU_{\mu x}, \tag{2.19}$$

and the invariant group measure dU is the Haar measure.

2.2.2 Fermion fields on the lattice

The fermionic part of the lattice action that we adopt in this thesis is, for a single quark flavour,

$$\begin{aligned} S_{\mathrm{F}} &= \frac{1}{2\kappa} \sum_x \bar{\psi}(x) \left(1 - \frac{\kappa r C_{SW}}{2} \sigma_{\mu\nu} F_{\mu\nu}(x) \right) \psi(x) \qquad (2.20) \\ &- \frac{1}{2\kappa} \sum_{x,\mu} \bar{\psi}(x) \left[(r - \gamma_\mu) U_\mu(x) \psi(x+\hat{\mu}) + (r + \gamma_\mu) U_\mu^\dagger(x-\hat{\mu}) \psi(x-\hat{\mu}) \right] \end{aligned}$$

The significance of Wilson's parameter r and the clover coefficient C_{SW} will be discussed shortly. Equation (2.20) is not complete as it stands and must be supplemented by boundary conditions on the fermion fields. In this thesis, we use antiperiodic boundary conditions in the time direction $\hat{4}$,

$$\psi\left(x + L_T \hat{4}\right) \equiv -\psi(x) \tag{2.21}$$

and periodic boundary conditions in each spatial direction $\hat{\jmath}$,

$$\psi(x + L\hat{\jmath}) \equiv \psi(x), \quad \hat{\jmath} = \hat{1}, \hat{2}, \hat{3}. \tag{2.22}$$

The *hopping parameter*

$$\kappa = \frac{1}{2ma + 8r} \tag{2.23}$$

is related to the bare quark mass through

$$m \equiv \frac{1}{2a}\left(\frac{1}{\kappa} - \frac{1}{\kappa_c}\right) \tag{2.24}$$

which we take as the definition of the *critical value* $\kappa_c = \kappa_c(a)$ of the hopping parameter. Henceforth, we shall absorb the factor $\frac{1}{2\kappa}$ in equation (2.20) into a redefinition of ψ and $\bar{\psi}$.

If more than one quark flavour is present, then the ψ, $\bar{\psi}$, κ and r in equation (2.20) will carry flavour indices and the fermionic action will be summed over flavours.

Setting $r = 0$ in equation (2.20) yields the naïve fermion action. Its leading discretisation errors are $\mathcal{O}(a^2)$. However, it gives rise to 16 degenerate fermion species in the continuum limit.

Setting $r \neq 0$, $C_{SW} = 0$ in equation (2.20) recovers Wilson's cure for the doubling problem. The 15 non-physical doublers of the naïve action are suppressed through the acquisition of a mass that diverges as $a \to 0$. But a high price has been paid for this cure: Wilson's term explicitly breaks chiral symmetry and introduces $\mathcal{O}(a)$ discretisation errors. In the remainder of this thesis, we shall take $r = 1$ for computational convenience, as $\frac{1}{2}(1 \pm \gamma_\mu)$ is a projection.

The purpose of the term proportional to C_{SW} is to cancel, at least partly, the $\mathcal{O}(a)$ discretisation errors introduced by the Wilson term. The clover coefficient C_{SW} owes its name to the fact that

$$\begin{aligned} iF_{\mu\nu}(x) = \tfrac{1}{8}\Big[\quad & U_{\mu x} U_{\nu x+\hat{\mu}} U^\dagger_{\mu x+\hat{\nu}} U^\dagger_{\nu x} \\ + \quad & U_{\nu x} U^\dagger_{\mu x-\hat{\mu}+\hat{\nu}} U^\dagger_{\nu x-\hat{\nu}} U_{\mu x-\hat{\nu}} \\ + \quad & U^\dagger_{\mu x-\hat{\mu}} U^\dagger_{\nu x-\hat{\mu}-\hat{\nu}} U_{\mu x-\hat{\mu}-\hat{\nu}} U_{\nu x-\hat{\nu}} \\ + \quad & U^\dagger_{\nu x-\hat{\nu}} U_{\mu x-\hat{\nu}} U_{\nu x+\hat{\mu}-\hat{\nu}} U^\dagger_{\mu x} \quad \Big] - h.c. \end{aligned} \tag{2.25}$$

reminds some lattice practitioners of a four-leaf clover. The subscripts SW are in deference to Sheikholeslami and Wohlert [7] who introduced this form of the action. Various prescriptions exist for fixing C_{SW}, and each raises issues which are subtle, but incidental to the thrust of this thesis.

The fermionic action (2.20) can be expressed more succinctly by

$$S_{\mathrm{F}} = \sum_{x,y} \bar{\psi}(x) M(x,y) \psi(y) \tag{2.26}$$

where $M(x,y)$ is the Wilson fermion matrix and the subject of the next chapter.

2.3 The lattice path integral

Expectation values of operators $\mathcal{O}\left(U,\psi,\bar{\psi}\right)$ are given on the lattice by

$$\left\langle \mathcal{O}\left(U,\psi,\bar{\psi}\right)\right\rangle = \frac{1}{Z}\int [dU]\left[d\bar{\psi}\right][d\psi]\,\mathcal{O}\left(U,\psi,\bar{\psi}\right)\exp\left(-S_{pg}-S_{\mathrm{F}}\right) \tag{2.27}$$

where

$$Z = \int [dU]\left[d\bar{\psi}\right][d\psi]\exp\left(-S_{pg}-S_{\mathrm{F}}\right) \tag{2.28}$$

is the lattice partition function. We are still working, for simplicity, in a universe with only one quark flavour; the generalisation to more than one quark flavour is straightforward.

As it stands, equation (2.27) is a statement about lattice QCD viewed as a statistical mechanical system. The connection with physics is made by identifying the continuum limit of the right hand side with the vacuum expectation value of a corresponding time-ordered operator in the continuum.

Although it is possible to design algorithms for evaluating integrals over Grassmann variables, the problem in general exhibits exponential complexity [10, 11] and this kind of approach is not currently feasible on any but the smallest of lattices. It is therefore standard practice to integrate out the Grassmann variables in equation (2.27) analytically, and then deal with the remaining integral over the gauge fields by other means.

The integral

$$\int \left[d\bar{\psi}\right]\left[d\psi\right]\exp\left(-\sum_{x,y}\bar{\psi}(x)\,M(x,y)\,\psi(y)\right) = \det(M) \tag{2.29}$$

is standard (see eg. [2]), and may be applied immediately to equation (2.28):

$$Z = \int [dU]\det(M[U])\exp(-S_{pg}[U]) \equiv \int [dU]\exp\left(-S_{pg}^{\text{eff}}[U]\right) \tag{2.30}$$

where we have introduced an effective action

$$S_{pg}^{\text{eff}}[U] = S_{pg}[U] - \log\det(M[U]) = S_{pg}[U] - \text{Tr }\log M[U] \tag{2.31}$$

which depends on the gauge fields alone.

More complicated Grassmannian integrals can be handled using the machinery of generating functionals. This formalism leads to the following result:

$$\langle\psi(y)\,\bar{\psi}(x)\rangle = \frac{1}{Z}\int [dU]\exp\left(-S_{pg}^{\text{eff}}[U]\right)M^{-1}(y,x) \tag{2.32}$$

and, more generally,

$$\begin{aligned} & \langle\psi(y_1)\,\bar{\psi}(x_1)\ldots\psi(y_n)\,\bar{\psi}(x_n)\,f[U]\rangle \qquad (2.33)\\ = \quad & \frac{1}{Z}\int [dU]\exp\left(-S_{pg}^{\text{eff}}[U]\right)f[U]\,\epsilon_{y_1\cdots y_n}^{z_1\cdots z_n}M^{-1}(z_1,x_1)\ldots M^{-1}(z_n,x_n)\end{aligned}$$

where $f[U]$ is any function of the gauge fields, and the symbol $\epsilon_{i_1\cdots i_n}^{j_1\cdots j_n}$ is 1 (-1) if $\{j_1\cdots j_n\}$ is an even (odd) permutation of $\{i_1\cdots i_n\}$, or zero if $\{j_1\cdots j_n\}$ is no permutation of $\{i_1\cdots i_n\}$ at all. Thus, performing the integral over fermion fields brings down a *quark propagator* $M^{-1} = M^{-1}[U]$ for each quark-antiquark pair in the integrand.

In general, what remains in the integrand of equation (2.27) after integrating out the fermions is the product of the effective action and an effective operator which depends on the gauge fields alone, and consists of the product of the original function of the gauge fields, and zero or more quark propagators (or, more usually, some contraction thereof).

$$\langle \mathcal{O}\left(U,\psi,\bar{\psi}\right)\rangle = \int [dU] \exp\left(-S^{\text{eff}}_{pg}[U]\right) \mathcal{O}^{\text{eff}}(U). \tag{2.34}$$

We are now in a position to discuss the two major bottlenecks of lattice QCD. The first bottleneck is performing the integral over gauge fields, and the second bottleneck is calculating quark propagators.

The integral over gauge fields is performed numerically, by Monte Carlo methods. Suppose that we have available a means for generating gauge configurations $\{U_1, U_2, \ldots U_N\}$ with probability

$$P(U_k) = \frac{1}{Z}\exp\left(-S^{\text{eff}}_{pg}[U_k]\right). \tag{2.35}$$

Then the expectation value may be estimated by

$$\langle \mathcal{O}\left(U,\psi,\bar{\psi}\right)\rangle \approx \frac{1}{N}\sum_{i=1}^{N} \mathcal{O}^{\text{eff}}(U_i). \tag{2.36}$$

The usual procedure is to simulate a *Markov process* that converges to the correct distribution in the "long time" limit, and then to sample gauge configurations at regular intervals after the simulation is believed to have equilibrated. These ideas will be made clearer in chapter 6.

These simulations are horrendously expensive in terms of computer resources because of the determinants (one for each flavour) in S^{eff}_{pg}. The determinant, $\det M$, is a highly non-local object. It can be computed in polynomial time by a factorisation scheme (eg. LU-decomposition), but this method requires $\mathcal{O}(12V)^3$ floating point operations and storage for $(12V)^2$ floating point numbers.[3]

In the *quenched approximation*, the computational effort is reduced by several orders of magnitude by the simple expedient of setting $\det M$ to a constant, and it

[3]With present-day technology, direct methods are completely infeasible on lattices larger than about $V = 8^4$.

is for this very reason that most simulations to date in lattice QCD have been carried out in the quenched approximation. Physically, the quenched approximation is equivalent to neglecting the effect of virtual quark loops. Despite some notable successes (and failures), this approximation remains completely uncontrolled.

One of the primary goals of pioneering simulations with *dynamical fermions*[4] is to quantify the systematic errors that the quenched approximation induces in various observables. Chapter 6 is devoted to the (generalised) Hybrid Monte Carlo algorithm, which incorporates the effect of the fermion determinants indirectly. HMC has yet to be displaced as the most popular method for taking into account the effect of virtual quark loops in a universe populated by quarks whose flavours come in mass-degenerate pairs. This, of course, is not the real world, and genuine simulations of "full QCD" are yet to come.

The second bottleneck is the calculation of quark propagators. Hitherto, we have employed the suggestive name fermion *matrix*; now we call upon the full machinery of linear algebra. The objects that the matrix M acts on have colour, spin and site indices, and belong to a vector space of dimension $n = 3 \times 4 \times V$. In this language, the object $M(x,y)$ is itself a 12×12 matrix coupling sites x and y; choosing a basis in which colour and spin indices run faster than site indices, it corresponds to one of the elementary squares in figure 3.1. At first sight, it appears that we will have to invert the matrix M, but fortunately[5] this is not so. Because of the translational invariance of the continuum theory, it is usually necessary only to consider a quark propagating from a fixed site (usually the origin) to a site labelled by x:

$$M^{-1}(x,0),$$

so only 12 rows of the inverse are actually required. These are obtained by solving the 12 linear systems

$$M\psi_i = \eta_i, \qquad i = 1, \ldots, 12 \tag{2.37}$$

[4]The fact that the term "dynamical fermions" has become more or less standard in lattice jargon, says much about the ubiquity of the quenched approximation.

[5]Although M is sparse, M^{-1} is dense, and on any lattice larger than about 8^4 we would not even have sufficient memory to store M^{-1}.

where each η_i is a point-like source of spin and colour at the origin. Despite the notation, the ψ_i and η_i in equation (2.37) are not, strictly speaking, lattice fermion fields (which are Grassmann-valued); instead they are humble column vectors, whose components can be put into one-to-one correspondence with those of a lattice fermion field. However, as no genuine fermion fields appear in any of the computer programs considered in this thesis, we shall follow accepted practice and refer to the objects exemplified by ψ_i and η_i as "fermion fields" or "4-spinors".

Chapters 4 and 5 are devoted to methods for solving equation (2.37).

Chapter 3

The Wilson Fermion Matrix

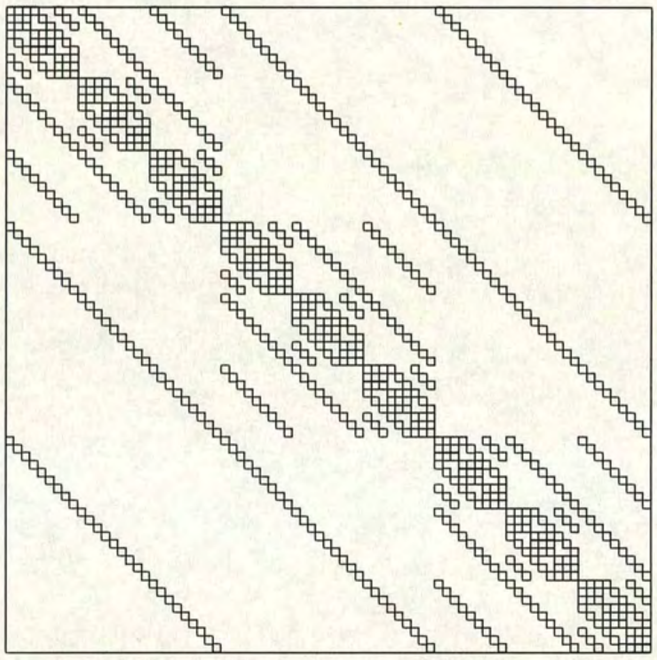


Figure 3.1: The block structure of the Wilson Fermion matrix on a 3^4 lattice. Each elementary square represents a 12×12 matrix coupling spin and colour.

3.1 Introduction

The fermion matrix M is given by

$$M = A - \kappa D \tag{3.1}$$

where the clover term

$$A = 1 - \frac{1}{2}\kappa C_{SW}\sigma_{\mu\nu}F_{\mu\nu} \tag{3.2}$$

is local, and the hopping term

$$D(x,y) = \sum_{\mu}\left\{U_\mu(x)(1-\gamma_\mu)\delta_{x+\hat{\mu},y} + U_\mu^\dagger(x-\hat{\mu})(1+\gamma_\mu)\delta_{x-\hat{\mu},y}\right\} \tag{3.3}$$

involves nearest neighbour couplings.

$M = M[U,\kappa,C_{SW}]$ couples spin to spin, colour to colour and site to site, so it has order $n = 4 \times 3 \times V$ where V is the lattice volume. For a typical lattice, $V = 24^3 \times 48$ and $n = 7962624$. M is very *large*.

As the space-time coupling is nearest neighbour, M is also *sparse*. Each row (column) of A has $3 \times 4 \times (8+1) = 108$ non-zero complex components.

M is not hermitian, but it is J-hermitian

$$JM = M^\dagger J \tag{3.4}$$

for $J = \gamma_5 \otimes I_{colour} \otimes I_{spatial}$, where $\gamma_5 = \gamma_5^\dagger = \gamma_5^{-1}$ is the familiar spin-permutation matrix.

It is customary to describe M as γ_5-symmetric (γ_5-hermitian might be more precise), and to write

$$M = \gamma_5 M^\dagger \gamma_5 \tag{3.5}$$

in which $\gamma_5 \otimes I_{colour} \otimes I_{spatial}$ has been abbreviated simply as γ_5.

The hermitian matrix

$$Q = \gamma_5 M \tag{3.6}$$

is also extremely useful, as is the lattice quark propagator $M^{-1} = Q^{-1}\gamma_5$. M and Q are *equivalent* but not *similar*.

3.2 Eigenvalues and Eigenvectors.

3.2.1 The eigenvalue spectra of Q and M.

The fact that Q is hermitian immediately gives us a lot of information about its eigenvalues λ_i and normalised eigenvectors v_i.

From

$$Qv_i = \lambda_i v_i \tag{3.7}$$

it is elementary to obtain

$$(\lambda_i - \lambda_j^*) v_i^\dagger v_j = 0 \tag{3.8}$$

from which follow the facts that

1. the eigenvalues of Q are real
2. eigenvectors of Q corresponding to distinct eigenvalues are orthogonal.

Q has a spectral expansion

$$Q = \sum_{i=1}^{n} \lambda_i v_i v_i^\dagger \tag{3.9}$$

as does its inverse, when it exists

$$Q^{-1} = \sum_{i=1}^{n} \lambda_i^{-1} v_i v_i^\dagger. \tag{3.10}$$

Expansions for M and M^{-1} can be obtained from these in the obvious way.

Properties (3.8 – 3.9) do not go through for non-hermitian matrices. However some insight into the eigenspectrum of M can be obtained from γ_5-symmetry.

It is necessary to distinguish between right eigenvectors

$$Mr_i = \rho_i r_i \tag{3.11}$$

and left eigenvectors

$$s_i^\dagger M = \sigma_i^* s_i^\dagger \tag{3.12}$$

First we notice that

$$(\gamma_5 r_i)^\dagger M = r_i^\dagger M^\dagger \gamma_5 = \rho_i^*(\gamma_5 r_i)^\dagger \tag{3.13}$$

and similarly

$$M(\gamma_5 s_i) = \sigma_i(\gamma_5 s_i). \tag{3.14}$$

The left eigenvectors are obtained by multiplying the right eigenvectors by γ_5, and vice-versa.

A useful lemma is obtained by considering Q sandwiched between eigenvectors of M.[1] On the one hand

$$r_i^\dagger \gamma_5 M r_j = \rho_j r_i^\dagger \gamma_5 r_j,$$

and on the other

$$r_i^\dagger M^\dagger \gamma_5 r_j = \rho_i^* r_i^\dagger \gamma_5 r_j,$$

and consequently

$$(\rho_i^* - \rho_j) r_i^\dagger \gamma_5 r_j = 0. \tag{3.15}$$

Setting $i = j$ tells us that each eigenvalue of M is either real or its corresponding eigenvector has zero *chirality* χ, defined by

$$\chi(x) = \frac{x^\dagger \gamma_5 x}{x^\dagger x}. \tag{3.16}$$

One widely known fact is that the ρ_i are either real or occur in complex-conjugate pairs. This can be proved by considering the characteristic polynomial for M.

$$\begin{aligned} P(\rho) &= \det(M - \rho I) \\ &= \det(\gamma_5)\det(M - \rho I)\det(\gamma_5) \\ &= \det(\gamma_5 M \gamma_5 - \rho I) \\ &= \det(M^\dagger - \rho I) \\ &= \det(M - \rho^* I)^* \end{aligned}$$

[1]The same result was published in reference [12] in the context of lattice QED$_2$, and hailed as a "new theorem". I noticed it in 1996, some 9 years after [13]. But see also [14, 15, 16, 17].

$$= P(\rho^*)^*$$

So if ρ satisfies $P(\rho) = 0$, then ρ^* does too. This completes the proof.

3.2.2 A 'spectral' expansion for M

We now seek an expansion for M in terms of its eigenvalues and eigenvectors.[2] Such an expansion is easily written down for hermitian matrices (see eg. equation (3.9)). The situation is a good deal more complicated for non-hermitian matrices, whose eigenvectors will in general not be orthogonal.

For the sake of clarity, let us assume that M is diagonalisable with non-degenerate eigenvalues. Appendix B shows how to handle the more general case of a diagonalisable J-hermitian matrix with possibly degenerate eigenvalues.

Because the eigenvalues are real or occur in complex conjugate pairs, and by the assumption of non-degeneracy of eigenvalues, we have that for each r_i satisfying (3.11) there is an unique $\tilde{r}_i$ satisfying

$$M\tilde{r}_i = \rho_i^*\tilde{r}_i. \tag{3.17}$$

When ρ_i is real, r_i and $\tilde{r}_i$ coincide. This we use to re-write equation (3.15) in the form

$$(\rho_i - \rho_j)\tilde{r}_i^\dagger\gamma_5 r_j = 0. \tag{3.18}$$

We now make one additional assumption, namely

$$\tilde{r}_i^\dagger\gamma_5 r_i \neq 0. \tag{3.19}$$

(This assumption is not as restrictive as it looks. It turns out that $\left|\tilde{r}_i^\dagger\gamma_5 r_i\right|$ is the

[2]Note added in proof. The same result was proven in 1987 by Itoh, Iwasaki and Yoshié [13], but no details were given for the case of degenerate eigenvalues. I give a constructive proof of the more general case in Appendix B.

reciprocal of the *spectral condition number* csp (ρ_i) [18] of the eigenvalue ρ_i. So if our assumption (3.19) is untrue, the eigenvalue problem (3.11) is ill-conditioned anyway.) Using this assumption in equation (3.18), we deduce that

$$\frac{\tilde{r}_i^\dagger \gamma_5 r_j}{\tilde{r}_i^\dagger \gamma_5 r_i} = \delta_{ij}. \tag{3.20}$$

Let R be the $n \times n$ matrix whose columns are the right eigenvectors of M, and let Δ be the diagonal matrix of eigenvalues.

$$\begin{aligned} R &= [r_1 \ldots r_n] &(3.21)\\ \Delta &= \mathrm{diag}(\rho_1, \ldots, \rho_n) &(3.22)\end{aligned}$$

Now consider the matrix product

$$\begin{bmatrix} \left(\frac{\gamma_5 \tilde{r}_1}{r_1^\dagger \gamma_5 \tilde{r}_1}\right)^\dagger \\ \vdots \\ \left(\frac{\gamma_5 \tilde{r}_n}{r_n^\dagger \gamma_5 \tilde{r}_n}\right)^\dagger \end{bmatrix} \begin{bmatrix} r_1 & \cdots & r_n \end{bmatrix} = \begin{bmatrix} \frac{\tilde{r}_1^\dagger \gamma_5 r_1}{\tilde{r}_1^\dagger \gamma_5 r_1} & \ldots & \frac{\tilde{r}_1^\dagger \gamma_5 r_n}{\tilde{r}_1^\dagger \gamma_5 r_1} \\ \vdots & \ddots & \vdots \\ \frac{\tilde{r}_n^\dagger \gamma_5 r_1}{\tilde{r}_n^\dagger \gamma_5 r_n} & \ldots & \frac{\tilde{r}_n^\dagger \gamma_5 r_n}{\tilde{r}_n^\dagger \gamma_5 r_n} \end{bmatrix} = I_{n\times n}. \tag{3.23}$$

The second equality follows from equation (3.20). We have established that

$$R^{-1} = \begin{bmatrix} \frac{\gamma_5 \tilde{r}_1}{r_1^\dagger \gamma_5 \tilde{r}_1} & \cdots & \frac{\gamma_5 \tilde{r}_n}{r_n^\dagger \gamma_5 \tilde{r}_n} \end{bmatrix}^\dagger \tag{3.24}$$

Notice now that

$$R\Delta = \begin{bmatrix} \rho_1 r_1 & \cdots & \rho_n r_n \end{bmatrix} = MR. \tag{3.25}$$

Thus M is diagonalised by R, and we obtain the following spectral expansion.

$$M = R\Delta R^{-1} = \sum_{i=1}^{n} \rho_i \frac{r_i \tilde{r}_i{}^\dagger \gamma_5}{\tilde{r}_i^\dagger \gamma_5 r_i} \equiv \sum_{i=1}^{n} \rho_i P_i \tag{3.26}$$

One readily checks that the $\{P_i\}$ are a set of n orthogonal projections

$$P_i^2 = P_i, \qquad P_i P_j = P_i \delta_{ij} \tag{3.27}$$

so that $\sum_{i=1}^n P_i$ must be a resolution of the identity.

By considering Tr γ_5, one observes that

$$0 = \text{Tr } \gamma_5 = \sum_{i=1}^{n} v_i^\dagger \gamma_5 v_i \tag{3.28}$$

and

$$0 = \text{Tr } \gamma_5 = \text{Tr } R^{-1}\gamma_5 R = \sum_{i=1}^{n} \frac{\tilde{r_i}^\dagger r_i}{\tilde{r}_i^\dagger \gamma_5 r_i}. \tag{3.29}$$

This last result provides a powerful consistency check on a complete set of eigenvectors of M computed by a numerical method such as the one described in chapter 8.

3.2.3 The spectrum of M as a function of κ

Definition. A matrix A is *shifted* with respect to some scalar σ if it is of the form

$$A = \sigma I + B \tag{3.30}$$

and B is independent of σ.

Writing

$$\begin{aligned} M(\kappa) &= 1 - \kappa Z && (3.31) \\ Z &= \frac{1}{2} C_{SW} \sigma_{\mu\nu} F_{\mu\nu} + D, && (3.32) \end{aligned}$$

we see that the matrix M/κ has this property for $\sigma = 1/\kappa$. The matrix Q/κ is not so fortunate (the γ_5 spoils things completely). We now use this property to show that:–

- The eigenvectors of $M(\kappa)$ are independent of κ for fixed C_{SW}.
- The eigenvalues of $M(\kappa')$ are completely determined by the eigenvalues of $M(\kappa)$.

Let the eigenvalues $\rho_i(\kappa)$ satisfying

$$M(\kappa)r_i = \rho_i(\kappa)r_i \tag{3.33}$$

be ordered by their real part

$$\text{Re }\rho_1(\kappa) \leq \text{Re }\rho_2(\kappa) \leq \ldots \leq \text{Re }\rho_n(\kappa). \tag{3.34}$$

Then

$$Zr_i = \frac{1}{\kappa}(1 - M(\kappa))r_i = \frac{1-\rho_i(\kappa)}{\kappa}r_i \tag{3.35}$$

and

$$\begin{aligned} M(\kappa')r_i &= M(\kappa+\delta\kappa)r_i \\ &= (1-\kappa Z)r_i - \delta\kappa\frac{1-\rho_i(\kappa)}{\kappa}r_i \\ &= \frac{1}{\kappa}\left((\kappa+\delta\kappa)\rho_i(\kappa) - \delta\kappa\right)r_i \\ &= \rho_i(\kappa+\delta\kappa)r_i \\ &= \rho_i(\kappa')r_i, \end{aligned}$$

that is,

$$\rho_i(\kappa+\delta\kappa) = \left(1+\frac{\delta\kappa}{\kappa}\right)\rho_i(\kappa) - \frac{\delta\kappa}{\kappa}. \tag{3.36}$$

Thus varying κ serves only to translate and dilate the spectrum. Consequently only real eigenvalues can be shifted to zero provided κ remains real. Given $\rho_c(\kappa)$ real, we can make $\rho_c(\kappa+\delta\kappa)$ vanish for $\delta\kappa = \kappa\rho_c(\kappa)(1-\rho_c(\kappa))^{-1}$.

Furthermore, suppose that

$$\text{Re}\ \rho_j(\kappa) - \text{Re}\ \rho_i(\kappa) \geq 0, \qquad \kappa > 0, \qquad |\delta\kappa| < \kappa. \tag{3.37}$$

Then

$$\rho_j(\kappa + \delta\kappa) - \rho_i(\kappa + \delta\kappa) = \left(1 + \frac{\delta\kappa}{\kappa}\right)(\rho_j(\kappa) - \rho_i(\kappa)), \tag{3.38}$$

and consequently

$$\text{Re}\ \rho_j(\kappa + \delta\kappa) - \text{Re}\ \rho_i(\kappa + \delta\kappa) \geq 0. \tag{3.39}$$

From this we see that shifts in κ preserve the ordering of the eigenvalues.

More generally, we remark that the transformations

$$M[\kappa] \to M[\kappa + \delta\kappa] \tag{3.40}$$

$$M[\kappa] \to M[\kappa]^{-1} \tag{3.41}$$

$$M[\kappa] \to M[\kappa + \delta\kappa]^{-1} M[\kappa] \tag{3.42}$$

induce corresponding transformations on the eigenvalues, leaving the eigenvectors unchanged. These are Möbius transformations,

$$\rho \to \rho' = \frac{a + b\rho}{c + d\rho} \tag{3.43}$$

where a, b, c, d are complex, $ac \neq bd$.

3.2.4 Summary

$Q = \gamma_5 M$	M
$Qv_i = \lambda_i v_i$	$Mr_i = \rho_i r_i$
$(\lambda_i - \lambda_j)v_i^\dagger v_j = 0$	$(\rho_i^* - \rho_j)r_i^\dagger \gamma_5 r_j = 0$
λ_i is real	ρ_i^* is also an eigenvalue $M\tilde{r}_i = \rho_i^* \tilde{r}_i$
$v_i^\dagger Q = \lambda_i v_i^\dagger$	$\tilde{r}_i^\dagger \gamma_5 M = \rho_i \tilde{r}_i^\dagger \gamma_5$
$v_i^\dagger v_j = \delta_{ij}$	$\frac{\tilde{r}_i^\dagger \gamma_5 r_j}{\tilde{r}_i^\dagger \gamma_5 r_i} = \delta_{ij}$
$Q = \sum_{i=1}^{n} \lambda_i v_i v_i^\dagger$	$M = \sum_{i=1}^{n} \rho_i \frac{r_i \tilde{r_i}^\dagger \gamma_5}{\tilde{r}_i^\dagger \gamma_5 r_i}$
$\operatorname{csp}(\lambda_i) = 1$	$\operatorname{csp}(\rho_i) = \left\lvert \tilde{r}_i^\dagger \gamma_5 r_i \right\rvert^{-1}$
$\text{Tr } Q = \sum_{i=1}^{n} \lambda_i = \sum_{i=1}^{n} \rho_i \frac{\tilde{r}_i^\dagger r_i}{\tilde{r}_i^\dagger \gamma_5 r_i}$	$\text{Tr } M = \text{Tr } M^\dagger = \sum_{i=1}^{n} \rho_i = \sum_{i=1}^{n} \lambda_i v_i^\dagger \gamma_5 v_i$
$\sum_{i=1}^{n} v_i^\dagger \gamma_5 v_i = 0$	$\sum_{i=1}^{n} \frac{\tilde{r_i}^\dagger r_i}{\tilde{r}_i^\dagger \gamma_5 r_i} = 0$
$\lambda_i(\kappa + \delta\kappa) = ?$	$\rho_i(\kappa + \delta\kappa) = \frac{1}{\kappa}\left((\kappa + \delta\kappa)\rho_i(\kappa) - \delta\kappa\right)$
v_i depends on κ	r_i is independent of κ
$\det(Q) = \prod_{i=1}^{n} \lambda_i = \prod_{i=1}^{n} \rho_i = \det(M) = \det\left(M^\dagger\right)$	

Chapter 4

Linear systems I – point algorithms

4.1 Introduction

In this chapter, we are concerned with finding solutions ψ to linear systems of the form

$$M\psi = \eta. \tag{4.1}$$

The coefficient matrix M is the non-hermitian Wilson fermion matrix of chapter 3, but many of the observations made here will have broader applicability. If the source η is a unit vector in some basis, then the solution ψ may be interpreted as a column of the quark propagator M^{-1} in the same basis. In hadron spectroscopy it is usually the case that for each gauge configuration in the ensemble, the calculation must be repeated several times, eg. 12 times for point-like sources of spin and colour at the origin. We use the word "point" in "point algorithms" to distinguish these methods from block algorithms, the subject of the next chapter.

Even on moderately sized lattices, direct methods such as Gaussian elimination are simply not viable. The memory requirements (essentially one needs to store n^2 complex numbers because the zero components of M *fill in* during the calculation) are beyond the specifications of even the largest supercomputers. Even if rows of the matrix were swapped out to disk, the problem of controlling *round-off* error would be intractable. It is therefore necessary to exploit the sparsity of M. Iterative methods do this in a natural way.

Instead of working with M at the component level, iterative methods require a subroutine to implement the matrix-vector product $y = Mx$. This operation is defined unambiguously by the gauge fields and a few parameters (κ, C_{SW} and boundary conditions). Thus the memory requirements are greatly reduced, and the computational cost of the matrix-vector multiply scales linearly with the lattice volume (instead of quadratically as it would for a matrix-vector multiply involving an arbitrary full matrix).

Iterative methods are traditionally divided into two classes.

1. *Stationary* iterative methods are characterised by relations between successive approximations $\psi^{(k)}$ of the form

$$\psi^{(k+1)} = B\psi^{(k)} + c \tag{4.2}$$

   where both the matrix B and vector c are *constant*, independent of the iteration count k. This class includes *Jacobi*, *Gauss-Seidel*, *SOR* and *SSOR*.

2. Non-stationary iterative methods.

Generally speaking, the non-stationary methods outperform the stationary ones to such an extent that interest in the stationary methods is now primarily historical or pedagogical in nature. We will have nothing more to say on stationary methods except to remark that SSOR lives on as an effective *preconditioner* for non-stationary methods.

Part and parcel of any iterative method are convergence criteria. Ideally we would like to terminate when the kth approximation $\psi^{(k)}$ is close enough to the "true" solution $\psi^{(*)} = M^{-1}\eta$, ie. when $||\psi^{(*)} - \psi^{(k)}||$ is sufficiently small.[1] But $\psi^{(*)}$ is not available and so convergence criteria are usually based on the *residual* vector $r^{(k)} = \eta - M\psi^{(k)}$ or its norm $||r^{(k)}||$, sometimes called the *residuum*. Obviously, if $||r^{(k)}|| = 0$ then $\psi^{(k)}$ satisfies equation (4.1) and we say that the method has converged. In practice, however, we use finite-precision floating point arithmetic

[1]Unless otherwise stated, the norm $||z||$ of a vector z shall be the 2-norm $||z||_2 = \sqrt{z^\dagger z}$.

and we must be content with

$$||r^{(k)}|| \leq \mu\delta \tag{4.3}$$

for some cut-off δ and scale μ. δ should be chosen to ensure that the solution is sufficiently accurate for the purposes to which it will be put (ie. that the error due to imperfect convergence is significantly smaller than statistical errors). μ reflects the scale of the problem; we generally take $\mu = ||\eta||$.[2] It is usually pointless to take δ smaller than the machine epsilon appropriate to the precision used to represent $\psi^{(k)}$ and $\eta^{(k)}$, even if dot products are performed in a higher precision, because then the error in $||r^{(k)}||$ will exceed $||r^{(k)}||$ itself, and subsequent iterations will not improve the solution.

The error in the solution is related to the residual by

$$\psi^{(*)} - \psi^{(k)} = M^{-1} r^{(k)} \tag{4.4}$$

and consequently when the convergence criterion (4.3) is satisfied the error is also bounded.

$$||\psi^{(*)} - \psi^{(k)}|| \leq \max_{\lambda \in \sigma(M)} \frac{1}{|\lambda|} \mu\delta \tag{4.5}$$

Here $\sigma(M)$ denotes the eigenvalue spectrum of M. This bound is considerably weaker than the bound on the residuum, especially near the chiral limit where M develops very small eigenvalues.

The remainder of the chapter is structured as follows. Section §(4.2) gives a brief introduction to the subject of preconditioning. We see that red-black preconditioning, now the traditional choice in lattice QCD and the one adopted here, can be arrived at in a number of ways. We note that when the Clover action is used, there are two alternative forms of red-black preconditioning. The first results in a γ_5-symmetric coefficient matrix and this is the form used in all numerical tests in this chapter. The second does not preserve the γ_5-symmetry of the coefficient

[2] A sensible alternative, often used by numerical analysts, is $\mu = ||r^{(0)}||$. Of course, $||r^{(0)}||$ and $||\eta||$ coincide when the initial guess is the zero vector. Some groups use $\mu = ||\psi||$. This must be recomputed at the end of each iteration.

matrix but does turn out to be faster in conjunction with algorithms that do not depend on γ_5-symmetry; we shall return to this point in the context of optimising Generalised Hybrid Monte Carlo simulations in chapter 6.

Section §(4.3) describes several of the more popular and successful iterative methods which have been applied to propagator calculations in lattice QCD. In the interests of brevity, I do not go into the underlying theory in any great detail. The reader may refer to the textbook [19] for a relatively accessible introduction to the theory of conjugate gradient methods, or to the book [20] for a more pragmatic survey of a wide variety of methods and a comprehensive set of references, or to the excellent review [21] of the state of the art of propagator calculations in lattice QCD.

Section §(4.4) discusses some of the issues involved in implementing and testing these methods. It will be seen that there is much in common in the overall structure of the various methods, and in the kinds of elementary operation that each performs. Consequently many of the computational "building blocks" were already available to me in the form of subroutines bequeathed by my predecessors [22, 23].

In section §(4.5) I compare the performance of these algorithms using realistic configurations and parameters.

In section §(4.6) I consider the case of exceptional configurations. By "exceptional configuration" I mean operationally that the corresponding fermion matrix develops a very small eigenvalue at a valence κ below the critical value κ_c of the ensemble. These are a problem in the quenched approximation, especially at large lattice volumes and large C_{SW}. I describe a method based on the idea of projecting out the eigenvector corresponding to the eigenvalue of smallest magnitude. Although this method converges in cases where MR, BiCG(γ_5), and BiCGSTAB fail miserably, it turns out that QMR(γ_5) is the method of choice.

In section §(4.7) I make some concluding remarks.

4.2 Preconditioning

The convergence rate of iterative methods depends on the spectral properties of the coefficient matrix M. The idea of preconditioning is to transform the original system to one with more favourable spectral properties. Thus a solution ψ to equation (4.1) also satisfies

$$V_1^{-1} M V_2^{-1} V_2 \psi = V_1^{-1} \eta \tag{4.6}$$

for any non-singular V_1, V_2. We identify $\tilde{M} = V_1^{-1} M V_2^{-1}$, $\tilde{\psi} = V_2 \psi$, $\tilde{\eta} = V_1^{-1} \eta$ and work with the transformed system

$$\tilde{M} \tilde{\psi} = \tilde{\eta}, \tag{4.7}$$

recovering $\psi = V_2^{-1} \tilde{\psi}$ at the end. The better that $V = V_1 V_2$ approximates M, the better the preconditioner. Sometimes a distinction is drawn between *left* preconditioning in which $V_2 = I$, *right* preconditioning in which $V_1 = I$ and *symmetric* (or *central*) preconditioning in which $V_1 = V_2$, but it is often useful to refer to V as *the* preconditioner and to discuss its properties without specifying the details of its factorisation. The residuals of the original and preconditioned systems are related by

$$r^{(k)} = \eta - M \psi^{(k)} = V_1 \left(\tilde{\eta} - \tilde{M} \tilde{\psi} \right) = V_1 \tilde{r}^{(k)}. \tag{4.8}$$

To be useful, the overhead of multiplying by V_1^{-1} and V_2^{-1} must be offset by a reduction in the number of iterations required for convergence.[3]

The choice of preconditioner is influenced by the requirement of scalability on parallel computers. This rules out some otherwise excellent preconditioners.

[3]Often the explicit form of V_1^{-1} and V_2^{-1} is not known, and one must instead solve equations of the form $V_* x = b$.

4.2.1 Polynomial preconditioning

Preconditioning with V^{-1} a polynomial in M inherits the parallelisation properties of the underlying matrix-vector product. When used in conjunction with a Krylov subspace method, polynomial preconditioning can be expected to require at best the same number of multiplications by M as the unpreconditioned system. To see this, take the view that a Krylov subspace method generates a polynomial in M which is a near optimal reductor for $r^{(0)}$. However, the polynomial preconditioner can reduce the number of iterations and hence the number of vector-vector operations. Practical experience indicates that polynomial preconditioning equation (4.1) usually does not lead to a reduction in wall-clock time. Nonetheless, it has been reported that 4th-order polynomial preconditioning extends the viability of MR to lighter masses, at a small cost in wall-clock time in the region where both 4th-order polynomial and red-black preconditioning converge[24].

4.2.2 Jacobi preconditioning

Jacobi preconditioning (or *diagonal scaling*) is simply preconditioning with $V = (v_{ij})$ equal to the diagonal of the coefficient matrix $M = (m_{ij})$, ie.

$$v_{ij} = m_{ii}\delta_{ij}.$$

Block Jacobi preconditioning is the obvious generalisation to preconditioning by diagonal blocks of M.

4.2.3 D-ILU

This is an incomplete factorisation method in which only the diagonal elements are altered. In the special case in which the coefficient matrix can be written as $I + L + U$ with L (U) strictly lower (upper) triangular, the preconditioner is simply $V = (I + L)(I + U)$.

4.2.4 Red-black preconditioning

Red-black (or even-odd) preconditioning has long been the *de facto* standard in lattice QCD.

Working in a basis where all even sites appear before all odd sites induces a block-structure on M, and equation (4.1) becomes

$$\begin{pmatrix} A_{ee} & -\kappa D_{eo} \\ -\kappa D_{oe} & A_{oo} \end{pmatrix} \begin{pmatrix} \psi_e \\ \psi_o \end{pmatrix} = \begin{pmatrix} \eta_e \\ \eta_o \end{pmatrix} \tag{4.9}$$

Left-multiplying equation (4.9) by

$$V_1^{-1} = I + \kappa D A^{-1} = \begin{pmatrix} I & \kappa D_{eo} A_{oo}^{-1} \\ \kappa D_{oe} A_{ee}^{-1} & I \end{pmatrix} \tag{4.10}$$

leads to the decoupled system

$$\begin{pmatrix} A_{ee} - \kappa^2 D_{eo} A_{oo}^{-1} D_{oe} & 0 \\ 0 & A_{oo} - \kappa^2 D_{oe} A_{ee}^{-1} D_{eo} \end{pmatrix} \begin{pmatrix} \psi_e \\ \psi_o \end{pmatrix} = \begin{pmatrix} \eta_e + \kappa D_{eo} A_{oo}^{-1} \eta_o \\ \eta_o + \kappa D_{oe} A_{ee}^{-1} \eta_e \end{pmatrix} \tag{4.11}$$

We can now choose either parity p (even or odd), and identifying

$$\tilde{M}_{pp} = A_{pp} - \kappa^2 D_{p\bar{p}} A_{\bar{p}\bar{p}}^{-1} D_{\bar{p}p} \tag{4.12}$$

$$\tilde{\eta}_p = \eta_p + \kappa D_{p\bar{p}} A_{\bar{p}\bar{p}}^{-1} \eta_{\bar{p}} \tag{4.13}$$

we can solve

$$\tilde{M}_{pp} \psi_p = \tilde{\eta}_p \tag{4.14}$$

on one subspace and reconstruct the other parity of the solution $\psi_{\bar{p}}$ from

$$\psi_{\bar{p}} = A_{\bar{p}\bar{p}}^{-1} \left(\eta_{\bar{p}} + \kappa D_{\bar{p}p} \psi_p \right). \tag{4.15}$$

This results in a significant reduction in memory utilisation and a speed-up of a factor of typically 2–3 in wall-clock time, due to the fact that $\tilde{M}_{pp}$ has a lower condition number than M. The costs of multiplying a vector by M or one parity by $\tilde{M}_{pp}$ are nearly identical.

Note that $\tilde{M}_{pp}$ retains the property of γ_5-symmetry. However, in contrast to the original matrix M, the preconditioned matrix is not proportional to a shifted matrix unless $C_{SW} = 0$, owing to the hidden κ-dependence in A^{-1}. It is for this reason that multiple-mass tricks [25, 26, 27] are much more attractive when $C_{SW} = 0$.

In exact arithmetic, the residua of the original and preconditioned system coincide. For example, solving on the even parity ($p = e$) we find after a simple calculation that

$$r = \eta - M\psi = \begin{pmatrix} \tilde{\eta}_e - \tilde{M}_{ee}\psi_e \\ 0 \end{pmatrix} = \begin{pmatrix} \tilde{r}_e \\ 0 \end{pmatrix} \tag{4.16}$$

In the finite precision that must be used in practice, the second equality begins to break down as the target residuum approaches the machine epsilon owing to cancellation errors. Using 32-bit precision for the fermion and gauge fields and a target residuum of $10^{-7}||\eta||$, one usually finds that $||r||$ is slightly larger than $||\tilde{r}_p||$.[4]

When $C_{SW} \neq 0$, it is sometimes worthwhile to apply a second block Jacobi preconditioner to the system (4.14), ie. left-multiply by A_{pp}^{-1} and work with the coefficient matrix

$$\tilde{\tilde{M}}_{pp} = I + \kappa^2 A_{pp}^{-1} D_{p\bar{p}} A_{\bar{p}\bar{p}}^{-1} D_{\bar{p}p} \tag{4.17}$$

instead, at the cost of destroying γ_5-symmetry. This alternative form of red-black preconditioning for the clover action may also be thought of as:

1. Block Jacobi in an even-odd basis with $V_1 = A$, $V_2 = I$, followed by D-ILU with $V_1 = I + L$, $V_2 = I + U$, $L = -\kappa A_{oo}^{-1} D_{oe}$, $U = -\kappa A_{ee}^{-1} D_{eo}$. The key

[4]A possible remedy is to restart using the full (unpreconditioned) matrix.

observation here is that

$$(I + L)^{-1} = I - L. \tag{4.18}$$

2. Block Jacobi in an even-odd basis with $V_1 = A$, $V_2 = I$, followed by polynomial preconditioning with $V_1^{-1} = I + \kappa A^{-1} D$, $V_2 = I$. Notice that V_1^{-1} is the approximation to $(I - \kappa A^{-1} D)^{-1}$ obtained by truncating its Neumann series at $O(\kappa)$.

3. SSOR preconditioning applied to the red-black ordering.

All of these methods lead to the same decoupled systems of equations. We will have more to say on this preconditioner in the context of accelerating Hybrid Monte-Carlo simulations in chapter 6.

4.2.5 LL-SSOR preconditioning

The recently introduced locally-lexicographic SSOR preconditioning, although beyond the present scope, has much to recommend it and it is somewhat apologetically that I refer the reader to the paper by Fischer *et. al.* [28] for a full treatment. My reason for not including LL-SSOR preconditioning in these studies is a pragmatic one: insufficient time. To implement LL-SSOR efficiently on parallel machines such as the Cray-T3D is known to be difficult, especially when the Clover action is used, and several man-years of effort had already gone into writing highly optimised matrix-vector multiply routines in assembly language for the Cray-T3D and Cray-T3E using the red-black preconditioned fermion matrix and Clover action.

4.3 Point Algorithms

The tests of iterative methods in this chapter were carried out using the γ_5-symmetric red-black preconditioning equation (4.14), except where explicitly indicated to the contrary. For convenience, I shall suppress for the remainder of this chapter all notational glyphs (such as $\tilde{\cdot}$) which distinguish between the original and preconditioned objects.

4.3.1 Over-relaxed minimal residual

We start with minimal residual (MR) because it is the simplest of all non-stationary iterative methods, can be applied for any non-singular coefficient matrix, and yet exhibits many generic features.

In each iteration of MR, the solution is updated in the direction of the most recent residual. When the over-relaxation parameter $\omega = 1$, the distance the solution moves along this direction is such that the next residual is minimised, and the algorithm is essentially a steepest descent method and liable to become trapped in a local minimum. When $\omega > 1$ ($\omega < 1$) we have over-relaxed (under-relaxed) Minimal Residual. In general, any particular linear system may have an optimal value of ω, not known *a priori*. However, the convergence rate of MR appears to be fairly insensitive to ω near its optimal value, and anything in the range $1.05 < \omega < 1.4$ is reasonable in QCD applications. We generally use $\omega = 1.1$[22].

The initial guess $\psi^{(0)}$ can be chosen freely. Our usual choice is $\psi^{(0)} = \eta$.[5]

In a practical implementation of MR, memory is allocated for only one solution vector and one residual vector, and the operations (4.22, 4.23) become saxpys. Additional storage is required for the work vector s. Once the initial residual $r^{(0)}$ has been set up, the source η is not required during the iteration, and this can be exploited to economise on memory usage, eg. by overlapping s with η. The price to pay is that it may be necessary to reconstruct η later (or even reload it from

[5]When available, a solution for the same source η and gauge configuration at a nearby κ will generally be a superior guess, saving a few iterations. The solution for a different source is usually inferior.

disk if smeared sources are used). This is the case (a) if red-black preconditioning is used, as the other parity of the source is needed to determine the other parity of the solution, or (b) if an independent check on the residual is desired, or (c) if the algorithm is to be restarted. As memory has become cheaper over recent years, the necessity of economising on memory usage has diminished somewhat, and in some cases (eg. fuzzed sources at heavy masses) the overhead of reloading the source from disk may be significant. A simple strategy to avoid the extra disk accesses is described in [29].

MR

```
$r^{(0)} = \eta - M\psi^{(0)}$                              (4.19)
for $k = 0, 1, 2, \ldots$ until convergence do {
    $s = Mr^{(k)}$                                           (4.20)
    $\alpha = s^\dagger r^{(k)} / s^\dagger s$               (4.21)
    $\psi^{(k+1)} = \psi^{(k)} + \omega\alpha r^{(k)}$       (4.22)
    $r^{(k+1)} = r^{(k)} - \omega\alpha s$                   (4.23)
}
```

4.3.2 Conjugate Gradient

The conjugate gradient (CG) method of Hestenes and Stiefel [30] requires an hermitian positive definite coefficient matrix. But for non-hermitian M, one must apply CG to one of the normal equations

$$\left(M^\dagger M\right) \psi = M^\dagger \eta \tag{4.24}$$

or

$$\left(M M^\dagger\right) \phi = \eta, \quad \psi = M^\dagger \phi. \tag{4.25}$$

The first form equation (4.24) is left preconditioning with $V_1^{-1} = M^\dagger$ and leads to a method that minimises the residuum. This is known as Conjugate Gradient on the Normal equation, least Residual formulation, and is usually abbreviated CGNR.

The second form equation (4.25) is right preconditioning with $V_2^{-1} = M^\dagger$, and leads to a method which minimises the norm of the error $||\psi^{(*)} - \psi^{(k)}||$. As this quantity is not computable, convergence criteria must still be based on the residuum. This is known as Conjugate Gradient on the Normal equation, least Error formulation, and is usually abbreviated CGNE. Simpson [22] found numerically that, iteration by iteration, CGNE delivers an iterate marginally closer[6] to the true solution than does CGNR (as is only to be expected) despite a considerably larger residuum. As our central purpose in this chapter and the next is to compare the performance of quite different algorithms, it behoves us to compare like with like and so we use CGNR, without further apology.

In exact arithmetic, the convergence of Conjugate Gradient in a fixed number of iterations[7] is guaranteed provided only that the coefficient matrix is non-singular. Transforming the original system to one of the normal equations is at first sight highly attractive. The price to pay is that both methods square the condition number of the system. Another way to see that the normal equation approaches are likely to have an adverse effect on the conditioning of the system is to regard $(M^{-1})^\dagger = \left(M^\dagger\right)^{-1} \equiv M^{-\dagger}$ as a preconditioner and to observe that $M^\dagger$ is a very poor approximation to M^{-1}.

Before leaving the subject of Conjugate Gradient, it is worth explaining why this method is so special. Conjugate Gradient is an example of an iterative method which produces approximate solutions that

1. are characterised by a (global) minimisation property over Krylov subspaces generated by the coefficient matrix (minimisation)

[6]in the 2-norm

[7]equal to the order of the coefficient matrix

2. can be computed with little work per iteration (short recurrences).

The class of methods having both of these properties are sometimes called *conjugate gradient methods.* That Conjugate Gradient itself has the minimisation property is due to the fact that an hermitian positive definite matrix A induces a norm $||.||_A$ through

$$||v||_A = \sqrt{v^\dagger A v}. \tag{4.26}$$

Although conjugate gradient methods do exist for coefficient matrices which are not hermitian positive definite, the fermion matrix M is not so blessed. Necessary and sufficient conditions for the existence of a conjugate gradient method are given in [31].

The remaining methods considered in this chapter work with M as the coefficient matrix. Motivated by the fact that lattice computations are memory-limited due to the large size of the objects involved, we focus on methods which preserve the property of short recurrences at the expense of minimisation. In some cases, it is possible to introduce a relaxed version of the minimisation property; for example, QMR minimises something (but it is neither the norm of the error nor the norm of the residual), and BiCGSTAB incorporates a local minimisation of the residuum over the last two search directions.

An example of a method which preserves global minimisation at the expense of short recurrences is GMRES. It is therefore optimal in terms of matrix multiplications amongst algorithms working within the same Krylov subspace. On large and difficult problems, GMRES will exhaust the available memory before convergence is reached and it becomes necessary to restart the method using the latest solution as the initial guess. Thus GMRES(k) stores at most k previous search directions. Also, the required number of vector-vector operations per iteration increases with the iteration number, and eventually the cost of vector-vector operations dominates over matrix-vector multiplies. Others, eg. [32, 21], have found that in QCD applications BiCGSTAB typically converges in a number of matrix multiplies comparable to that of even full GMRES (GMRES(∞)). Assuming the optimality of GMRES(∞) they conclude that, there being no room for major improvements from choice or design of algorithm, research in this field is essentially mature.[8]

4.3.3 Bi-Conjugate Gradient

The Bi-Conjugate Gradient algorithm (BiCG) of Lanczos [33] and Fletcher [34] may be viewed as a generalisation of CG to non-hermitian, indefinite systems. As we shall be studying a more general version of the algorithm in chapter 5, we shall not dwell on the details of the algorithm here. However, a few remarks are in order.

BiCG constructs five sequences of vectors, $x^{(k)}$ (the approximate solutions), $r^{(k)}$ (the usual residual), $\bar{r}^{(k)}$, $p^{(k)}$, $\bar{p}^{(k)}$, satisfying

$$r^{(k)} = \eta - Mx^{(k)}, \tag{4.27}$$

$$\left(\bar{r}^{(i)}\right)^\dagger r^{(j)} = \left(r^{(i)}\right)^\dagger \bar{r}^{(j)} = 0 \qquad \forall j < i, \tag{4.28}$$

(*bi-orthogonality*), and

$$\left(\bar{p}^{(i)}\right)^\dagger M p^{(j)} = \left(p^{(i)}\right)^\dagger M^\dagger \bar{p}^{(j)} = 0 \qquad \forall j < i \tag{4.29}$$

(*biconjugacy*). The kth residual $r^{(k)}$ is constructed to lie within the Krylov subspace

$$K_k\left(M, r^{(0)}\right) = \text{span}\{r^{(0)}, Mr^{(0)}, \ldots, M^{k-1}r^{(0)}\}. \tag{4.30}$$

Similarly,

$$\bar{r}^{(k)} \in K_k\left(M^\dagger, \bar{r}^{(0)}\right). \tag{4.31}$$

Like CG, BiCG can be implemented using short recurrences, but unlike CG, it does not minimise the norm of the residual vector. However, assuming exact arithmetic, BiCG must terminate in $\nu \leq n$ iterations. If $\nu = n$, $r^{(\nu)}$ will be

[8]This does not rule out the possibility of accelerating propagator calculations through other avenues such as preconditioning, block algorithms or multi-grid.

orthogonal to a space of dimension n and hence must vanish. Premature termination, due to the breakdown of the underlying Lanczos process, is possible. The symptom is that the next iteration calls for division by zero. Exact breakdowns are extremely rare in practice, but a near breakdown can lead to fatal loss of precision.

The convergence history of $||r^{(k)}||$ is far from monotonic, typically showing local fluctuations of several orders of magnitude. Nonetheless, BiCG is less susceptible than MR to critical slowing down as $\kappa \to \kappa_c$.

This erratic convergence can lead to numerical problems. Using 64-bit precision alleviates these in most cases, but does not address the underlying cause.

We also remark that restarting BiCG is usually undesirable. Following a restart the solution will pick up components in directions previously eliminated by biorthogonality, and, since BiCG lacks any minimisation property, a large number of iterations may be required before the solution begins to improve. It may be better to use a different algorithm to "polish" the solution.

For the specific *choice* $\bar{r}^{(0)} = \gamma_5 r^{(0)}$, it turns out that $\bar{r}^{(k)} = \gamma_5 r^{(k)}\ \forall k$ and similarly $\bar{p}^{(k)} = \gamma_5 p^{(k)}\ \forall k$. The proof, by induction, relies heavily on the γ_5-symmetry of M. This trick, pointed out in [35, 25], means that vectors $\bar{r}^{(k)}$ and $\bar{p}^{(k)}$ can be eliminated completely, as can all multiplications by $M^\dagger$. Moreover, all the coefficients become real, which further reduces the cost of the remaining vector-vector operations. As there is no *a priori* reason to suppose that the choice $\bar{r}^{(0)} = \gamma_5 r^{(0)}$ should have any adverse affect on the convergence of the algorithm, it is fair to say that the trick reduces the computational effort of BiCG by a factor of 2. We call the resulting algorithm BiCG(γ_5).[9]

Storage is required for the vectors $\psi^{(k)}, \eta, r^{(k)}, p^{(k)}, t$. As for MR and CGNR, η is not required during the iteration.

[9]Early studies of BiCG in the context of fermion matrix inversions [36] did not make use of this trick but still found regimes in which BiCG outperformed MR and CG.

BiCG(γ_5)

```
$r^{(0)} = p^{(0)} = \eta - M\psi^{(0)}$
$\alpha_0 = \left(r^{(0)}\right)^\dagger \gamma_5 r^{(0)}$
for $k = 0, 1, 2, \ldots$ until convergence do {
    $t = Mp^{(k)}$
    $\delta = \alpha_k / \left(p^{(k)}\right)^\dagger \gamma_5 t$
    $\psi^{(k+1)} = \psi^{(k)} + \delta p^{(k)}$
    $r^{(k+1)} = r^{(k)} - \delta t$
    $\alpha_{k+1} = \left(r^{(k+1)}\right)^\dagger \gamma_5 r^{(k+1)}$
    $p^{(k+1)} = r^{(k+1)} + (\alpha_{k+1}/\alpha_k)\, p^{(k)}$
}
```

4.3.4 Quasi-Minimal Residual

The Quasi-Minimal Residual method of Freund and Nachtigal [37] was formulated in an effort to overcome the problem of breakdowns in BiCG. Like BiCG, QMR is based on the non-hermitian Lanczos process, and the two methods are intimately related.[10] The Lanczos vectors themselves, here denoted $v^{(k)}$ and $w^{(k)}$, appear explicitly in QMR but not in BiCG.

In its full version, QMR employs a look-ahead Lanczos algorithm [38] to avoid all such breakdowns except the so-called "incurable" ones. The look-ahead Lanczos algorithm relaxes the bi-orthogonality condition between the two Lanczos sequences whenever the standard algorithm would break down. Although the computational cost of a look-ahead step is approximately the same as that of

[10]In fact, the BiCG iterates and residuals can be recovered from the QMR ones.

a standard iteration, there is a potentially large memory overhead: in essence, previous Lanczos vectors must be remembered for as many iterations as are necessary to "cure" the breakdown. In this work we consider only a simpler version of QMR without look-ahead. Even without look-ahead, QMR offers certain advantages over BiCG: its smoother convergence history places less demands on the numerical bandwidth of the supporting architecture.

QMR constructs its approximate solutions in the Lanczos basis and imposes a 'quasi'-minimisation on the residual.[11] This uses the QR decomposition and can be updated cheaply, ie. without sacrificing short recurrences.

The γ_5-symmetry of M can be exploited to eliminate the multiplication by $M^\dagger$ in each iteration, as well as reducing the memory costs. One *chooses* $w^{(0)} = \gamma_5 v^{(0)}$ then proves by induction that $w^{(k)} = \gamma_5 w^{(k)}, \; \forall k$. We call the resulting algorithm QMR(γ_5).

Note that QMR does not provide a recurrence for the residual vector. However, an upper bound on $||r^{(k)}||$ in terms of $||r^{(0)}||$, k and $s_1, \ldots, s_k$ was established in [37]. It is easy to formulate an alternative stopping criterion based on this bound, and to explicitly calculate the residual (requiring an additional matrix-vector multiplication) only after the bound is small enough.[12] Occasionally, when working in 32-bit precision with an ambitious target residue, it was observed that, approaching convergence, the true residual had a norm larger than that of its theoretical upper bound. This was interpreted as a symptom of accumulated numerical problems (round-off and loss of bi-orthogonality), and the code was modified to force a restart under these conditions. However, even with this modification, the method sometimes stagnates with $||r^{(k)}||$ one or two orders of magnitude larger than the machine epsilon. It is for this reason that I recommend that QMR be used in 64-bit precision for quark propagator calculations.

[11]The quasi-minimisation idea has recently been applied to a host of algorithms, eg. QMR-CGS and QMRCGSTAB(k) which can be found in the literature.

[12]In my implementation, I compute the true residual once the upper bound is within an order of magnitude of the convergence target for $||r^{(k)}||$, and I find typically that only a few additional iterations are required.

QMR(γ_5)

```
$r^{(0)} = \tilde{v}^{(0)} = \eta - M\psi^{(0)}$
$\mu = \rho = ||r^{(0)}||$
$p^{(-1)} = p^{(-2)} = v^{(-1)} = 0;\ v^{(0)} = \tilde{v}^{(0)}/\rho$
$\delta_{-1} = c_{-1} = c_0 = 1;\ s_{-1} = s_0 = 0$
for $k = 0, 1, 2, \ldots$ until convergence do {
    $\delta_k = \left(v^{(k)}\right)^\dagger \gamma_5 v^{(k)}$
    $t = Mv^{(k)}$
    $\alpha = \left(v^{(k)}\right)^\dagger \gamma_5 t/\delta_k$
    $\beta = \rho\delta_k/\delta_{k-1}$
    $\tilde{v}^{(k+1)} = t - \alpha v^{(k)} - \beta v^{(k-1)}$
    $\rho = ||\tilde{v}^{(k+1)}||;\ v^{(k+1)} = \tilde{v}^{(k+1)}/\rho$
    $\theta = s_{k-1}\beta;\ \gamma = c_{k-1}\beta$
    $\epsilon = c_k\gamma + s_k\alpha;\ \xi = -s_k^*\gamma + c_k\alpha$
    $\nu = \sqrt{|\xi|^2 + |\rho|^2}$
    $c_{k+1} = |\xi|/\nu$
    $s_{k+1}^* = (\xi\ == 0\ ?\ 1\ :\ c_{k+1}\rho/\xi)$        (4.32)
    $\chi = c_{k+1}\xi + s_{k+1}\rho$
    $p^{(k)} = (v^{(k)} - \epsilon p^{(k-1)} - \theta p^{(k-2)})/\chi$
    $\psi^{(k+1)} = \psi^{(k)} + c_{k+1}\mu p^{(k)}$
    $\mu = -s_{k+1}^*\mu$
}
```

Equation (4.32) is written using the ternary operator (? :) borrowed from the C

programming language. The digit 1 preceding the colon was incorrectly printed as 0 in reference [25].

4.3.5 Stabilised Bi-Conjugate Gradient

So far in our treatment of Krylov-subspace methods we have focused on recurrence relations between vectors. From a theoretical point of view it is sometimes more fruitful to view these methods in terms of recurrence relations between polynomials. For example, the statement that the BiCG residual lies in the Krylov-subspace generated by M and $r^{(0)}$ may be written in the form

$$r^{(k)} = \rho_k(M)\, r^{(0)} \tag{4.33}$$

where $\rho_k(M)$ is a polynomial in M of degree $k-1$ satisfying $\rho_k(0) = 1$. Statements about the convergence properties of an algorithm translate into statements about the efficiency of their residual polynomials as reductors acting on $r^{(0)}$.

Sonneveld [39] devised an algorithm (Bi)Conjugate Gradient Squared (CGS), in which the residual polynomial of BiCG is applied twice

$$r^{(k)}_{CGS} = \rho_k(M)^2\, r^{(0)}. \tag{4.34}$$

An attractive feature of this method is that, in each iteration, the multiplication by $M^\dagger$ in BiCG is eliminated in favour of an additional multiplication by M, which is used to expand the Krylov subspace.[13] CGS often (but not always) converges significantly faster than BiCG. However, due to the squaring of the residual polynomial, the erratic convergence behaviour of BiCG is exacerbated in CGS.

A drawback of CGS is that, even if $\rho_k(M)$ is a good reductor for $r^{(0)}$, it does not

[13]The literature of numerical analysis often places a high premium on transpose-free methods in which multiplications by the transpose (or hermitian conjugate) of the coefficient matrix are avoided, as these operations are often difficult to implement efficiently. For us, however, multiplication by $M^\dagger$ is no more difficult than multiplication by M, and we have equally efficient subroutines for both.

necessarily follow that $\rho_k(M)$ will be a good reductor for $\rho_k(M)\,r^{(0)}$.

Van der Vorst [40], following a suggestion by Sonneveld, devised another method, Stabilised Bi-Conjugate Gradient (BiCGSTAB), in which the residual polynomial of CGS $\rho_k(M)^2$ is replaced by the product $\rho_k(M)\,\tau_k(M)$. The polynomial τ_k is built up in factored form

$$\tau_k(\xi) = (1-\chi_0\xi)(1-\chi_1\xi)\dots(1-\chi_{k-1}\xi) \tag{4.35}$$

where a new zero $1/\chi_m$ is added in the mth iteration, corresponding to a 1-dimensional minimisation of $||r^{(m)}||$. It is for this reason that BiCGSTAB is sometimes described as a hybrid method, the product of BiCG and restarted GMRES(1).

BiCGSTAB was subsequently extended to complex arithmetic by Gutknecht [41], and it is his version that we use in the present work. Earlier numerical studies in the context of lattice QCD can be found in [42, 32].[14]

A detailed analysis of the relationships between the various sequences of orthogonal polynomials would take us too far afield, so I content myself with setting out the algorithm and making a few general remarks. For variety, I have suppressed iteration numbers wherever possible in writing down the algorithm. Iteration numbers are convenient for discussing the theoretical properties of an algorithm, but get in the way when designing a computer implementation; here one is more concerned with minimising storage and memory-memory copies.

Storage is required for the vectors $\psi, \eta, r_0, r, v, p, s, t$. A significant improvement over the naïve implementation is achieved by overlapping the vectors r and s in memory, thus saving storage for one vector, and reducing the floating point operations count. Although this was pointed out in the original paper by Van der Vorst [40], it does not appear to have been widely exploited in lattice implementations. As usual, the source η is not required during the iteration, and if

[14]A thorough numerical comparison of CGNR, BICGSTAB and BiCGSTAB2 with Wilson fermions in compact lattice QED was reported in [43].

necessary the memory allocated to η can be reused by another vector, eg. r.

BiCGSTAB

```
$r = r^{(0)} = \eta - M\psi$
$v = p = 0$
$\rho_0 = \alpha = \omega = 1$
for $k = 0, 1, 2, \dots$ until convergence do {
    $\rho_{k+1} = \left(r^{(0)}\right)^\dagger r$
    $\beta = (\rho_{k+1}/\rho_k)(\alpha/\omega)$
    $p = r + \beta(p - \omega v)$
    $v = Mp$
    $\alpha = \rho_{k+1}/\left(r^{(0)}\right)^\dagger v$
    $s = r - \alpha v$
    $t = Ms$
    $\omega = t^\dagger s / t^\dagger t$
    $\psi = \psi + \omega s + \alpha p$
    $r = s - \omega t$
}
```

Note that one iteration of BiCGSTAB involves two multiplications by M.

BiCGSTAB represents a considerable advance over BiCG. The local minimisations leads to a much smoother convergence than in BiCG, with a concomitant reduction in cancellation errors. Hence BiCGSTAB can generally achieve a smaller residuum than can BiCG using arithmetic of the same precision. Furthermore, the local minimisations make restarting with BiCGSTAB a viable proposition.

However, BiCGSTAB will break down whenever BiCG does.

It is only when BiCG can be applied in its γ_5-symmetric form BiCG(γ_5) that the computational effort required becomes comparable to that of BiCGSTAB. BiCGSTAB of course has the advantage of being applicable to systems that do not possess this symmetry.

4.3.6 Another Stabilised Bi-conjugate Gradient

If BiCGSTAB is viewed as the product of BiCG and GMRES(1), it is natural to expect that products of BiCG with GMRES(k) should exist. Gutknecht [41] derived such a method, called by him BiCGSTAB2, which replaces the 1-dimensional minimisations of BiCGSTAB with 2-dimensional ones. Gutknecht argued that BiCGSTAB2 should be particularly beneficial for real non-symmetric matrices with complex spectrum.

The algorithm given below is entirely equivalent to Gutknecht's, but I have suppressed unnecessary iteration numbers and re-ordered operations to simplify the coding task. It is immediately apparent that both the storage requirements and the number of vector-vector operations per iteration are greatly increased with respect to BiCGSTAB. Consequently, if BiCGSTAB2 is to be preferred to BiCGSTAB, a compensating reduction in matrix-multiplications must be forthcoming. In all my tests I found this not to be the case; the small reduction in the number of iterations was never sufficient to offset the increased work per iteration, and BiCGSTAB2 invariably consumed more wall-clock time than BiCGSTAB.

Storage is required for the vectors $\psi^{(k+1)}$, $\psi^{(k)}$, $\psi^{(k-1)}$, $p^{(k+1)}$, $p^{(k)}$, $p^{(k-1)}$, q, $r^{(0)}$, r, s, t, u, v, x. $\psi^{(k+1)}$ may be overlapped with $\psi^{(k-1)}$, as may $p^{(k+1)}$ with $p^{(k-1)}$, and u with r.

BiCGSTAB2

```
$r = r^{(0)} = p^{(0)} = \eta - M\psi^{(0)};\ \rho_0 = ||r^{(0)}||^2$
for $k = 0, 1, 2, \ldots$ until convergence do {
    if ($k$ is odd) { $q = s + \beta p^{(k-1)};\ x = t + \beta v$}
    $v = Mp^{(k)}$
    $\alpha_k = \rho_k / \left(r^{(0)}\right)^\dagger v$
    if ($k$ is odd) { $x = s - \alpha_k x$ }
    $s = r - \alpha_k v$
    $t = Ms$
    if ($k$ is even) {
        $\omega = t^\dagger s / t^\dagger t$
        $\psi^{(k+1)} = \psi^{(k)} + \alpha_k p^{(k)} + \omega s$
        $r = s - \omega t$
        $\rho_{k+1} = \left(r^{(0)}\right)^\dagger r$
        $\beta = (\rho_{k+1}/\rho_k)(\alpha_k/\omega)$
        $p_{k+1} = r + \beta(p^{(k)} - \omega v)$
    } else {
        compute $\xi, \chi$ which depend on $(s - x), t, x$
        $\psi^{(k+1)} = (1-\xi)\left[\psi^{(k-1)} + \alpha_{k-1}p^{(k-1)} + \alpha_k q\right] + \xi\left[\psi^{(k)} + \alpha_k p^{(k)}\right] - \chi s$
        $r = (1-\xi)x + \xi s + \chi t$
        $\rho_{k+1} = \left(r^{(0)}\right)^\dagger r;\ \beta = -(\rho_{k+1}/\rho_k)(\alpha_k/\chi)$
        $p^{(k+1)} = r + \beta\left[(1-\xi)q + \xi p^{(k)} + \chi v\right]$
    }
}
```

4.4 Implementation and testing

It is appropriate to preface this section by some historical remarks. When I started this work, the UKQCD collaboration had an existing solver code which ran on various architectures, including the Cray-T3D, Cray-YMP and DEC-Alpha workstations. The only algorithm supported in the original code was MR. Thus there was already available to me a trusted framework into which I could insert new subroutines implementing different algorithms provided only that I kept the interfaces plug-compatible. At the same time, others were attempting to improve the performance of the solver application. One strand was parallelisation of input and output. More major was the optimisation of the matrix-vector multiplication for the Cray-T3D (and later Cray-T3E) by rewriting key subroutines in assembly language. A further complication was the advent of a Fortran-90 compiler on the Cray-T3D, which made it possible to store vectors in 32-bit fields, potentially an important optimisation as memory bandwidth is a major limiting factor on the Cray-T3D and Cray-T3E. These and other changes meant that there was an ongoing need to reintegrate updated modules into the solver code and to retest the latest version, a task which generally fell to me.

It will be appreciated that developing code in such a shifting environment demands a firm adherence to existing styles, standards and interface specifications; changes with global impact are usually best avoided. I therefore postponed some changes, such as an overhaul of the parameter file reader and the ability to specify the precision of external files at run-time, until after the code had settled down.[15]

4.4.1 Implementation

Examining the operations performed in each algorithm reveals a large degree of commonality. The operation which requires the bulk of the computational effort in each iteration is that of multiplying a vector by the red-black preconditioned fermion matrix, while vector-vector operations account for most of the remainder. Compared to these, the cost of control logic and operations involving only scalars

[15]Details of these changes may be found in [29].

is entirely negligible.

I do not give details here of the implementation of the matrix-vector multiplication nor of the geometric decomposition of the lattice over processor elements on a parallel machine; these are adequately covered in the documentation of the UKQCD MPP codes, and useful background is provided in [23, 22]. For our purposes it suffices that there exist trusted subroutines which multiply a vector by the fermion matrix or its hermitian conjugate, and similar ones for the red-black preconditioned fermion matrix. The reader should be aware that these subroutines involve significant inter-processor communications.

Before discussing vector-vector operations in detail, it is worth describing how each vector is laid out in memory. We do this with reference to the Fortran declaration of a fermionic field taken from the solver code.

```
CFTRANS psi :I :I :I :site :
      Fpoint psi(0:Ncomplex-1,0:Ncolour-1,0:Nspin4-1,
     $ 0:Max_body-1,0:Npar-1)
```

Here `Fpoint` is a macro which is expanded by the C-preprocessor to a standard Fortran declaration determining the precision of each array element; this may be either 32-bit or 64-bit but the exact form of the Fortran declaration will vary according to the compiler and platform.[16] `Ncomplex`, `Ncolour`, `Nspin4`, and `Npar` are compile-time constants, taking respectively the values 2 (a complex number is represented by an ordered pair of reals),[17] 3 (number of colours), 4 (number of components in a 4-spinor) and 2 (number of parities – even or odd); these

[16]Despite the fact that the Cray-T3D, Cray-T3E and DEC workstations all use DEC Alpha microprocessors, single and double precisions have different meanings. Thus on the DEC workstations, REAL and DOUBLE PRECISION respectively declare 32-bit and 64-bit floating point numbers, whereas on Cray systems REAL declares a 64-bit object and 32-bit objects may be declared as REAL(KIND=4). The same distinction is seen in the naming conventions of BLAS routines: the Cray equivalents of the workstation SAXPY and DAXPY are respectively HAXPY and SAXPY.

[17]This representation compels the programmer to keep track of real and imaginary parts manually instead of delegating these aspects to the compiler. Historically, the reason for this decision was not for any mistrust of compilers' ability to handle complex arithmetic (justified

compile-time constants serve a purely documentary purpose and should not be altered – thus setting `Ncolour` to 2 will not change the gauge group to $SU(2)$, but rather introduce a host of inconsistencies into a code which is dedicated to $SU(3)$. The reason that all sites of even parity appear before all sites of odd parity is because of a commitment to red-black preconditioning for which this ordering is convenient and efficient.[18]

Fermion fields are distributed over processor elements using the same geometric decomposition as other lattice fields. Thus the compile-time constant `Max_body` is derived from other compile-time parameters (which can be changed) and determines the maximum number of sites of a single parity on the local lattice; it usually includes some additional padding intended to reduce cache conflicts in operations involving two fermion fields.

The line commencing with CFTRANS in column 1 is a directive to Stephen Booth's utility `ftrans` to allow re-ordering of array indices according to a command-line option after C preprocessing and before invoking the Fortran compiler; in this instance the directive instructs `ftrans` that the fourth index `site` can be moved to the front. On a vector supercomputer it is usually preferable to have the `site` index varying fastest; the ordering with `site` varying slowly is the one adopted on Cray-T3D and Cray-T3E systems.

These same fermion fields (or one parity thereof if red-black preconditioning is in use) play the rôle of vectors in our iterative methods, the ordering of indices determining the basis. From the preceding comments it is apparent that the components of each vector are not necessarily laid out contiguously in memory, a fact which must be borne in mind when considering the use of BLAS routines to implement a vector-vector operation.

The dominant storage requirements of the algorithms considered here, are (after

though this might have been), but rather because DOUBLE COMPLEX was not part of the Fortran77 standard.

[18]Not surprisingly, this ordering is neither convenient nor efficient for LL-SSOR preconditioning.

the gauge fields through which matrix multiplication is defined) the vectors themselves; these are listed in table 4.1. As memory must be allocated at a higher level in the call tree for both parities of the source η and solution ψ, it is generally true that a subroutine performing a solve with the red-black preconditioned matrix can usurp this memory for four one-parity vectors, at the cost of overwriting the source. In this situation, the memory requirements of CGNR and BiCG(γ_5) are effectively no worse than that of MR. The memory requirements given for QMR include an allowance for work vectors to compute the equation residual in the last few iterations. BiCGSTAB2 is by far the most expensive in memory terms; I can seen no way to reduce the number of vectors below 11.

Algorithm	Vectors
MR	3
CGNR	4
BiCG(γ_5)	4
BiCGSTAB	6
BiCGstab2	11
QMR(γ_5)	8

Table 4.1: Vector storage requirements, excluding η, for various point algorithms.

In principle, all the vector-vector operations required here could be implemented in terms of some minimal set of elementary operations which would include little more than scalar multiplication, vector addition and an inner product. Doing so would be grossly sub-optimal. To sustain anything approaching the peak performance of the Cray-T3D the programmer must arrange to keep the pipe-line full, which in turn requires a careful consideration of how memory loads and stores are scheduled – memory speed is usually the limiting factor, and minimising memory accesses is generally a profitable strategy. Combining elementary operations into less general ones opens up numerous avenues for optimisation. However, such a strategy can be pushed too far; eventually one ends up with an undesirable situation involving proliferation of less general, more complicated subroutines, each requiring maintenance, optimisation and documentation. One seeks therefore a happy mean. The need for a set of standard routines for this kind of application is, of course, not new, and it was to meet this need that the Basic Linear Algebra Subroutines (BLAS) library was developed. The vector-vector operations encountered here are the domain of Level 1 BLAS. BLAS libraries are widely available on almost any platform capable of seriously tackling a problem of this size, and the subroutines themselves are typically highly optimised for the target architecture.

The strategy followed in UKQCD's MPP codes was to develop a set of routines, similar to Level 1 BLAS, specifically designed to deal with vectors defined by a single parity of a fermion field. The set of operations supported is somewhat wider than Level 1 BLAS. All these routines are written in Fortran, but, according to compile-time flags specifying the precision, architecture and ordering of indices, preprocessing will replace the Fortran code by one or more calls to BLAS routines when it is advantageous to do so.

Table 4.2 lists some of our routines, the operations they perform and the number of times each is used in each iteration of MR, BiCG(γ_5), QMR(γ_5) and BiCGSTAB, in the current implementation. Figures for CGNR and BiCGSTAB2 are not given. We have no current implementation of (point) CGNR and to quote figures from the block version would be unfair. The complexity of BiCGSTAB2 means that there are so many ways of achieving the same end that a breakdown would not be particularly meaningful; suffice to say that the cost of vector-vector operations averaged over even and odd iterations of BiCGSTAB2 is at least 50% more than that of BiCGSTAB.

The operations implemented by the subroutines `faxpy` through `fysx` (in the order of table 4.2) are all local, in the sense that no inter-processor communications are required. This is the case because every processor has the same components of every vector in its local memory. Of these, only `faxpy`, `fcaxpy`, `fscal` and `fcscal` can be implemented using a single call to a standard level 1 BLAS subroutine.[19]

[19]On vector supercomputers (where the `site` index varies fastest and components of each vector are not contiguous in memory) several BLAS calls may be required.

The other routines exist because to achieve the same end through two BLAS calls would entail additional memory accesses; for example the effect of `faxpz` could be achieved by `fcopy` followed by `faxpy`, both of which are built on BLAS routines, but a well designed Fortran loop should perform better.

The operations implemented by the subroutines and functions `fmod2` through `fc_pseudo_dot` involve scalar products and are non-local. Each processor computes a local scalar product, but the results must then be summed over processors. Both levels of summation are performed in 64-bit precision on all platforms. BLAS routines exist for the local part of the scalar products in `fmod2` and `fcdot`.[20] The routines `fc_pseudo_norm` and `fc_pseudo_dot_re` arise in the γ_5-symmetric algorithms BiCG(γ_5) and QMR(γ_5).[21] It is obviously desirable to fold the multiplication by γ_5 into the calculation of the scalar product. However, despite prodigious unrolling of loops, these routines proved particularly difficult to optimise, and their performance is poor compared to the efficient BLAS CDOTC used in `fc_dot`.

Further attempts at optimisation in the area of vector-vector arithmetic can be undertaken. For example, it is often the case that an operation such as $y = \alpha x + y$ is followed by computing the 2-norm of the result. Telescoping the two operations into a single subroutine would eliminate one complete access to y, but unless the new routine is written in assembly language[22] the speed-up, if any, is likely to be marginal.

[20]Actually, our implementations of `fmod2` and `fcdot` avoid the use of BLAS when the fermion fields are stored in 32-bit precision on the Cray-T3D and Cray-T3E. This is because the BLAS functions HDOT and GDOTC (the 32-bit equivalents of SDOT and CDOTC on Cray systems) return 32-bit (KIND=4) results which have still to be summed over processors.

[21]The subroutine `fc_pseudo_dot` is included only for completeness. It turns out that in all cases where an operation of the form $y^\dagger \gamma_5 x$ is required, the answer is known to be real due to the γ_5-symmetry of M.

[22]Efforts along these lines were made by Cray to optimise the MR solver, the benchmark code for the tender leading to the purchase of the Cray-T3E at Edinburgh. The performance improvement achieved was dwarfed by improvements to the matrix-vector multiplication.

Routine	Operation	BLAS	flops per site	MR	BiCG (γ_5)	QMR (γ_5)	BiCG STAB
faxpy	$y = \alpha x + y$, $\alpha \in R$	Y	48		2	3	
faxpz	$y = \alpha x + z$, $\alpha \in R$	N	48				
faypx	$y = \alpha y + x$, $\alpha \in R$	N	48		1	2	
fcaxpy	$y = \alpha x + y$, $\alpha \in C$	Y	96	2			4
fcaxpz	$y = \alpha x + z$, $\alpha \in C$	N	96				
fcaypx	$y = \alpha y + x$, $\alpha \in C$	N	96				1
fcax	$y = \alpha x$, $\alpha \in C$	N	72				
fscal	$x = \alpha x$, $\alpha \in R$	Y	24			2	
fcscal	$x = \alpha x$, $\alpha \in C$	Y	72				
fysx	$y = y - x$	N	24				
fcopy	$y = x$	Y	0				
fmod2	$x^\dagger x$	Y	48,G	2	1	1	2
fcdot	$y^\dagger x$	Y	96,GG	1			3
fc_pseudo_norm	$x^\dagger \gamma_5 x$	N	24,G		1	1	
fc_pseudo_dot_re	Re $y^\dagger \gamma_5 x$	N	48,G		1	1	
fc_pseudo_dot	$y^\dagger \gamma_5 x$	N	96,GG				
rb_matrix	$y = Mx$	N		1	1	1	2

Table 4.2: Operations performed in each iteration of various point algorithms. A letter G in the "flops per site" column indicates a global sum.

4.4.2 Testing

Testing numerical codes such as these is not simply a case of checking that the code delivers the correct answer. The "correct answer" is an idealisation that we never expect to achieve; we must be content with an approximate solution ψ that is sufficiently close to the true solution $M^{-1}\eta$. The meaning of "sufficiently close" must be determined from the physics programme making use of the propagators; at the least one requires that errors induced by imperfect convergence be small compared to statistical fluctuations over the ensemble of gauge configurations. At this point it is wise to recall that the closeness of our solution to the true one is controlled only by a convergence criterion based on the norm of the residual vector, and that this control necessarily becomes weaker as the quark mass decreases.

Bear in mind also that distinct but equally satisfactory solutions can be delivered in a number of ways. Even a minor change in the order of summation (arising from running the code on a different number of processors, or changing a compile time option to enable calls to BLAS, or changing the order of indices in the fermion fields) will give rise to a measurably different solution. Choice of algorithm will, of course, have a major effect.

The solver code has an in-built independent check on the equation residual,[23] performed at the end of every solve, which goes part of the way to confirming the goodness of the solution. A more genuinely independent check is to load from disk two solutions and to compute the distance between them; I have written a utility to perform this check. Such a tool, used in conjunction with reference solutions obtained by a trusted code using a tight convergence criterion, becomes an invaluable mechanism for change control. It can also be used to check that the convergence criteria used at particular quark masses are indeed "sufficiently close" (in the above sense) to the true solution.

[23]Although the subroutine performing the check uses the full matrix instead of the red-black preconditioned matrix, this check is not truly independent as the two matrix multiplication routines do themselves share some subroutines.

During the early stages of program development and debugging, the algorithm is unlikely to converge at all, and other tests must be devised. Writing individual test harnesses and constructing test data for every new subprogram is, although advocated by some, often impractical; usually the effort to write the harness is much greater than the effort to write the subprogram, and the harness itself must be debugged. I generally try to devise a quantity or quantities which can be computed in two ways, one of which uses trusted routines as far as possible and the other making use of the new routine(s); it is easy to see how this might be done for some of the operations in table 4.2.

Once confidence in the elementary building blocks has been established, one turns to the task of debugging the main body of the algorithm. Often the algorithm itself suggests useful consistency tests. For example, BiCG(γ_5) and QMR(γ_5) both depend on bi-orthogonality relations between sequences of vectors, and it is easy to insert code to monitor the growth of bi-orthogonality violations,[24] and then to remove (or disable) the inserted code before commencing final tests on the production version.

One test in particular proved such a useful diagnostic for all algorithms (except QMR) that I left compile-time flags in the code so that it could be re-enabled easily if required. This test is to compute explicitly the equation residual at the end of each iteration. If the algorithm is working correctly, the discrepancy between the equation residual $\eta - M\psi^{(k)}$ and that given by recurrences $r^{(k)}$ should be small; a good quantity to monitor is $||r^{(k)} - \left(\eta - \psi^{(k)}\right)||$. This test, strong as it is, does not catch all bugs. For example, BiCG(γ_5) will pass this test even if the coefficient δ is computed incorrectly.

A bug in the main body of an algorithm will usually prevent the algorithm from converging. Unfortunately this is not always the case, so convergence does not in itself guarantee the correctness of the code. When implementing BiCGSTAB2, I

[24]BiCGSTAB depends theoretically on the same bi-orthogonality relations as BiCG, but most of the vectors concerned do not appear explicitly in BiCGSTAB. Nonetheless there are some properties which can usefully be checked.

overlooked a typographical error in the computation of a coefficient; the bug broke the global biorthogonality properties of the underlying BiCG but did not affect the local minimisation of BiCGSTAB2; the resulting code converged often but not consistently.[25] It is therefore helpful to have some idea of how an algorithm should behave – how smoothly and how quickly it is expected to converge – in deciding whether it has been implemented correctly.

4.5 Comparison

Figure 4.1 overlays the convergence histories of MR, BiCG(γ_5), QMR(γ_5) BiCGSTAB and BiCGSTAB2 at two different quark masses.[26]

At the heavier mass ($\kappa = 0.1400$), MR is noticeably slower than the other four methods, and the gap widens as the quark mass is decreased ($\kappa = 0.14200$). This behaviour is typical.

The relatively smooth convergence history of QMR(γ_5) tends to follow the troughs of the erratic convergence history of BiCG(γ_5). This, too, is typical, and reflects the fact that the two methods are closely related.

BiCGSTAB and BiCGSTAB2 trace out very similar convergence histories, the residual norm of BiCGSTAB being slightly above that of BiCGSTAB2 in each iteration.

There is a suggestion that at lighter masses, the performance of BiCG(γ_5) and QMR(γ_5) improves relative to that of BiCGSTAB as the convergence criterion is tightened.

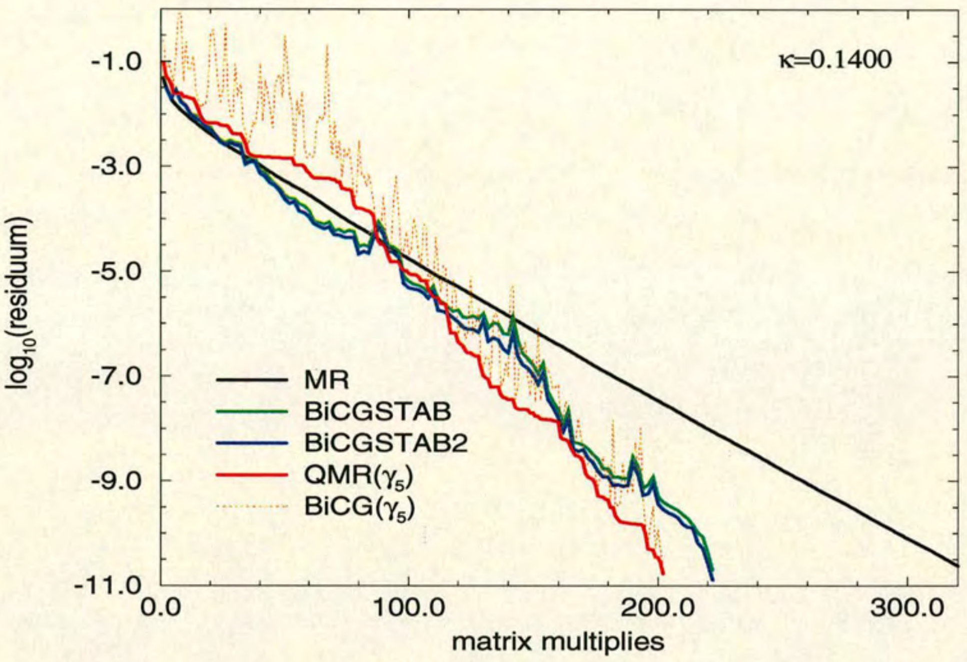
In deciding which algorithm to use for calculating quark propagators in a production run, there is no substitute for timing each candidate on the intended architecture using the intended parameters (κ, C_{SW}, convergence criterion) on one or more configurations drawn from the ensemble of interest. These tests

[25]It was some time before I was prepared to accept that the code was in error, and it took a further two weeks to locate the bug.

[26]Convergence histories for CGNR may be found in chapter 5.

have been carried out from time to time in preparation for production runs on the Cray-T3D,[27] and BiCGSTAB has usually proved to be the method of choice, typically outperforming QMR(γ_5) and BiCG(γ_5) by some 10%. The size of this gap, due in part to the fact that the vector-vector operations used by BiCGSTAB have more efficient implementations, is not absolutely compelling. More significant perhaps is the fact that BiCGSTAB more consistently achieves a target residuum of $10^{-7}||\eta||$ when 32-bit precision is used for the fermion fields than do BiCG(γ_5) and QMR(γ_5); as performance on the Cray-T3D is dominated by memory-bandwidth, using 32-bit precision where possible represents a major optimisation.

[27]These runs have involved quenched gauge configurations at $\beta = 6.0 - 6.2$, lattices of volume $16^3 \times 48$ and larger, both tadpole and non-perturbatively improved clover coefficients, and target residuums of $10^{-7}||\eta||$. The timing exercises concentrated on the lightest masses (typically near the strange mass) as these are the most time-consuming.



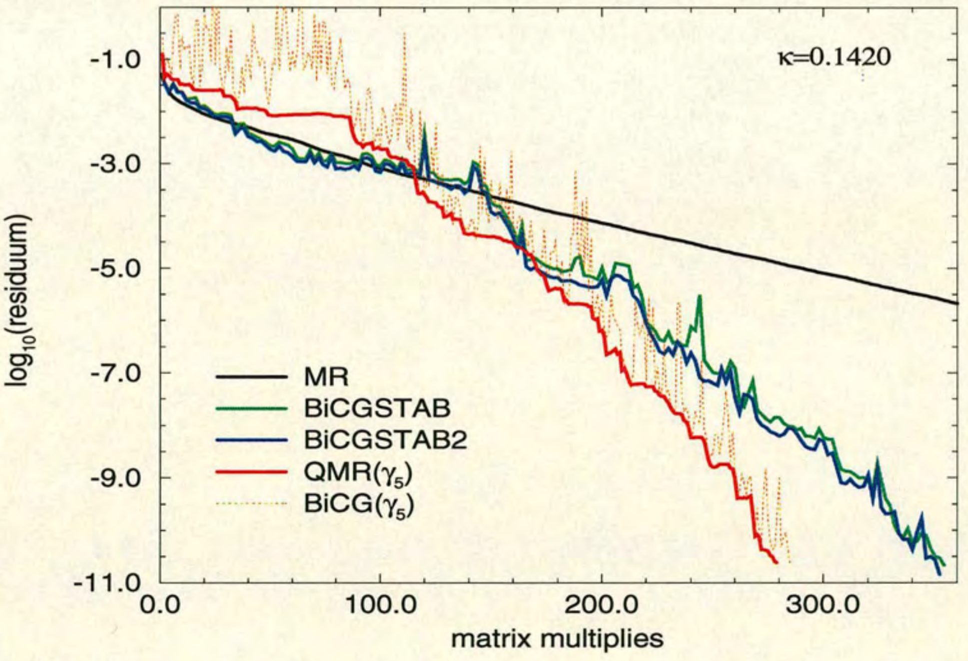


Figure 4.1: Convergence histories for :–
MR ($\omega = 1.1$), BiCGSTAB, BiCGSTAB2, QMR(γ_5), BiCG(γ_5).
$\kappa = 0.14,\ 0.142,\ \beta = 5.7,\ V = 8^3 \times 8.\ C_{SW} = 1.5678$ (tadpole-improved).

4.6 Exceptional Configurations

In this section, we consider a method for computing quark propagators when the fermion matrix develops a single, very small, isolated eigenvalue. The original hope was that this method might have been useful in handling the "exceptional configurations" observed in quenched UKQCD data [44]. These configurations were such that $Q = \gamma_5 M$ had a very small eigenvalue at a valence κ near the strange mass, well below the critical value κ_c of the ensemble. Our usual methods of computing propagators (BiCGSTAB and MR) failed miserably at this κ on these configurations, and 3,000 CGNR iterations reduced the residuum to only about $10^{-3}||\eta||$.

The essential idea is to reduce the coefficient matrix by projecting out the eigenvector corresponding to the smallest eigenvalue.[28] The next few paragraphs show why this might be expected to work from a theoretical point of view.

Let A be hermitian with eigenvalues λ_i ordered by their absolute value, so that

$$|\lambda_1| \leq |\lambda_2| \leq \ldots \leq |\lambda_n| \tag{4.36}$$

and let $\{v_i\}$ be the corresponding orthonormal eigenvectors.

$$Av_i = \lambda_i v_i, \qquad v_i^\dagger v_j = \delta_{ij} \tag{4.37}$$

Further suppose that the lowest eigenvalue is non-degenerate and very small. Under these conditions, the condition number of A is very large and standard methods for solving

$$Ax = b \tag{4.38}$$

will often fail.

[28]A similar idea was proposed in [45, 46] in the context of the Polynomial Hybrid Monte Carlo algorithm.

Now define

$$\begin{aligned} P &= v_1 v_1^\dagger \\ P_\perp &= 1 - P = \sum_{i=2}^{n} v_i v_i^\dagger \end{aligned} \tag{4.39}$$

These are projections

$$\begin{aligned} P^2 &= P \\ P_\perp^2 &= P_\perp \end{aligned} \tag{4.40}$$

and orthogonal

$$P P_\perp = 0 = P_\perp P. \tag{4.41}$$

Defining also

$$\begin{aligned} A_P &= \lambda_1 v_1 v_1^\dagger = \lambda_1 P \\ A_\perp &= \sum_{i=2}^{n} \lambda_i v_i v_i^\dagger = A - A_P \end{aligned} \tag{4.42}$$

we can re-write equation (4.38) in the form

$$(A_P + A_\perp)(P + P_\perp)x = (P + P_\perp)b. \tag{4.43}$$

Using the fact that

$$A_P P_\perp = 0 = A_\perp P \tag{4.44}$$

and defining $z_P = Pz$, $z_\perp = P_\perp z$, we can solve

$$A_P x_P = b_P \tag{4.45}$$

on the subspace spanned by v_1, then solve

$$A_\perp x_\perp = b_\perp \tag{4.46}$$

on the subspace orthogonal to v_1, and finally reconstruct the solution on the whole space from

$$x = x_P + x_\perp. \tag{4.47}$$

The solution x_P to (4.45) is just

$$x_P = \lambda_1^{-1} v_1^\dagger b \; v_1 \tag{4.48}$$

and the solution $x_\perp$ to (4.46) can be found by any one of a number of standard iterative methods. The hope is that equation (4.46) will be much easier to solve than the original equation (4.38). It will certainly have a lower condition number.

$$\mathrm{cond}(A_\perp) = \frac{|\lambda_n|}{|\lambda_2|} = \frac{|\lambda_1|}{|\lambda_2|}\mathrm{cond}(A) \tag{4.49}$$

This method requires knowing both λ_1 and v_1. We will not know either exactly, and will have to work with some $\lambda = \lambda_1 + \delta$ and $v = v_1 + \epsilon$. This will induce errors in both x_P and $x_\perp$. We still have some freedom in how we choose to represent $A_\perp$. In exact arithmetic, all of the following expressions are equal:

$$A_\perp = A - \lambda_1 v_1 v_1^\dagger = A(1 - v_1 v_1^\dagger) = (1 - v_1 v_1^\dagger)A = (1 - v_1 v_1^\dagger)A(1 - v_1 v_1^\dagger) \tag{4.50}$$

I prefer to work with the last form $A_\perp = (1 - v_1 v_1^\dagger)A(1 - v_1 v_1^\dagger)$ because

1. it is manifestly hermitian

2. A is 'protected' on *both* sides

3. it gives us better control over the errors.

The last assertion may be justified by considering the error induced in the equation residual

$$r = b - Ax \tag{4.51}$$

by the errors in λ_1 and v_1.

The strategy is as follows. To solve $M\psi = \eta$, find the smallest eigenvalue λ_1 and corresponding eigenvector v_1 of the hermitian matrix $Q = \gamma_5 M$. Then use the above method in conjunction with a standard iterative scheme for hermitian matrices to solve $Ax = b$ where A=Q, $b = \gamma_5\eta$. Then $\psi = x$ is the solution we want. The algorithm set out below applies CGNR to the normal equation

$$Q_\perp^2 x_\perp = Q_\perp b_\perp. \tag{4.52}$$

Note that equation (4.53) for σ holds if everything is exact. In practice we choose σ to minimise $||\eta - M\psi||$, and we do not use λ_1 at all.

This method was tested using a configuration on an 8^4 lattice at a κ for which $|\lambda_2| > 10|\lambda_1|$. The eigenvector v_1 was pre-computed using the method described in [47, 48, 49] and read in from disk.[29] Under these conditions, MR, BiCG(γ_5) and BiCGSTAB did not converge at all, stagnating or beginning to diverge at $||r|| \sim 10^{-3}||\eta||$. CGNR with projection converged considerably faster than CGNR without. However, QMR(γ_5) converged several times faster again. QMR(γ_5) was later was applied successfully to compute propagators for the "exceptional configurations" encountered in [44].[30]

In conclusion, the method is viable, but should only be adopted as a last resort when other methods fail. It will perform best when the ratio $|\lambda_2|/|\lambda_1|$ is large. Practical experience indicates that the method requires an extremely good approximation of the eigenvector v_1, in order to keep the reconstructed residual $\eta - M\,(\psi_P + \psi_\perp)$ under control. In fact, tighter convergence criteria on the eigenvector than those used in [49] are required for this method to be successful. It should also be pointed out that the cost of computing the eigenvector is comparable to that of several solves; so this method is most certainly not to be regarded as a means of accelerating propagator calculations.

[29]I am grateful to the authors of [49] for making their code available to me.

[30]Ultimately these propagators were not included in the analysis of the hadron spectrum.

CGNR projecting out v_1

```
$b_\perp = (1-P)\gamma_5\eta = \gamma_5\eta - \sigma v_1$
$x^{(0)} = (1-P)\psi^{(0)}$
$r^{(0)} = b_\perp - Q_\perp x^{(0)}$
$p^{(0)} = Q_\perp r^{(0)}$
$\rho_0 = \left(p^{(0)}\right)^\dagger p^{(0)}$
for $k = 0, 1, 2, \ldots$ until convergence do {
    $t = Q_\perp p^{(k)}$
    $\pi = t^\dagger t$
    $\alpha_k = \pi^{-1}\rho_k$
    $x^{(k+1)} = p^{(k)}\alpha_k + x^{(k)}$
    $r^{(k+1)} = -t\alpha_k + r^{(k)}$
    $t = Q_\perp r^{(k+1)}$
    $\rho_{k+1} = t^\dagger t$
    $\beta_k = \rho_k^{-1}\rho_{k+1}$
    $p^{(k+1)} = t + p^{(k)}\beta_k$
}
$\sigma = v_1^\dagger\gamma_5\eta/\lambda_1$                                  (4.53)
$\psi = x + \sigma v_1$
```

4.7 Concluding Remarks

Of the methods considered in this chapter, BiCGSTAB is the usually the method of choice for quark propagator calculations in hadron spectroscopy. With respect to BiCGSTAB, BiCGSTAB2 does not reduce the number of matrix multiplications sufficiently to offset the additional vector-vector operations.

MR is simple and robust, but its performance degrades more rapidly than the other methods considered in this chapter as $\kappa \to \kappa_c$. It is too slow in the important regime of light quark mass, but it remains a viable option for heavy quarks.

BiCG(γ_5) and QMR(γ_5) offer performance which is competitive with BiCGSTAB, but I do not recommend the use of these algorithms when the fermion fields are stored in 32-bit precision. Nonetheless, QMR(γ_5) has certain advantages which earn it a place in every well-equipped armoury of lattice QCD tools. It copes surprisingly well when the fermion matrix has a very small or even negative eigenvalue; because it works with the Lanczos vectors directly it can be modified to provide estimates of the smallest and largest eigenvalues; and look-ahead versions of QMR exist which avoid the problem of Lanczos breakdown.

CGNR and CGNE are certainly not made obsolete by BiCGSTAB, and these remain the most reliable methods for exploring the region $\kappa \gtrsim \kappa_c(\beta)$.

It is unlikely that further research in Krylov-subspace methods will lead to an algorithm which significantly outperforms BiCGSTAB and its γ_5-symmetric competitors. There is however considerable scope for improvement in the area of preconditioning.

Chapter 5

Linear systems II – block algorithms

Previous studies of block algorithms in the context of lattice QCD [50, 51] have been restricted to small lattices. A re-examination is now in order, given the recent improvements in Krylov subspace methods and the advent of large distributed memories.

5.1 Motivation

Krylov subspace methods generate successive approximations $\psi^{(k)}$ to $\psi = M^{-1}\eta$ such that

$$\psi^{(k)} - \psi^{(0)} \in K_k\left(M, y\right), \tag{5.1}$$

where

$$K_k\left(M, y\right) = \operatorname{span}\{y, My, \ldots, M^{k-1}y\} \tag{5.2}$$

is the k-th Krylov subspace generated by the matrix M and the vector

$$y = r^{(0)} = \eta - M\psi^{(0)}. \tag{5.3}$$

These methods either

1. minimise (or quasi-minimise) the residuum $||r^{(k)}|| = ||\eta - M\psi^{(k)}||$ over $K_k\left(M, y\right)$, and/or

2. enforce bi-orthogonality with respect to some other Krylov subspace $r^{(k)} \perp K_k\left(M^\dagger, \bar{y}\right)$.

When one has several systems $M\psi_1 = \eta_1, \ldots, M\psi_s = \eta_s$ to solve, using such a method s times leads to the construction of the Krylov subspaces

$$K_{n_1}(M, y_1), \ldots, K_{n_s}(M, y_s).$$

These subspaces will in general overlap with each other, and in the worst case

$$n_1 = n_2 = \ldots = n_s = n = \text{order}\,(M)$$

the overlap will be complete.

By solving the s systems simultaneously, block algorithms eliminate the redundant matrix-vector operations in the above approach. One assembles the s right-hand sides into an $n \times s$ matrix $H = (\eta_1, \ldots, \eta_s)$ and solves

$$M\Psi = H \tag{5.4}$$

for $\Psi = (\psi_1, \ldots, \psi_s)$. Information from all s Krylov subspaces is now available to update each column of the solution.

The preceding discussion ignores the effect of preconditioning. A perfect preconditioner, ie. one which coincides exactly with the inverse of the coefficient matrix, solves the system with a single multiplication; in this case there is no gain to be had from the block algorithm. In practice we hope to have good preconditioners, so that we usually solve the point ($s = 1$) problem to the desired accuracy in considerably less than n multiplications.

These considerations lead one to expect that blocking will be most effective on badly conditioned systems and/or small volumes, ie. when the number of iterations required for the point algorithm to converge is comparable to the order of the matrix.

Blocking introduces certain overheads. The first of these is memory; the storage requirements for vectors in the point algorithm are multiplied by the blocking factor s when going to the block algorithm.

Secondly, vector-vector operations such as

$$y = \alpha x + y \text{ and } \beta = y^\dagger x$$

in the point algorithm generalise to operations of the form

$$Y = X\alpha + Y \text{ and } \beta = Y^\dagger X$$

where X and Y are $n \times s$ matrices and α and β have become $s \times s$ matrices. Thus the number of vector-vector operations required per iteration scales with the *square* of the blocking factor. We are in the business of trading matrix-vector operations for vector-vector operations.

Thirdly, the trivial operations of multiplication and division by complex numbers generalise to the less trivial operations of multiplication by and inversion of complex matrixes. Moreover, on a parallel architecture, these operations have to be replicated on every processing node.

5.2 Block Algorithms

5.2.1 Block Minimal Residual

Minimal Residual is not strictly a Krylov subspace method. Each iteration deals with a different one-dimensional Krylov subspace and the algorithm lacks any termination property. There is no *a priori* reason to expect that a block version should outperform the point version.

Nonetheless, a block generalisation of (unrelaxed) MR is easily derived, and the exercise of doing so reveals some generic characteristics of block algorithms.

In the point algorithm, the correction to solution at each iteration is in the direction of the residual vector. In the block algorithm, we want to use information from all s linear systems when updating each column of the solution. To achieve this, the corrections to each column of the solution must be linear combinations of the residual vectors of all s systems. So the core of our iterative method will take the form

$$\Psi^{(k+1)} = \Psi^{(k)} + R^{(k)}\alpha^{(k)} \tag{5.5}$$

for some $s \times s$ matrix $\alpha^{(k)}$ which is yet to be determined.

We also want our recurrence relations to ensure that

$$R^{(k)} = H - M\Psi^{(k)} \tag{5.6}$$

is true for all k. We can make this so for $k = 0$. Assuming it holds for some k, and using equation (5.5) we observe that

$$\begin{aligned} R^{(k+1)} &= H - M\left(\Psi^{(k)} + R^{(k)}\alpha^{(k)}\right) \\ &= \left(H - M\Psi^{(k)}\right) - MR^{(k)}\alpha^{(k)}. \end{aligned} \tag{5.7}$$

This tells us how to update the residuals.

It remains to determine $\alpha^{(k)}$. In the point algorithm, the coefficient $\alpha^{(k)}$ is chosen to minimise $r^{(k+1)}$. In the block algorithm, we minimise instead $\mathrm{Tr}\ \left(R^{(k+1)}\right)^\dagger R^{(k+1)}$, which is equivalent to minimising the root mean square of the norms of the residual vectors of the s systems. This is accomplished by setting

$$\frac{\partial}{\partial \alpha_{ij}^{(k)*}} \mathrm{Tr}\ \left(R^{(k+1)}\right)^\dagger R^{(k+1)} = 0 \tag{5.8}$$

and solving for $\alpha_{ij}^{(k)}$.

Thus we arrive at the following algorithm.

B-MR

```
$R^{(0)} = H - M\Psi^{(0)}$
for $k = 0, 1, 2, \dots$ until convergence do {
    $T = MR^{(k)}$
    $\alpha^{(k)} = \left(T^\dagger T\right)^{-1} T^\dagger R^{(k)}$
    $\Psi^{(k+1)} = R^{(k)}\alpha^{(k)} + \Psi^{(k)}$
    $R^{(k+1)} = -T\alpha^{(k)} + R^{(k)}$
}
```

I have implemented this method and found that increasing s yields a negligible improvement in convergence. For $s = 1$, it reduces to the unrelaxed ($\omega = 1.0$) Minimal Residual considered in chapter 4.

5.2.2 B-CGNR

The generalisation of Conjugate Gradient to block form is due to O'Leary [52], who obtained the algorithm as a special case of her Block Bi-Conjugate Gradient. McCarthy [50] found that Block Conjugate Gradient could be applied advantageously to quark propagator calculations with staggered fermions.

We, however, are using Wilson fermions, and we apply Block Conjugate Gradient to the normal equation

$$M^\dagger M\Psi = M^\dagger H. \tag{5.9}$$

We call the resulting algorithm Block Conjugate Gradient on the Normal equation, least Residual formulation. The same algorithm was applied to quark propagator calculations with Wilson fermions by the authors of [51], who called it MDMCG.

Increasing s yields a clear improvement in convergence (see figure 5.1), but not enough to defray the cost of squaring the condition number.

B-CGNR

```
$R^{(0)} = H - M\Psi^{(0)}$                                  (5.10)
$P^{(0)} = M^\dagger R^{(0)}$                                (5.11)
$\rho^{(0)} = \left(P^{(0)}\right)^\dagger P^{(0)}$          (5.12)
for $k = 0, 1, 2, \ldots$ until convergence do {
    $T = MP^{(k)}$                                           (5.13)
    $\pi = T^\dagger T$                                      (5.14)
    $\alpha = \pi^{-1}\rho^{(k)}$                            (5.15)
    $\Psi^{(k+1)} = P^{(k)}\alpha + \Psi^{(k)}$              (5.16)
    $R^{(k+1)} = -T\alpha + R^{(k)}$                         (5.17)
    $T = M^\dagger R$                                        (5.18)
    $\rho^{(k+1)} = T^\dagger T$                             (5.19)
    $\beta = \left(\rho^{(k)}\right)^{-1}\rho^{(k+1)}$       (5.20)
    $P^{(k+1)} = T + P^{(k)}\beta$                           (5.21)
}
```

5.2.3 Block Lanczos

Henty et. al [51] studied a method based on the hermitian block Lanczos process and applied it to

$$\gamma_5 M \Psi = \gamma_5 H. \tag{5.22}$$

In this formulation, we need each column of the RHS to have unit norm, and we work with the hermitian matrix $Q = \gamma_5 M$, solving

$$Q\Phi = \gamma_5 H \nu^{-1}, \tag{5.23}$$

where

$$\nu = \mathrm{diag}(||\eta_1||, \ldots, ||\eta_s||) \tag{5.24}$$

with η_j denoting the jth column of H. The solution to the original system is $\Psi = \Phi\nu$.

This algorithm does not yield the residual vector R directly. However, it does yield an $s \times s$ matrix $r^{(k)}$ such that $\left(r^{(k)}\right)^\dagger r^{(k)} = \left(R^{(k)}\right)^\dagger R^{(k)}$ which can be tested for convergence.

Note also that, in this formulation at least, B-Lanczos is rather more restricted than the other algorithms we have considered. For example, it does not allow for any intial guess to be supplied, so any restarts must be done using a different algorithm. The block-vector $X^{(0)}$ used to start the Lanczos sequence coincides with the initial residual of equation (5.23) for the guess $\Phi^{(0)} = 0$. Furthermore, $X^{(0)}$ must satisfy $\left(X^{(0)}\right)^\dagger X^{(0)} = I$. This imposes some restrictions on how we can block the right hand sides if using extended (eg. smeared) sources.

The algorithm uses the Cholesky decomposition to orthonormalise the columns of the Lanczos block-vectors. In one iteration, we find an $s \times s$ upper triangular matrix $\beta^{(k+1)}$ satisfying equation (5.26), and in the next iteration, equation (5.25) ensures that $\left(X^{(k+1)}\right)^\dagger X^{(k+1)} = I$. Unfortunately, the Cholesky decomposition in equation (5.26) occasionally fails, attempting to take the square root of a negative number. Now the Cholesky decomposition is stable for hermitian positive definite matrices (in fact, it is often used to test an hermitian matrix for positive definiteness), and its failure may be interpreted as indicating that $W^\dagger W$ is indefinite, ie. that W lacks full column rank. This kind of problem (linear dependence of the Lanczos vectors or search directions) must be addressed in any robust implementation of a block algorithm; we shall have more to say on this topic in the next section.

Henty found, as we do, that B-Lanczos clearly outperforms B-CGNR. Unfortunately, γ_5 is a poor preconditioner for equation (5.4),[1] and the wisdom of working with the hermitian system is questionable. Consequently, we now proceed directly to non-hermitian methods, making no attempt to lift the aforementioned restrictions in B-Lanczos.

[1]Although $\gamma_5 M$ has the same condition number as M, its spectral radius is approximately twice that of M. Alternatively, given the structure of M, even the identity matrix will be a better approximation to M^{-1} than will γ_5.

B-Lanczos

```
$\nu = \mathrm{diag}(||\eta_1||, \ldots, ||\eta_s||);\ X^{(0)} = \gamma_5 H \nu^{-1}$
$a^{(0)} = t^{(0)} = I;\ b^{(0)} = y^{(0)} = 0$
$\alpha^{(0)} = \left(X^{(0)}\right)^\dagger \gamma_5 M X^{(0)}$
$W = \gamma_5 M X^{(0)} - X^{(0)} \alpha^{(0)}$
$\left(\beta^{(0)}\right)^\dagger \beta^{(0)} = W^\dagger W;\ \rho^{(0)} = \left(\beta^{(0)}\right)^{-\dagger}$
$U^{(0)} = 0;\ V^{(0)} = -X^{(0)} \left(\beta^{(0)}\right)^{-1}$
for $k = 0, 1, 2, \ldots$ until convergence do {
    $X^{(k+1)} = W \left(\beta^{(k)}\right)^{-1}$    (5.25)
    $\alpha^{(k+1)} = \left(X^{(k+1)}\right)^\dagger \gamma_5 M X^{(k+1)}$
    $W = \gamma_5 M X^{(k+1)} - X^{(k)} \left(\beta^{(k)}\right)^\dagger - X^{(k+1)} \alpha^{(k+1)}$
    $\left(\beta^{(k+1)}\right)^\dagger \beta^{(k+1)} = W^\dagger W$    (5.26)
    $a^{(k+1)} = -\alpha^{(k+1)} \rho^{(k)} + b^{(k)}$
    $\sigma^{(k)} = \left(a^{(k+1)}\right)^{-1} t^{(k)}$
    $U^{(k+1)} = U^{(k)} + X^{(k+1)} \rho^{(k)}$
    $V^{(k+1)} = V^{(k)} - U^{(k+1)} \sigma^{(k)}$
    $b^{(k+1)} = -\beta^{(k+1)} \rho^{(k)}$
    $y^{(k+1)} = y^{(k)} + \sigma^{(k)}$
    $t^{(k+1)} = -b^{(k+1)} \sigma^{(k)}$
    $d^{(k+1)} = \left(y^{(k+1)} + \alpha^{(0)} \left(\beta^{(0)}\right)^{-1}\right)^{-1}$
    $r^{(k+1)} = d^{(k+1)} t^{(k+1)}$
    if converged $\{\Psi^{(k+1)} = -V^{(k+1)} d^{(k+1)} \nu;$ break$\}$
    $\rho^{(k+1)} = \left(\beta^{(k+1)}\right)^{-\dagger} a^{(k+1)}$
}
```

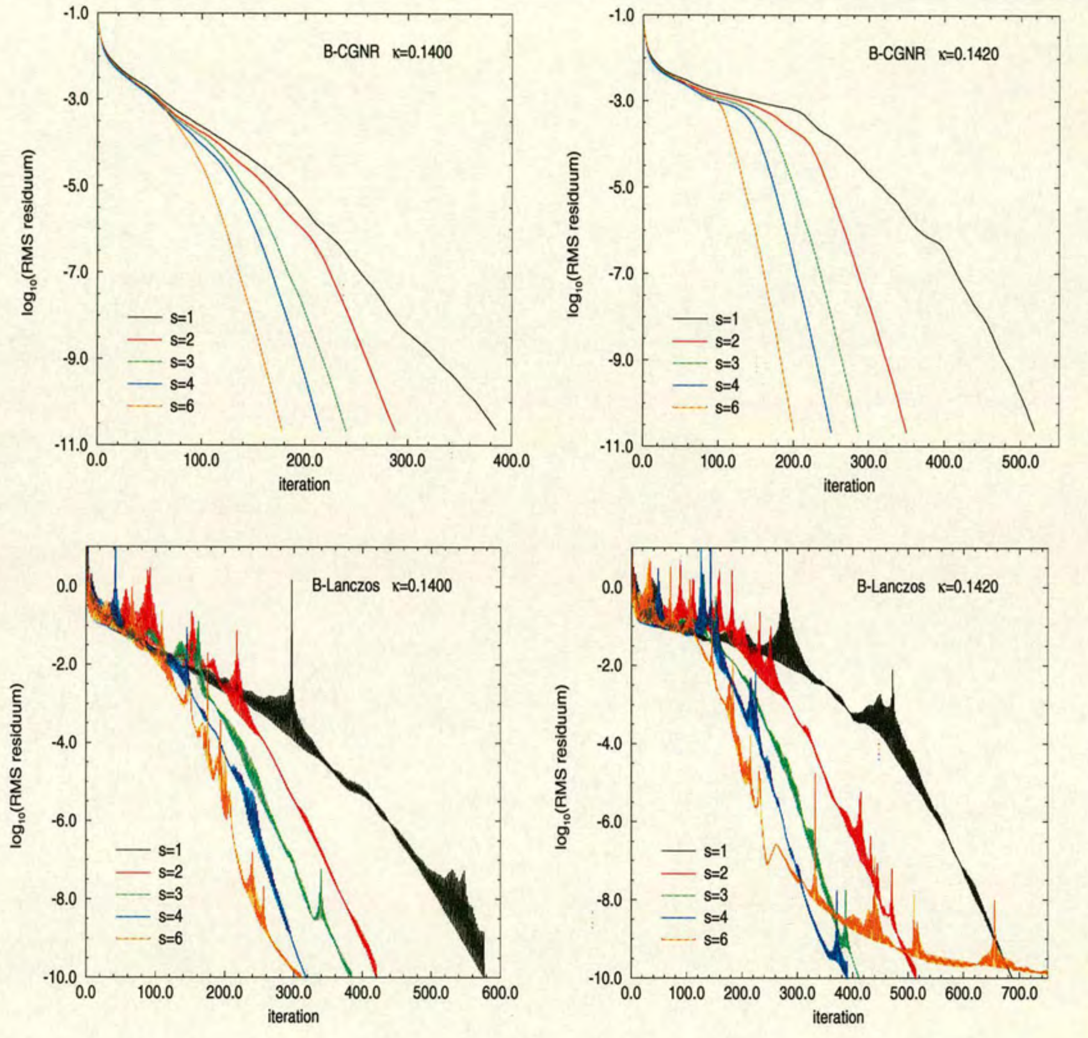


Figure 5.1: Convergence histories for B-CGNR (top) and B-Lanczos at various blocking factors s.

$\kappa = 0.14,\ 0.142$, $V = 8^4$, $\beta = 5.7$, $C_{SW} = 1.5678$ (tadpole-improved).

Note that B-CGNR has two matrix-vector operations per iteration whereas B-Lanczos has one.

5.2.4 Block BiCG(γ_5)

Our starting point is the Block Bi-Conjugate Gradient algorithm (B-BiCG) [52]. For convenience, we give the full algorithm here, in a slightly different notation to that of O'Leary.

```
$R^{(0)} = H - M\Psi^{(0)}$
Choose $\bar{R}^{(0)}$ with full column rank
$P^{(0)} = VR^{(0)}\sigma^{(0)}$
$\bar{P}^{(0)} = V^T\bar{R}^{(0)}\bar{\sigma}^{(0)}$
for $k = 0, 1, 2, \ldots$ until convergence do {
    $\alpha^{(k)} = \left(\left(\bar{P}^{(k)}\right)^T MP^{(k)}\right)^{-1} \left(\bar{\sigma}^{(k)}\right)^T \left(\bar{R}^{(k)}\right)^T VR^{(k)}$
    $\bar{\alpha}^{(k)} = \left(\left(P^{(k)}\right)^T M^T\bar{P}^{(k)}\right)^{-1} \left(\sigma^{(k)}\right)^T \left(R^{(k)}\right)^T V^T\bar{R}^{(k)}$
    $\Psi^{(k+1)} = P^{(k)}\alpha^{(k)} + \Psi^{(k)}$
    $R^{(k+1)} = -MP^{(k)}\alpha^{(k)} + R^{(k)}$
    $\bar{R}^{(k+1)} = -M^T\bar{P}^{(k)}\bar{\alpha}^{(k)} + \bar{R}^{(k)}$
    $\beta^{(k)} = \left(\sigma^{(k)}\right)^{-1} \left(\left(\bar{R}^{(k)}\right)^T VR^{(k)}\right)^{-1} \left(\bar{R}^{(k+1)}\right)^T VR^{(k+1)}$
    $\bar{\beta}^{(k)} = \left(\bar{\sigma}^{(k)}\right)^{-1} \left(\left(R^{(k)}\right)^T V^T\bar{R}^{(k)}\right)^{-1} \left(R^{(k+1)}\right)^T V^T\bar{R}^{(k+1)}$
    $P^{(k+1)} = \left(VR^{(k+1)} + P^{(k)}\beta^{(k)}\right)\sigma^{(k+1)}$
    $\bar{P}^{(k+1)} = \left(V^T\bar{R}^{(k+1)} + \bar{P}^{(k)}\bar{\beta}^{(k)}\right)\bar{\sigma}_{k+1}$
}
```

The $n \times n$ matrix V is an arbitrary pre-conditioner, and the choice of V can affect the convergence of the algorithm. The iterates $\Psi^{(k)}$ and residuals $R^{(k)} = H - M\Psi^{(k)}$ are invariant with respect to the choice of non-singular matrices $\sigma^{(k)}$ and $\bar{\sigma}^{(k)}$.

B-BiCG can be reformulated to use the hermitian conjugate operation $(.)^\dagger$ instead of the transpose operation $(.)^T$. One takes complex conjugates of the equations for $\bar{P}^{(k)}$, $\bar{R}^{(k)}$, $\bar{\alpha}^{(k)}$, $\bar{\beta}^{(k)}$ and $\bar{\sigma}^{(k)}$, and writes $\hat{P}^{(k)} = \left(\bar{P}^{(k)}\right)^*$, $\hat{R}^{(k)} = \left(\bar{R}^{(k)}\right)^*$, $\hat{\alpha}^{(k)} = \left(\bar{\alpha}^{(k)}\right)^*$, $\hat{\beta}^{(k)} = \left(\bar{\beta}^{(k)}\right)^*$ and $\hat{\sigma}^{(k)} = \left(\bar{\sigma}^{(k)}\right)^*$. The resulting algorithm is identical to the above with the lexical replacements $(*)^T \to (*)^\dagger$ and $\bar{*} \to \hat{*}$.

Just as in the point algorithm [25], it is possible to exploit the γ_5-symmetry of M to eliminate the sequences $\{\hat{R}^{(k)}\}$ and $\{\hat{P}^{(k)}\}$, as well as the multiplication by $M^\dagger$, which reduces the computational cost by a factor of two, and reduces the memory requirement from 6 vectors to 4 ($R^{(*)}$, $\Psi^{(*)}$, $P^{(*)}$, $T^{(*)} \equiv MP^{(*)}$). Provided we restrict ourselves to γ_5-symmetric preconditioners $V = \gamma_5 V^\dagger \gamma_5$, the particular choices

$$\hat{R}^{(0)} = \gamma_5 R^{(0)} \tag{5.27}$$

and

$$\hat{\sigma}^{(k)} = \sigma^{(k)}, \forall k \tag{5.28}$$

together imply that

$$\begin{aligned} \hat{R}^{(k)} &= \gamma_5 R^{(k)} & (5.29)\\ \hat{P}^{(k)} &= \gamma_5 P^{(k)} & (5.30)\\ \hat{\alpha}^{(k)} &= \alpha^{(k)} & (5.31)\\ \hat{\beta}^{(k)} &= \beta^{(k)} & (5.32) \end{aligned}$$

hold for all k.[2] Moreover, the terms

$$\left(\hat{P}^{(k)}\right)^\dagger MP^{(k)} = \left(P^{(k)}\right)^\dagger \gamma_5 MP^{(k)}$$

and

$$\left(\hat{R}^{(k+1)}\right)^\dagger VR^{(k+1)} = \left(R^{(k+1)}\right)^\dagger \gamma_5 VR^{(k+1)}$$

become hermitian, which fact can be exploited to reduce the number of dot

[2]The proof, by induction, is straightforward but tedious. I do not give it here.

products per iteration.

The simplest choice for the pre-conditioner V is the identity matrix, but there are other γ_5-symmetric objects about, viz. A, A^{-1}, M and their hermitian conjugates, along with γ_5 itself.[3]

We record the following properties of B-BiCG(γ_5):

$$\begin{aligned}
\left(R^{(k)}\right)^\dagger \gamma_5 R^{(j)} &= 0, \quad j \neq k && (5.33)\\
\left(P^{(k)}\right)^\dagger \gamma_5 M P^{(j)} &= 0, \quad j \neq k && (5.34)\\
\left(R^{(k)}\right)^\dagger \gamma_5 M P^{(k)} &= \left(\delta^{(k)}\right)^\dagger \left(P^{(k)}\right)^\dagger \gamma_5 M P^{(k)} && (5.35)\\
\left(R^{(k)}\right)^\dagger \gamma_5 P^{(j)} &= 0, \quad j < k && (5.36)\\
\left(R^{(k)}\right)^\dagger \gamma_5 R^{(k)} &= \left(R^{(k)}\right)^\dagger \gamma_5 P^{(k)} \delta^{(k)}. && (5.37)
\end{aligned}$$

O'Leary suggested choosing $\sigma^{(k)}$ to orthonormalise the columns of $P^{(k)}$. Until this step was incorporated in my implementation, convergence (even in 64-bit precision) was the exception rather than the rule.

Eliminating the sequences $\{\hat{R}^{(k)}\}$ and $\{\hat{P}^{(k)}\}$ and writing $\delta^{(k)} = \left(\sigma^{(k)}\right)^{-1}$ leads to the following form of the algorithm.[4] The coefficients α, β, ρ, γ, δ are $s \times s$ complex matrices.

[3] $V = \gamma_5$ is a superficially attractive choice, because the norm of the residual may be extracted from the diagonal components of $\left(R^{(k)}\right)^\dagger \gamma_5 V R^{(k)}$ at no extra cost, and the fact that this matrix is positive helps the stability of the algorithm which requires its inverse. However, practical tests indicate that the γ_5-preconditioned version takes more than twice as many iterations to converge as the $V = 1$ version.

[4] To the best of my knowledge, this is the first γ_5-symmetric version of B-BiCG applied to lattice QCD.

B-BiCG(γ_5)

```
$R^{(0)} = H - M\Psi^{(0)}$
$\rho^{(0)} = \left(R^{(0)}\right)^\dagger \gamma_5 R^{(0)}$
$P^{(0)}\delta^{(0)} = R^{(0)}$                                    (5.38)
for $k = 0, 1, 2, \ldots$ until convergence do {
    $T = MP^{(k)}$
    $\alpha^{(k)} = \left(\left(P^{(k)}\right)^\dagger \gamma_5 T\right)^{-1} \left(\delta^{(k)}\right)^{-\dagger} \rho^{(k)}$
    $\Psi^{(k+1)} = P^{(k)}\alpha^{(k)} + \Psi^{(k)}$
    $R^{(k+1)} = -T\alpha^{(k)} + R^{(k)}$
    $\rho^{(k+1)} = \left(R^{(k+1)}\right)^\dagger \gamma_5 R^{(k+1)}$
    $\beta^{(k)} = \delta^{(k)} \left(\rho^{(k)}\right)^{-1} \rho^{(k+1)}$
    $T = R^{(k+1)} + P^{(k)}\beta^{(k)}$
    $P^{(k+1)}\delta^{(k+1)} = T$                                  (5.39)
}
```

The purpose of the QR decompositions in equations (5.38) and (5.39) is to orthonormalise the columns of $P^{(k)}$. The same end could be achieved by first taking the Cholesky decomposition

$$\left(\delta^{(k+1)}\right)^{\dagger} \delta^{(k+1)} = T^{\dagger} T \tag{5.40}$$

(choosing $\delta^{(k+1)}$ upper triangular), then setting

$$P^{(k+1)} = T \left(\delta^{(k+1)}\right)^{-1}, \tag{5.41}$$

and my original implementation did just that. However, occasional failures were observed, of the same type that were previously encountered in B-Lanczos. These problems were avoided by switching to the QR decomposition which I implemented using the Modified Gram Schmidt method (MGS).[5] Despite the fact that performing the QR decomposition in this manner enabled B-BiCG(γ_5) to converge in many cases where it had previously failed, there remain nonetheless some grounds for anxiety: if the columns of T were linearly independent (or almost so), then this must be reflected in its QR decomposition, perhaps in one column of $P^{(k+1)}$ being rather poorly determined, or in $\delta^{(k+1)}$ being ill-conditioned. In any event, a robust production code should make some attempt to address this problem. A minimal answer would be to diagnose near-linear independence and restart (perhaps using a different algorithm) if some suitably chosen criterion were triggered. A more ambitious code would drop one column from the solution and attempt to continue on the deflated system.[6]

It is my experience that the stability problems due to the erratic convergence history of BiCG are even more pronounced in the block version. Users unable (or unwilling) to implement B-BiCG in 64-bit precision should choose another algorithm.

[5]As a bonus, the floating point operations count for MGS is roughly half that of equations (5.40) and (5.41).

[6]Since, as will be seen later, the benefits of block algorithms on typical production lattices turned out to be marginal, the (non-trivial) modifications along these lines that would be required in a production code were never undertaken.

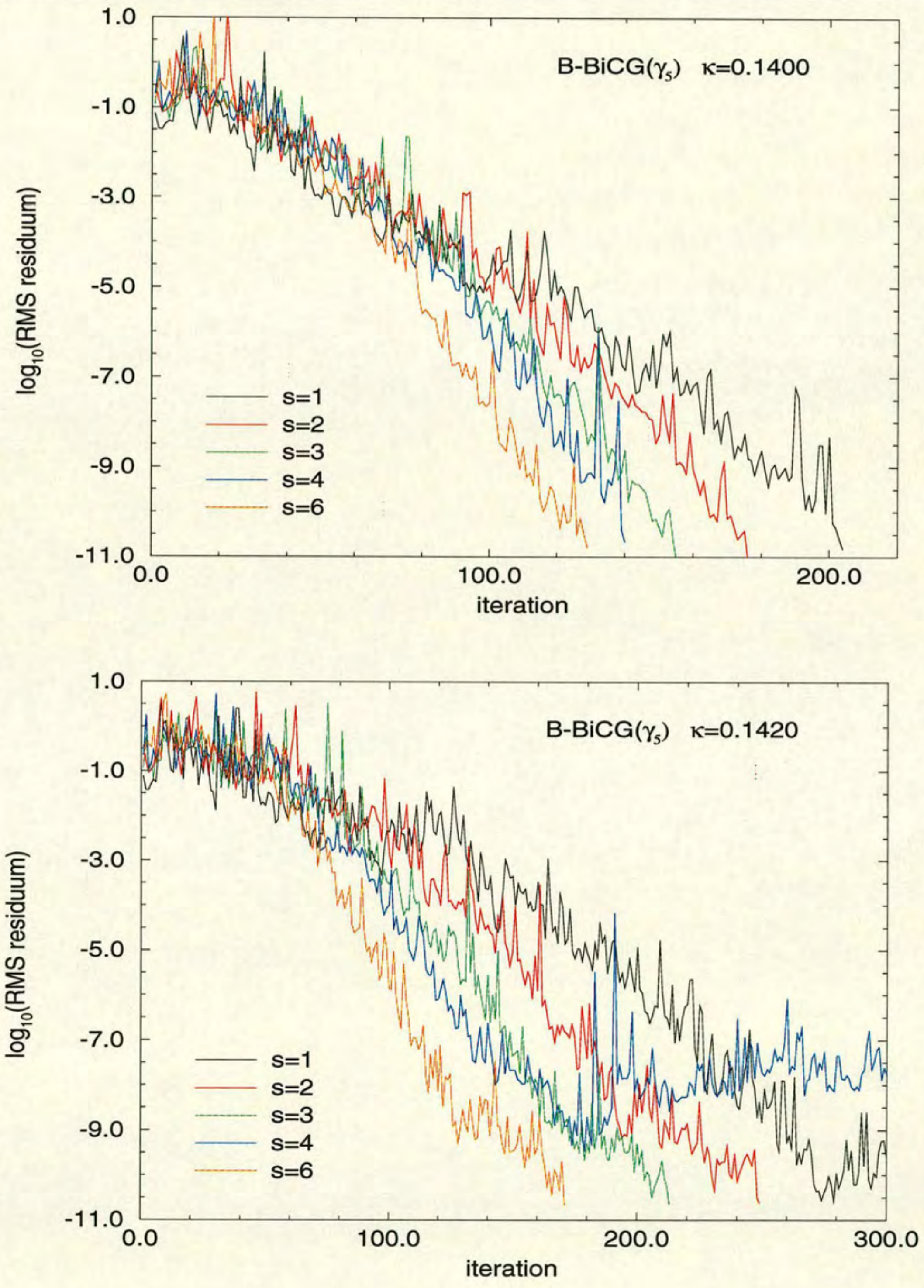


Figure 5.2: Convergence history for B-BiCG(γ_5) at various blocking factors s. $\kappa = 0.14, 0.142$, $V = 8^4$, $\beta = 5.7$, $C_{SW} = 1.5678$ (tadpole-improved).

5.2.5 Block Quasi-Minimal Residual (B-QMR)

The original QMR (Quasi-Minimal Residual) algorithm of Freund and Nachtigal [37] used blocks of variable sizes in look-ahead steps to avoid Lanczos breakdown. Boyse and Seidl [53] described a block version of QMR for complex symmetric matrices, using fixed-size blocks to accelerate convergence. Subsequently Freund and Malhotra [54] discovered a non-hermitian version.[7] The version presented here was developed by me in ignorance of the earlier work of Freund and Malhotra. It follows closely the ideas of [53], and is less general and less robust than the version of [54].

We use a block non-hermitian Lanczos process, constructing two sequences of $n \times s$ matrices $V^{(1)}, \ldots, V^{(n)}$, $W^{(1)}, \ldots, W^{(n)}$ with the property (block bi-orthogonality)

$$\left(W^{(i)}\right)^\dagger V^{(j)} = \delta^{(i)} \delta_{ij}. \tag{5.42}$$

Let

$$\hat{V}^{(k)} = \left[V^{(1)} \ldots V^{(k)}\right] \tag{5.43}$$

and let

$$\hat{T}^{(k)} = \begin{bmatrix} \alpha^{(1)} & \beta^{(1)} & & & \\ \rho^{(2)} & \alpha^{(2)} & \beta^{(2)} & & \\ & \rho^{(3)} & \alpha^{(3)} & \ddots & \\ & & \ddots & \ddots & \beta^{(k-1)} \\ & & & & \alpha^{(k)} \\ & & & & \rho^{(k+1)} \end{bmatrix} \tag{5.44}$$

where the $\alpha^{(*)}$, $\beta^{(*)}$, $\rho^{(*)}$ are $s \times s$ complex matrices. Then the recurrence relations for the $V^{(*)}$ can be written in matrix form:

$$M\hat{V}^{(k)} = \hat{V}^{(k+1)} \hat{T}^{(k)} \tag{5.45}$$

[7]The method of Freund and Malhotra has since been applied to QCD by Fiebach, Frommer and Freund [55]. I point out that their conclusions regarding the practical utility of Block QMR in lattice QCD are more optimistic than my own.

There is a similar relation for the $W^{(*)}$ involving $M^\dagger$. The $W^{(*)}$ can be eliminated using γ_5-symmetry and the special choice $W^{(1)} = \gamma_5 V^{(1)}$.

Let $V^{(1)}\rho^{(1)}$ be the QR decomposition of $R^{(0)} = H - M\Psi^{(0)}$. We aim to construct successive approximations of the form

$$\Psi^{(k)} = \Psi^{(0)} + \hat{V}^{(k)} Z^{(k)} \tag{5.46}$$

for some $ks \times s$ matrix $Z^{(k)}$. Now

$$\begin{aligned} R^{(k)} &= H - M\Psi^{(k)} \\ &= H - M\Psi^{(0)} - M\hat{V}^{(k)} Z^{(k)} \\ &= R^{(0)} - M\hat{V}^{(k)} Z^{(k)} \\ &= V^{(1)}\rho^{(1)} - \hat{V}^{(k+1)}\hat{T}^{(k)} Z^{(k)} \\ &= \hat{V}^{(k+1)} \left[e_1\rho^{(1)} - \hat{T}^{(k)} Z^{(k)} \right] \end{aligned}$$

where the $(k+1)s \times s$ matrix e_1 is given by

$$e_1 = \left[\begin{array}{cccc} I_{s\times s} & 0 & \dots & 0 \end{array} \right]^T \tag{5.47}$$

We would like to choose $Z^{(k)}$ to minimise $\mathrm{Tr}\left(R^{(k)}\right)^\dagger R^{(k)}$ but this is not possible using short recurrences due to the NO-GO theorem of Faber and Manteuffel [31]. Instead the QMR approach is to minimise the norms of the columns of $\left[e_1\rho^{(1)} - \hat{T}^{(k)} Z^{(k)}\right]$. This "quasi-minimisation" can be accomplished using the QR decomposition, and updated cheaply from one iteration to the next.[8]

[8]In order to spare the reader some tedious algebra, we do not give the details here; however, this part of the derivation follows closely the paper by Boyse and Seidl [53] to which the interested reader may refer.

B-QMR(γ_5)

```
$P^{(0)} = P^{(-1)} = V^{(0)} = 0$
$c^{(0)} = b^{(-1)} = b^{(0)} = 0$
$a^{(0)} = d^{(-1)} = d^{(0)} = I$
$R^{(0)} = \tilde{V}^{(1)} = H - M\Psi^{(0)}$
$V^{(1)}\rho_1 = \tilde{V}^{(1)}$                                        (5.48)
$\tilde{\tau}^{(1)} = \rho^{(1)}$
for $k = 1, 2, \ldots$ until convergence do {
    $\delta^{(k)} = \left(V^{(k)}\right)^\dagger \gamma_5 V^{(k)}$
    $\beta^{(k)} = \left(\delta^{(k-1)}\right)^{-1} \left(\rho^{(k)}\right)^\dagger \delta^{(k)}$
    $T = MV^{(k)} - V^{(k-1)}\beta^{(k)}$
    $\alpha^{(k)} = \left(\delta^{(k)}\right)^{-1} \left(V^{(k)}\right)^\dagger \gamma_5 T$
    $\tilde{V}^{(k+1)} = T - V^{(k)}\alpha^{(k)}$
    $V^{(k+1)}\rho^{(k+1)} = \tilde{V}^{(k+1)}$                          (5.49)
    $\theta^{(k)} = b^{(k-2)}\beta^{(k)}$
    $\epsilon^{(k)} = a^{(k-1)}d^{(k-2)}\beta^{(k)} + b^{(k-1)}\alpha^{(k)}$
    $\tilde{\zeta}^{(k)} = c^{(k-1)}d^{(k-2)}\beta^{(k)} + d^{(k-1)}\alpha^{(k)}$
    $\begin{pmatrix} a^{(k)} & b^{(k)} \\ c^{(k)} & d^{(k)} \end{pmatrix}^\dagger \begin{pmatrix} \zeta^{(k)} \\ 0 \end{pmatrix} = \begin{pmatrix} \tilde{\zeta}^{(k)} \\ \rho^{(k+1)} \end{pmatrix}$          (5.50)
    $P^{(k)} = \left(V^{(k)} - P^{(k-1)}\epsilon^{(k)} - P^{(k-2)}\theta^{(k)}\right)\left(\zeta^{(k)}\right)^{-1}$
    $\tau^{(k)} = a^{(k)}\tilde{\tau}^{(k)}$
    $\Psi^{(k)} = \Psi^{(k-1)} + P^{(k)}\tau^{(k)}$
    $\tilde{\tau}^{(k+1)} = c^{(k)}\tilde{\tau}^{(k)}$
}
```

The operations (5.48), (5.49) and (5.50) are QR decompositions. The purpose of (5.50) is to update the QR decomposition at the heart of the quasi-minimisation. I make use of the LAPACK library routines CGEQRF and CUNGQR to implement this operation.[9] The purpose of (5.48) and (5.49) (implemented using MGS) is orthonormalisation of the columns of the Lanczos block-vectors, and the same considerations concerning linear independence mentioned in the previous sections apply here with equal force. My implementation of B-QMR(γ_5) makes no attempt to address this problem, but the authors of [54] show how to bring it under control, at the same time as incorporating look-ahead steps to cure Lanczos breakdowns.

If one were to naïvely expand equation (5.45), one would obtain a superficially different set of recurrence relations for the Lanczos block-vectors $V^{(k)}$ than that given here. However, in exact arithmetic the two methods are equivalent, due to block bi-orthogonality. In finite precision, the method used here helps to slow the inevitable loss of block bi-orthogonality through round-off.[10]

The algorithm does not give a recurrence for the residual $R^{(k)}$, but the 'quasi-residual' Tr $\left(\tilde{\tau}^{(k)}\right)^{\dagger} \tilde{\tau}^{(k)}$ is of the same order of magnitude as Tr $\left(R^{(k)}\right)^{\dagger} R^{(k)}$. When the quasi-residual is small enough, one can calculate the true residual $R^{(k)} = H - M\Psi^{(k)}$.

[9]On Cray systems. On workstations, the equivalent routines are called ZGEQRF and ZUNGQR.

[10]In exact arithmetic, the term $\left(V^{(k)}\right)^{\dagger} \gamma_5 T$ is clearly hermitian, and it is a tempting optimisation to set the anti-hermitian part equal to zero. However, numerical experiments indicate that the small anti-hermitian part that arises in finite arithmetic has an important rôle to play in slowing the loss of bi-orthogonality, and I counsel against this optimisation.

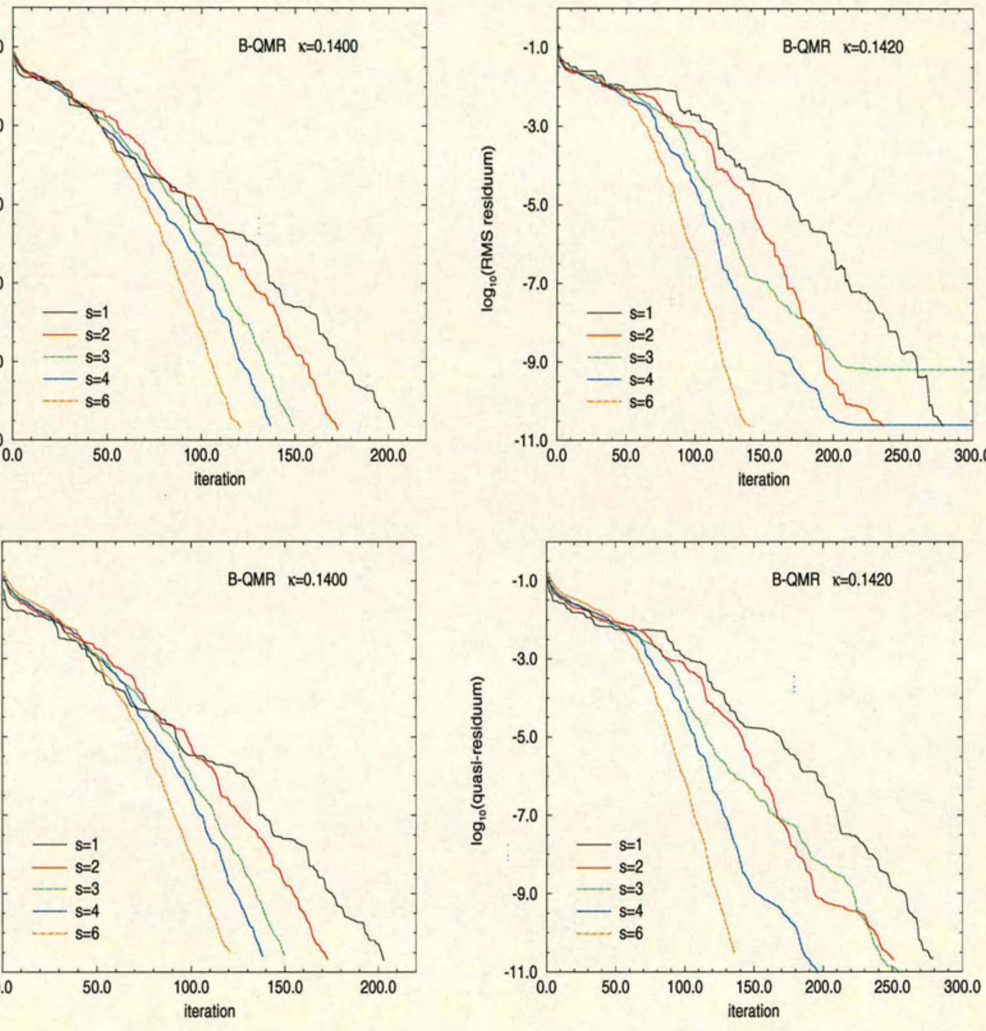


Figure 5.3: Convergence history for B-QMR(γ_5) at various blocking factors s. $\kappa = 0.14, 0.142$, $V = 8^4$, $\beta = 5.7$, $C_{SW} = 1.5678$ (tadpole-improved).

5.3 Implementation

As many of the comments made in section §(4.4) in the context of point algorithms apply equally to block algorithms, I shall here focus (somewhat more briefly) on those aspects which are peculiar to block algorithms.

The basic object that a block algorithm deals with is a block-vector. Here is a Fortran declaration of such an object, taken from the block solver code.

```
CFTRANS Psi :I :I :I :site :
      Fpoint Psi(0:Ncomplex-1,0:Ncolour-1,0:Nspin4-1,
     $           0:Max_body-1,0:Max_block-1)
```

The compile-time parameter `Max_block` determines the maximum blocking factor that the compiled code will be able to handle. There is no parity index as we are working with the red-black preconditioned fermion matrix. These objects may be thought of as arrays of s single-parity fermion fields.

Algorithm	Vectors
B-MR	$3s$
B-CGNR	$4s$
B-Lanczos	$5s$
B-BiCG(γ_5)	$4s$
B-QMR(γ_5)	$7s$

Table 5.1: Vector storage requirements, excluding H, for various block algorithms.

The block algorithm also has to deal with complex $s \times s$ matrices, such as α in operations of the form

$$Y = X\alpha + Y \tag{5.51}$$

or

$$\alpha = Y^{\dagger}X \tag{5.52}$$

in which X and Y are block-vectors. We need the ability to add or multiply two such objects, as well as the ability to invert them. I have written routines to perform the following operations:

Operation	Routine
$\beta = \alpha$	bcopy
$\beta = -\alpha$	bminus
$a_i = \left(\alpha^\dagger \alpha\right)_{ii}$ (no sum)	bmod2
$\gamma = \alpha^{-1}\beta$	bainvb
$\beta = \alpha^\dagger$	bdagger
$\alpha = 0$	bzero
$\alpha = I$	bidentity
$\eta = \alpha\xi + \eta$	baxpy
$\eta = \alpha\xi$ or $\eta = \alpha^\dagger\xi$	bax
$t = \mathrm{Tr}\ \alpha^\dagger\alpha$	btradaga

Table 5.2: Operations involving $s \times s$ matrices and the subroutines that implement them.

The Cholesky decomposition of equation (5.26) and the QR decomposition of equation (5.50) are implemented respectively by the subroutines `bupper_root` and `bqmrqr`. The routines `bainvb` and `bqmrqr` are written to use LAPACK calls; the remainder are written entirely in Fortran. These operations are all local (they require no inter-processor communications) and are replicated on each processing node. Little effort was made to optimise these routines, apart from choosing a data structure to arrange that all components of these matrices were contiguous in memory.

The computational cost of operations involving block-vectors is more significant. Table 5.3 lists the set of subroutines I wrote to provide block-vector support for the algorithms discussed in this chapter.

Routine	Operation	A	B	C	D	E
bfxcapy	$Y = X\alpha + Y$	2	3	4	2	3
bfxcapz	$Y = X\alpha + Z$				1	2
bfxca	$Y = X\alpha$			1		1
bfysx	$Y = Y - X$					
br_diag_mult_r	$X = X\alpha$ or $X = X\alpha^{-1}$, α diagonal					
bfcopy	$Y = X$					
bfzero	$X = 0$					
bfmod2	$\|\|X_i\|\|^2$, $i = 1, \ldots, s$	1	1		1	
bfcdot	$X^\dagger Y$	2	2	1		
bfc_pseudo_dot	$X^\dagger \gamma_5 Y$			1	2	2
bfnorm	$X_i = X_i/\|\|X_i\|\|$, $i = 1, \ldots, s$					
borthonorm	$QR = X$				1	1
brb_matrix	$Y = MX$ or $Y = M^\dagger X$	1	2	1	1	1
bgamma5_matrix	$Y = \gamma_5 Y$			1		

Table 5.3:
Operations performed in each iteration of (A) B-MR, (B) B-CGNR, (C) B-Lanczos, (D) B-BICG(γ_5) and (E) B-QMR(γ_5).

For the most part, it proved convenient and efficient to write these routines to use calls to the more elementary routines of table 4.2. However, the extra level of complexity gives rise to more opportunities for optimisation, and a few wrinkles are worth mentioning.

The floating point operations count to compute $Y^\dagger X$ and $Y^\dagger \gamma_5 X$ can be reduced if it is known that the answer is hermitian, and further reduced if $Y = X$. Accordingly, the interfaces to the subroutines `bfcdot` and `bfc_pseudo_dot` have Boolean arguments indicating whether these conditions are satisfied.

The subroutines `bfmod2`, `bfcdot` and `bfc_pseudo_dot` require that each component of the local result be globally summed over processors. Most of the overhead due to communications latency can be avoided by performing all the local operations first, then using a single call to a vector global summation routine. The price to pay for this optimisation was a proliferation of codes; I took clones of `fmod2`, `fcdot`, `fc_pseudo_dot` and `fc_pseudo_dot_re`, added a suffix `_local` to the name of each, and suppressed the summation over processors.

The operation $Y = X + Y\alpha$ is difficult to implement efficiently, and I decided to make do without such a subroutine. Sometimes the need for such an operation can be avoided by a cunning trick. For example, the last operation in each iteration of B-CGNR is

$$P = T + P\beta,$$

and T will be overwritten at the start of the next iteration

$$T = MP.$$

The trick is to compute instead

$$T = T + P\beta,$$

and then interchange the rôles of T and P.[11]

Under certain conditions (64-bit precision on the Cray-T3D with the `site` index varing slowly), the level 3 BLAS routine CGEMM can be used to perform the bulk of the work in the subroutines `bfcdot` and `bfxcapy`, and there is a compile-time option in the code to enable this. However, somewhat surprisingly, timings on the Cray-T3D indicated that the versions using level 3 BLAS were no faster than the versions relying solely on level 1 BLAS.[12]

[11]In C this would be done by swapping pointers. In Fortran 77, one can achieve the same end by abusing array indices.

[12]The relative efficiencies of level 1 and level 3 BLAS routines are subject to change without notice.

Block algorithms also open up new avenues for optimisation in the matrix multiplication routine, arising from the fact that the coefficient matrix M is now applied to several vectors at once. These avenues have not been explored.

5.4 Scaling

In this section, we discuss how the computational requirements of block algorithms scale with the blocking factor. We are interested in the 'work' required to converge one column of the solution vector Ψ as a function of the blocking factor s. In practice (assuming a dedicated computer), the only truly meaningful way of quantifying 'work' is to measure the elapsed wall-clock time, and this measurement will depend on the architecture used and on the relative performance of the subroutines that implement particular operations on that architecture. To discuss this 'work' more abstractly, *ie.* outwith the context of any particular architecture, we resort to the often misleading method of counting floating point operations,[13] and consequently we are neglecting such items as inter-processor communications.

In general we expect the work W, measured in floating point operations, required to converge one column to be described by the ansatz

$$W = \frac{w(s)}{s}\left[V\left(as + bs^2 + cs\right) + d + es + fs^2 + gs^3\right] \tag{5.53}$$

where $a, b, \ldots, g$ are calculable coefficients to be described below, V is the lattice volume and $w(s)$ is the number of iterations required to converge all s columns. In general, $w(s)$ cannot be determined *a priori* and must be measured; what we can learn from this exercise is how $w(s)$ must behave for the block algorithm to break even with respect to the point algorithm.

The coefficient a is the product of the number of matrix multiplies per iteration and the number of floating point operations per lattice site required to implement

[13]Often in the literature of block algorithms, this 'work' is quantified solely by counting matrix multiplies. This method can only be justified when the cost of a vector-vector operation is completely negligible compared to that of a matrix-vector multiply.

a matrix-vector multiply. The total floating point operations in a matrix-vector multiply is proportional to V by virtue of the structure of the fermion matrix. Were we working with an arbitrary coefficient matrix, the cost of a matrix-vector multiply would be proportional to the square of the order n of the matrix, and for $n \gg s^2$ all other contributions could be neglected.

The principal contributions to the coefficient b are from operations of the form $Y = X\alpha + Z$ and $\beta = Y^\dagger X$ for $n \times s$ matrices X, Y, Z and $s \times s$ matrices α and β. For some algorithms, b also gets a contribution from other operations such as the QR decomposition.

The coefficient c accounts for the remaining vector operations. The major contribution will usually be from the computation of the norms of the columns of the residual block-vector.

The remaining terms involving the other coefficients d, e, f, g account for general control logic and operations with $s \times s$ matrices. These terms, being independent of V, will be neglected here. However, it should be remembered that the overhead of these terms will increase with the number of processors on a parallel architecture as it is impractical to parallelise the underlying operations; in practice they are replicated on each processing node.

For our implementation of B-BiCG(γ_5), counting floating point operations, neglecting terms not proportional to V and re-arranging gives

$$W = \frac{w(s)}{s} V \left(4766s + 72s\left(6s - 1\right)\right). \tag{5.54}$$

The apparent negative value of c is due to the fact that optimisations in $O(s^2)$ vector-vector operations (exploiting hermiticity) give a saving with an $O(s)$ contribution. The value $a = 4766$ is that for non-zero C_{SW} on a scalar architecture; for $C_{SW} = 0$, $a = 2640$.[14]

[14]The overhead due to the clover term has since been reduced greatly by the trick described in Appendix C.

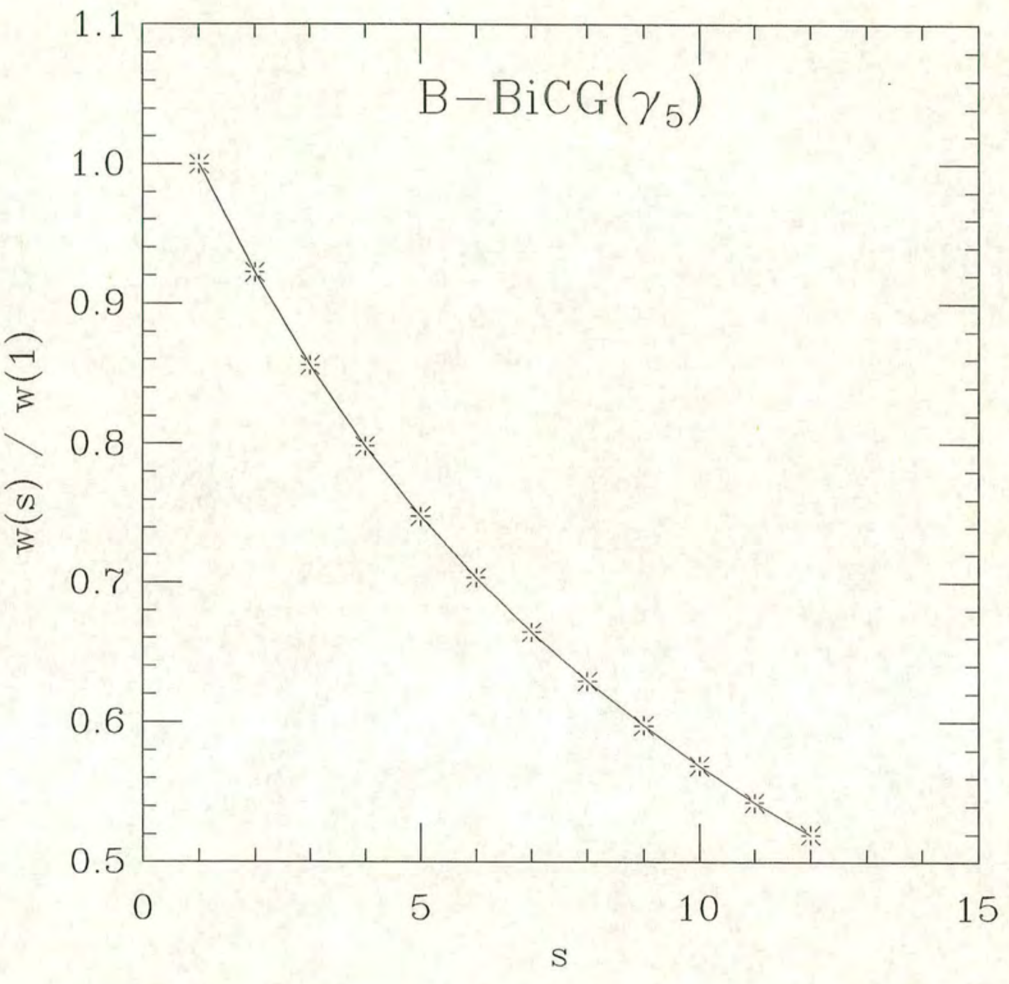


Figure 5.4: The ratio $\frac{w(s)}{w(1)}$, where n is the number of iterations required for the convergence of B-BiCG(γ_5), is plotted against the blocking factor s. The bursts linked by the solid curve (to guide the eye) show the theoretical break-even point based on equation (5.54); a measurement of $\frac{w(s)}{w(1)}$ above (below) this curve would indicate that the block algorithm is slower (faster) than the point algorithm.

5.5 Concluding Remarks

In this chapter, we have studied five block algorithms and their application to quark propagator calculations with Wilson fermions.

We motivated the investigation by observing that, in exact arithmetic, a block version of a Krylov-subspace method will terminate in at most n/s iterations. This gave us cause to hope that block algorithms could reduce significantly the number of matrix-vector multiplications required for convergence. Whether such a reduction would translate into savings in wall-clock time on realistic problems, ie. at values of β, κ and lattice volume used in current and planned physics programmes, was an open question, due to the fact that the overhead of vector-vector operations necessarily grows as s^2. Before answering this question, we first consider the relative merits of the algorithms studied here.

The convergence properties of B-MR are not improved by increasing the blocking factor. This is not surprising, as the algorithm does not enforce any global (bi-)orthogonality among the search directions.

Block Conjugate Gradient must be applied to the normal equation, and so suffers in comparison to the other methods due to the squaring of the condition number. B-CGNR converges smoothly and increasing the blocking factor improves the convergence, as figure 5.1 shows. The same figure also shows several generic features of block algorithms. Firstly, as the problem becomes more difficult (κ approaches κ_c from below), blocking becomes more effective. Secondly, as the blocking factor increases, the convergence history, plotted semi-logarithmically, becomes increasingly concave. An important consequence of the second feature is that the benefit of blocking increases as the convergence criterion is tightened; if performance is one's main criterion for choosing between two algorithms, then one must specify the target residuum at which the comparison is to be made.

Block Lanczos converges more rapidly, if somewhat less reliably, than B-CGNR. The same generic features are present in its convergence histories (figure 5.1), but somewhat obscured by oscillations. It suffers in comparison to B-BiCG(γ_5) and

B-QMR(γ_5) due to the inferior spectral properties of its coefficient matrix $\gamma_5 M$.

B-BiCG(γ_5) (figure 5.2) and B-QMR(γ_5) (figure 5.3) work with the non-hermitian matrix M directly, and they convincingly outperform both B-CGNR and B-Lanczos. However, B-BiCG(γ_5) (figure 5.2) and B-QMR(γ_5) show a weaker improvement as the blocking factor is increased than does B-CGNR.

At the start of this chapter, I suggested that the lattice volume, or order of the coefficient matrix, must have some effect on the viability of block algorithms. This is easy to see at small volumes. Recalling the discussion at the start of section §(5.1), the Krylov-subspace associated with one of the s linear systems is more likely at smaller volumes to be relevant[15] to the solution of another, inasmuch as there are only so many dimensions in the solution space. This accounts for the observations in [51] that, at sufficiently small volumes and small quark mass, the number of iterations for convergence becomes independent of quark mass, and the residuals of all s systems fall catastrophically around n/s iterations.

At large lattice volumes, arguments of this kind cease to hold water. Although some residual effect due to the finite order of the coefficient matrix will persist, the viability of block algorithms will depend more strongly on the spectral properties of the coefficient matrix. In [52], O'Leary showed that Block Conjugate Gradient removes the deleterious effect of the $s-1$ smallest eigenvalues on the asymptotic rate of convergence. If, as expected, similar effects hold for other block algorithms, then at large volumes, blocking will be most useful when M has several small eigenvalues, isolated from the bulk of the spectrum. Although a study of how the eigenvalue spectrum of M depends of β (and κ_{sea}) is well beyond the scope of the present work, McCarthy [50] found ranges in β where Block Conjugate Gradient is particularly effective for staggered fermions.

My more modest purpose was to determine whether block algorithms could usefully be employed to accelerate quark propagator calculations in hadron spec-

[15]In the sense that ψ_i will have a significant overlap with span$\{\eta_j, M\eta_j, \ldots, M^{k-1}\eta_j\}$ for $i \neq j$.

troscopy. The short answer appears to be no.

In recent years, the UKQCD programme of quenched hadron spectroscopy has concentrated on $\beta \geq 6.0$, $V \geq 16^3 \times 48$ and quark masses extending down to the strange mass (or slightly lighter), with C_{SW} fixed by the value of β and the improvement prescription (both tadpole improvement, and more recently, non-perturbative improvement). Even at the lightest of these masses, the smallest of these volumes and the strongest of these couplings, the fastest block algorithms exhibited at best a marginal reduction in wall-clock time on the Cray-T3D compared to their $s = 1$ versions, so that to use a block algorithm in preference to point BiCGSTAB would not have been warranted. Subsequent improvements to the matrix multiplication routine (highly optimised assembly language routines on the Cray-T3E and a better implementation of the clover term as described in appendix C) have weakened the case for block algorithms even further, as the relative cost of matrix-vector to vector-vector operations has fallen. Adding to this the fact that B-BiCG(γ_5) and B-QMR(γ_5) in 32-bit precision cannot consistently achieve our usual target residuum of $10^{-7}||\eta||$, we are lead to the inescapable conclusion that block algorithms are unlikely to impact on the future of quenched hadron spectroscopy.

The outlook is similar for hadron spectroscopy using dynamical fermions. I timed B-BiCG(γ_5) and B-QMR(γ_5) in 64-bit precision on the Cray-T3E, using gauge configurations generated as part of the HMC simulations described in chapter 7,[16] and found that, in terms of wall-clock time, $s = 1$ was optimal for all $\kappa_{\text{valence}} \leq \kappa_{\text{sea}}$.

[16] $\beta = 5.2$, $N_f = 2$, $V = 12^3 \times 24$

Chapter 6

Generalised Hybrid Monte Carlo

In this chapter, I describe the generalised Hybrid Monte Carlo algorithm (GHMC) [56], and show how it may be used for lattice QCD simulations involving two mass-degenerate flavours of dynamical quark. It is not my purpose to review alternative methods, and my treatment of the background theory will be brief. Instead, I focus primarily on the design, implementation and optimisation of the code.

6.1 Introduction

The partition function of lattice QCD in the presence of two mass-degenerate flavours of dynamical quark may be written

$$Z = \int [dU] \det\left(M[U]^\dagger M[U]\right) \exp\left(-S_{pg}[U]\right). \tag{6.1}$$

Our goal is to generate gauge field configurations U, $U_{\mu x} \in SU(3)$, with probability

$$\frac{1}{Z} \det\left(M[U]^\dagger M[U]\right) \exp\left(-S_{pg}[U]\right).$$

For reasons which will be discussed later, we work instead with the red-black preconditioned matrix, by making use of the identities[1]

$$\begin{aligned} \det M &= \det\left(A_{ee}\right)\det\left(A_{oo} - \kappa^2 D_{oe} A_{ee}^{-1} D_{eo}\right) && (6.2) \\ &= \det\left(A_{oo}\right)\det\left(A_{ee} - \kappa^2 D_{eo} A_{oo}^{-1} D_{oe}\right) && (6.3) \end{aligned}$$

which we summarise by [2]

$$\det M = \det\left(A_{\bar{p}\bar{p}}\right)\det\left(\hat{M}_{pp}\right). \tag{6.4}$$

After rewriting the fermionic determinant as an integral over bosonic fields (*pseudofermions*) ϕ defined only on sites of parity p, our problem becomes one of generating gauge field configurations with probability

$$\mathcal{P}\left(U,\phi\right) = \frac{1}{Z_{eff}}\exp\left(-S_{eff}\left[U,\phi\right]\right) \tag{6.5}$$

where

$$S_{eff}\left[U,\phi\right] = S_{pg}\left[U\right] - 2\ln\det A_{\bar{p}\bar{p}} + \phi^\dagger\left(\hat{M}_{pp}^\dagger \hat{M}_{pp}\right)^{-1}\phi \tag{6.6}$$

and

$$Z_{eff} = \int \left[dU\right]\left[d\phi^*\right]\left[d\phi\right]\ \exp\left(S_{eff}\left[U,\phi\right]\right) \tag{6.7}$$

We do this using a generalised Hybrid Monte Carlo algorithm (GHMC).

[1]These follow from properties of determinants and the factorisations

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} = \begin{pmatrix} A & 0 \\ C & I \end{pmatrix}\begin{pmatrix} I & A^{-1}B \\ 0 & D - CA^{-1}B \end{pmatrix} = \begin{pmatrix} I & B \\ 0 & D \end{pmatrix}\begin{pmatrix} A - BD^{-1}C & 0 \\ D^{-1}C & I \end{pmatrix}$$

applied to the fermion matrix in the even-odd basis.

[2]I apologise for the change in notation; in this chapter and the next I use the notation $\hat{M}_{pp}$ to conform with [4, 57].

6.2 The Algorithm

One begins by introducing fictitious momentum fields P, canonically conjugate to U, and then defines a Hamiltonian

$$\mathcal{H}\left[P,U,\phi\right] = \frac{1}{2}\mathrm{Tr}\ P_\mu^2 + S_{eff}\left[U,\phi\right], \tag{6.8}$$

and a partition function

$$Z_{\mathcal{H}} = \int \left[dP\right]\left[dU\right]\left[d\phi^*\right]\left[d\phi\right]\ \exp\left(-\mathcal{H}\left[P,U,\phi\right]\right) \tag{6.9}$$

in which S_{eff} plays the rôle of a potential. The conjugate momenta reside on the lattice links and belong to the $su(3)$ algebra. It is therefore convenient to represent them in terms of the Gell-Mann matrices §(A.3)

$$P_{\mu x} = \sum_{a=1}^{8} P_{\mu x}^a \lambda_a, \tag{6.10}$$

and we need store only 8 real numbers on each link.

The algorithm GHMC, set out schematically overleaf, defines a Markov process which has $\exp\left(-\mathcal{H}\right)$ as its fixed point.[3] Consequently, after equilibration, GHMC will generate P, U, ϕ with probability

$$\begin{aligned}
\mathcal{P}\left(P,U,\phi\right) &= \frac{1}{Z_{\mathcal{H}}}\exp\left(-\mathcal{H}\left[P,U,\phi\right]\right) &(6.11)\\
&= \left(\frac{Z_{eff}}{Z_{\mathcal{H}}}\exp\left(-\frac{1}{2}\mathrm{Tr}\ P_\mu^2\right)\right)\left(\frac{1}{Z_{eff}}\exp\left(-S_{eff}\right)\right) &(6.12)\\
&\equiv \mathcal{P}\left(P\right)\mathcal{P}\left(U,\phi\right). &(6.13)
\end{aligned}$$

Thus the probability distribution factorises and the conjugate momenta have no effect on observables.

[3]The proof for HMC may be found in [58].

GHMC

1. Given an initial configuration of U and P fields at time $\tau = 0$, refresh the pseudofermions ϕ by
$$\phi = \hat{M}_{pp}^{\dagger}\left[U\right]\eta \tag{6.14}$$
where η is a complex vector of zero-mean, unit-variance Gaussian noise.

2. Integrate §(6.3.2) the equations of motion §(6.3.1) to calculate P and U at a time $N_{\mathrm{md}}d\tau$ later.

3. Reverse the sign of P.

4. Metropolis. Accept the new configuration with probability
$$\min\left\{1, \exp\left(-\Delta\mathcal{H}\right)\right\}, \qquad \Delta\mathcal{H} = \mathcal{H}|_{\tau=N_{\mathrm{md}}d\tau} - \mathcal{H}|_{\tau=0} \tag{6.15}$$
otherwise restore the original $\tau = 0$ momentum and gauge fields.

5. Reverse the sign of P.

6. Refresh the momenta by mixing in a real field ξ of Gaussian noise, with mean zero and variance $\frac{1}{2}$.[a]
$$P_{\mu x}^{a} \to P_{\mu x}^{a}\cos\theta + \xi_{\mu x}^{a}\sin\theta, \qquad 1 = 1,\ldots,8 \tag{6.16}$$

7. Go to 1 with the resulting P and U fields.

[a]If we had used $\frac{\lambda_a}{2}$ in equation (6.10), we would use variance 1 here.

The pseudofermions are held fixed throughout the molecular dynamics. Refreshing pseudofermions in the manner of step (1) ensures that the distribution of pseudofermion fields converges simultaneously with the distribution of gauge fields.

The purpose of the molecular dynamics integration in step (2) is to propose a new pair of U and P fields which are distant (in configuration space) from the original pair, at the same time as having a high probability of being accepted in step (4). Violations of energy conservation are only important insofar as they affect the acceptance rate. However, violations of reversibility are crucial as they destroy the correctness of the algorithm

The ergodicity of GHMC arises out of the stochastic nature of the η and ξ in steps (1) and (6). It is the Metropolis accept-reject step (4) that makes the algorithm exact.

The tunable parameters of GHMC are the mixing angle θ, the step size $d\tau$ and the trajectory length N_{md}. The standard Hybrid Monte-Carlo algorithm of [58] is the special case of GHMC for which $\theta = \pi/2$. GHMC with $N_{\text{md}} = 1$ is equivalent to the L2MC of Horowitz [59, 60].

6.3 Integrating the equations of motion

6.3.1 The equations of motion

The Hamiltonian (6.8) gives rise to the equations of motion[4]

$$\begin{aligned}\dot{U}_{\mu x} &= iP_{\mu x}U_{\mu x} && (6.17)\\ \dot{P}_{\mu x} &= -i\mathcal{F}_{\mu x}. && (6.18)\end{aligned}$$

The dots denote differentiation with respect to molecular dynamics time τ. The force term $\mathcal{F}_{\mu x}$ is equal to the traceless part of

$$U_{\mu x}\left(-\frac{\beta}{6}\mathcal{G}_{\mu x}+\kappa\mathcal{W}_{\mu x}-\frac{i}{8}\kappa C_{SW}\mathcal{C}_{\mu x}\right)-h.c.$$

and has contributions originating from the pure gauge ($\mathcal{G}_{\mu x}$), pure Wilson ($\mathcal{W}_{\mu x}$) and clover ($\mathcal{C}_{\mu x}$) parts of the action.

We define fields X and Y

$$X=\begin{pmatrix}X_e\\ X_o\end{pmatrix},\qquad Y=\begin{pmatrix}Y_e\\ Y_o\end{pmatrix} \tag{6.19}$$

which are given on sites of parity p by

$$\begin{aligned}X_p &= \left(\hat{M}^\dagger_{pp}\hat{M}_{pp}\right)^{-1}\phi && (6.20)\\ Y_p &= \hat{M}^\dagger_{pp}X_p && (6.21)\end{aligned}$$

and on the other parity $\bar{p}$ by

$$\begin{aligned}X_{\bar{p}} &= \kappa A^{-1}_{\bar{p}\bar{p}}D_{\bar{p}p}X_p && (6.22)\\ Y_{\bar{p}} &= \kappa A^{-1}_{\bar{p}\bar{p}}D^\dagger_{\bar{p}p}Y_p. && (6.23)\end{aligned}$$

[4]For a detailed derivation of the equations of motion, the reader is referred to [4] or [57], and useful background can be found in [61].

Labelling sites of parity p by x_p, we denote by $(X_p)_{x_p}$ that part of X_p which is local to site x_p, and we treat it as a 12-component column vector. It is convenient also to define

$$\begin{aligned}\Lambda x_p &= (Y_p)_{x_p} (X_p)^\dagger_{x_p} + (X_p)_{x_p} (Y_p)^\dagger_{x_p} && (6.24)\\ \Lambda x_{\bar{p}} &= (Y_{\bar{p}})_{x_{\bar{p}}} (X_{\bar{p}})^\dagger_{x_{\bar{p}}} + (X_{\bar{p}})_{x_{\bar{p}}} (Y_{\bar{p}})^\dagger_{x_{\bar{p}}} + 2\left(A^{-1}_{\bar{p}\bar{p}}\right)_{x_{\bar{p}}}. && (6.25)\end{aligned}$$

Each Λ_x has 144 components, and the object $\mathrm{Tr}_{\mathrm{spin}}\Lambda_x$ is the 3×3 complex matrix that results when Λ_x is traced over its hidden spin indices. Explicit forms for $\mathcal{G}_{\mu x}$, $\mathcal{W}_{\mu x}$, and $\mathcal{C}_{\mu x}$ can now be given in terms of the X, Y, Λ and the $SU(3)$ link variables $U_{\mu x}$.

$$\begin{aligned}\mathcal{G}_{\mu x} &= \sum_\nu \left(U_{\nu x+\hat\mu} U^\dagger_{\mu x+\hat\nu} U^\dagger_{\nu x} + U^\dagger_{\nu x+\hat\mu-\hat\nu} U^\dagger_{\mu x-\hat\nu} U_{\nu x-\hat\nu}\right) && (6.26)\\ \mathcal{W}_{\mu x} &= \mathrm{Tr}_{\mathrm{spin}} \left[Y_{x+\hat\mu} X^\dagger_x (1+\gamma_\mu) + X_{x+\hat\mu} Y^\dagger_x (1-\gamma_\mu)\right] && (6.27)\\ \mathcal{C}_{\mu x} &= \sum_\nu \mathrm{Tr}_{\mathrm{spin}}\left[\sigma_{\mu\nu}\Lambda_{x+\hat\mu}\right] U_{\nu x+\hat\mu} U^\dagger_{\mu x+\hat\nu} U^\dagger_{\nu x}\\ &+ \sum_\nu U_{\nu x+\hat\mu} \mathrm{Tr}_{\mathrm{spin}}\left[\sigma_{\mu\nu}\Lambda_{x+\hat\mu+\hat\nu}\right] U^\dagger_{\mu x+\hat\nu} U^\dagger_{\nu x}\\ &+ \sum_\nu U_{\nu x+\hat\mu} U^\dagger_{\mu x+\hat\nu} \mathrm{Tr}_{\mathrm{spin}}\left[\sigma_{\mu\nu}\Lambda_{x+\hat\nu}\right] U^\dagger_{\nu x}\\ &+ \sum_\nu U_{\nu x+\hat\mu} U^\dagger_{\mu x+\hat\nu} U^\dagger_{\nu x} \mathrm{Tr}_{\mathrm{spin}}\left[\sigma_{\mu\nu}\Lambda_x\right]\\ &- \sum_\nu \mathrm{Tr}_{\mathrm{spin}}\left[\sigma_{\mu\nu}\Lambda_{x+\hat\mu}\right] U^\dagger_{\nu x+\hat\mu-\hat\nu} U^\dagger_{\mu x-\hat\nu} U_{\nu x-\hat\nu}\\ &- \sum_\nu U^\dagger_{\nu x+\hat\mu-\hat\nu} \mathrm{Tr}_{\mathrm{spin}}\left[\sigma_{\mu\nu}\Lambda_{x+\hat\mu-\hat\nu}\right] U^\dagger_{\mu x-\hat\nu} U_{\nu x-\hat\nu}\\ &- \sum_\nu U^\dagger_{\nu x+\hat\mu-\hat\nu} U^\dagger_{\mu x-\hat\nu} \mathrm{Tr}_{\mathrm{spin}}\left[\sigma_{\mu\nu}\Lambda_{x-\hat\nu}\right] U_{\nu x-\hat\nu}\\ &- \sum_\nu U^\dagger_{\nu x+\hat\mu-\hat\nu} U^\dagger_{\mu x-\hat\nu} U_{\nu x-\hat\nu} \mathrm{Tr}_{\mathrm{spin}}\left[\sigma_{\mu\nu}\Lambda_x\right] && (6.28)\end{aligned}$$

6.3.2 Leapfrog integration

The integration scheme used to integrate the equations of motion in the molecular dynamics part of GHMC is required to be area-preserving and reversible. Symmetric, symplectic integrators satisfy this requirement, and perhaps the simplest of these is the so-called "leapfrog" integration scheme.

We define operators $\mathcal{T}_U$ and $\mathcal{T}_P$ by

$$\mathcal{T}_U(t)\begin{pmatrix} U \\ P \end{pmatrix} = \begin{pmatrix} \exp(itP)\,U \\ P \end{pmatrix} \tag{6.29}$$

$$\mathcal{T}_P(t)\begin{pmatrix} U \\ P \end{pmatrix} = \begin{pmatrix} U \\ P - it\mathcal{F} \end{pmatrix} \tag{6.30}$$

and then use the compound operator

$$\mathcal{T}(t) = \mathcal{T}_P(t/2)\,\mathcal{T}_U(t)\,\mathcal{T}_P(t/2) \tag{6.31}$$

to integrate equations (6.17) and (6.18) through one timestep $d\tau$:

$$\begin{pmatrix} U(d\tau) \\ P(d\tau) \end{pmatrix} = \mathcal{T}(d\tau)\begin{pmatrix} U(0) \\ P(0) \end{pmatrix} \tag{6.32}$$

A molecular dynamics trajectory is integrated by repeated application of equation (6.32):

$$\begin{aligned} \begin{pmatrix} U(N_{\mathrm{md}}d\tau) \\ P(N_{\mathrm{md}}d\tau) \end{pmatrix} &= \mathcal{T}(d\tau)^{N_{\mathrm{md}}}\begin{pmatrix} U(0) \\ P(0) \end{pmatrix} \\ &= \mathcal{T}_P(d\tau/2)\left(\mathcal{T}_U(d\tau)\,\mathcal{T}_P(d\tau)\right)^{N_{\mathrm{md}}-1}\mathcal{T}_U(d\tau)\,\mathcal{T}_P(d\tau/2) \end{aligned} \tag{6.33}$$

in which we used the fact that $\mathcal{T}_P(t/2)^2 = \mathcal{T}_P(t)$. The integration scheme (6.33) conserves energy up to terms $\mathcal{O}(d\tau)^2$.

Following a suggestion of Sexton and Weingarten [62], we can decompose $\mathcal{F}_{\mu x}$

into a pure gauge part and a fermionic part.

$$\mathcal{F}_{\mu x} = \mathcal{F}_{\mu x}^{(pg)} + \mathcal{F}_{\mu x}^{(f)} \tag{6.34}$$

The pure gauge part $\mathcal{F}_{\mu x}^{(pg)}$ consists of those terms in the definition of $\mathcal{F}_{\mu x}$ which are not proportional to κ. The fermionic part $\mathcal{F}_{\mu x}^{(f)}$ is much more expensive to compute as it depends on the fields X and Y. We define corresponding operators in the obvious way:

$$\mathcal{T}_P^{(pg)}(t)\begin{pmatrix} U \\ P \end{pmatrix} = \begin{pmatrix} U \\ P - it\mathcal{F}^{(pg)} \end{pmatrix} \tag{6.35}$$

$$\mathcal{T}_P^{(f)}(t)\begin{pmatrix} U \\ P \end{pmatrix} = \begin{pmatrix} U \\ P - it\mathcal{F}^{(f)} \end{pmatrix}, \tag{6.36}$$

and observe that

$$\mathcal{T}_P(t) = \mathcal{T}_P^{(pg)}(t)\,\mathcal{T}_P^{(f)}(t) = \mathcal{T}_P^{(f)}(t)\,\mathcal{T}_P^{(pg)}(t). \tag{6.37}$$

We obtain a *generalised leapfrog scheme* by using the compound operator

$$\mathcal{T}_{\mathcal{G}}(t) = \mathcal{T}_P^{(f)}\left(\frac{t}{2}\right)\left[\mathcal{T}_P^{(pg)}\left(\frac{t}{2n_{\mathcal{G}}}\right)\mathcal{T}_U\left(\frac{t}{n_{\mathcal{G}}}\right)\mathcal{T}_P^{(pg)}\left(\frac{t}{2n_{\mathcal{G}}}\right)\right]^{n_{\mathcal{G}}}\mathcal{T}_P^{(f)}\left(\frac{t}{2}\right) \tag{6.38}$$

for $n_{\mathcal{G}} = 1, 2, \ldots$ in place of $\mathcal{T}(t)$ in equation (6.33). The standard leapfrog scheme is recovered by setting $n_{\mathcal{G}} = 1$ and making use of equation (6.37). The hope is that better energy conservation can be had for $n_{\mathcal{G}} > 1$. In some regimes, we find that $n_{\mathcal{G}} = 2$ provides a modest improvement in acceptance rate at little additional cost, but that further increases are rarely worthwhile [4]. We generally use $n_{\mathcal{G}} = 1$ or 2 in production.

6.4 Solvers and preconditioning in GHMC

The solution X_p to

$$\hat{M}_{pp}\,[U]^\dagger\,\hat{M}_{pp}\,[U]\,X_p = \phi \tag{6.39}$$

must be computed each time the gauge fields are altered. This is the most computationally demanding part of GHMC, and hence the most promising target for optimisation.

The obvious, and traditional, strategy is to use the Conjugate Gradient Method with the hermitian, positive and (hopefully) definite $\hat{M}^\dagger_{pp}\hat{M}_{pp}$ as the coefficient matrix. It was in anticipation of this that we chose to re-express the fermionic determinant in terms of the red-black preconditioned matrix (equation (6.4)). If instead we had formulated GHMC on the full matrix M with pseudofermions defined on both parities, we would now be dealing with the larger system

$$M^\dagger M X = \phi$$

which is not amenable to red-black preconditioning due to the fact that $M^\dagger M$ involves next-to-nearest neighbour couplings.

Although Conjugate Gradient is in some sense optimal for an arbitrary hermitian positive definite coefficient matrix,[5] it is possible here to exploit the fact that the coefficient matrix is given in factored form, computing X_p with two successive solves

$$1. \qquad \hat{M}^\dagger_{pp} Y_p = \phi \tag{6.40}$$

$$2. \qquad \hat{M}_{pp} X_p = Y_p \tag{6.41}$$

using an efficient non-hermitian solver such as BiCGSTAB. The hope is that reduction in condition number will offset the additional solve. We find that the two-step method using BiCGSTAB yields very significant performance gains (of

[5]It is optimal amongst iterative methods that deal with the same Krylov subspace.

up to 40%) on the Cray T3E, Cray T3D and Alpha workstations.[6] To eliminate the effect of the imperfect convergence of Y_p on the accuracy of X_p, we "polish" X_p with a Conjugate Gradient restart.

It should be pointed out that the two-step solve method can use red-black preconditioning regardless of whether GHMC is formulated to use the full fermion matrix or the red-black preconditioned matrix.

In section §(4.2), I alluded to the possibility of applying a further block Jacobi preconditioner to the red-black preconditioned system. We actually use that preconditioning here. Equation (6.41) can be preconditioned by left-multiplying by A_{pp}^{-1}. Since A_{pp} is hermitian, equation (6.40) can be preconditioned in the same way. To handle equation (6.39), we use central preconditioning, and solve the transformed system

$$\left(A_{pp}^{-1}\hat{M}_{pp}^{\dagger}\hat{M}_{pp}A_{pp}^{-1}\right)\left(A_{pp}X_p\right) = A_{pp}^{-1}\phi, \tag{6.42}$$

in order to work with a coefficient matrix that is manifestly hermitian and positive.

I set out the Conjugate Gradient algorithm, with explicit preconditioning, below. All vectors are defined on one parity only, and I have suppressed all parity indices for notational convenience. $R^{(k)} = \phi - \hat{M}^{\dagger}\hat{M}X^{(k)}$ is the residual after the kth iteration. The preconditioner $V = V_1V_2$ is applied centrally ($V_1 = V_2 = C$), modifying the iterates

$$X^{(k)} \rightarrow \tilde{X}^{(k)} = CX^{(k)} \tag{6.43}$$

and residuals

$$R^{(k)} \rightarrow \tilde{R}^{(k)} = C^{-1}R^{(k)}. \tag{6.44}$$

In practice, we take either $C = A_{pp}$ or $C = I$.

[6] This conclusion is, at least in part, architecture specific. Some groups claim no advantage to BiCGSTAB over Conjugate Gradient on APE/Quadrics machines.

Conjugate Gradient

```
$R^{(0)} = \phi - \hat{M}^\dagger \hat{M} X^{(0)}$
$\tilde{X}^{(0)} = C X^{(0)}$
$P^{(0)} = \tilde{R}^{(0)} = C^{-1} R^{(0)}$
$\rho_0 = \left(\tilde{R}^{(0)}\right)^\dagger \tilde{R}^{(0)}$
for $k = 0, 1, 2, \ldots$ until convergence do {
    $T = \hat{M} C^{-1} P^{(k)}$
    $\pi = T^\dagger T$
    $\alpha = \pi^{-1} \rho_k$
    $\tilde{X}^{(k+1)} = P^{(k)} \alpha + \tilde{X}^{(k)}$
    $Z = C^{-1} \hat{M}^\dagger T$
    $\tilde{R}^{(k+1)} = -Z\alpha + \tilde{R}^{(k)}$
    $\rho_{k+1} = \left(\tilde{R}^{(k+1)}\right)^\dagger \tilde{R}^{(k+1)}$
    $R^{(k+1)} = C \tilde{R}^{(k+1)}$                                  (6.45)
    $\beta = \rho_k^{-1} \rho_{k+1}$
    $P^{(k+1)} = \tilde{R}^{(k+1)} + P^{(k)} \beta$
}
$X = C^{-1} \tilde{X}$
```

An interesting and sometimes useful aspect of Conjugate Gradient is that estimates of the smallest and largest eigenvalues (and hence condition number) of the coefficient matrix can be obtained at little extra cost.[7] The key observation, by properties of the Conjugate Gradient algorithm, is that the matrix

$$T^{(k)} = \begin{pmatrix} \left(r^{(0)}\right)^\dagger \\ \vdots \\ \left(r^{(k)}\right)^\dagger \end{pmatrix} C\hat{M}^\dagger \hat{M} C \left(r^{(0)} \ \ldots r^{(k)}\right), \qquad r^{(j)} \equiv \frac{\tilde{R}^{(j)}}{\sqrt{\left(\tilde{R}^{(j)}\right)^\dagger \tilde{R}^{(j)}}}$$

is real, symmetric and tridiagonal, with non-zero components that can be easily extracted from coefficients computed during the iteration. Since the $\left\{r^{(j)}\right\}$ are orthonormal, we are in effect accomplishing a Lanczos tridiagonalisation of the coefficient matrix. When Conjugate Gradient converges after (say) m iterations, the extremal eigenvalues of $T^{(m)}$ are taken as approximations to those of $C\hat{M}^\dagger \hat{M} C$. Agreement to five or six significant figures is routinely achieved.

The BiCGSTAB algorithm that we use in GHMC is set out below, with explicit preconditioning. Once again, all vectors are defined on one parity only, and parity indices have been suppressed. L is an optional left preconditioner which modifies the residuals $R^{(k)} \to \tilde{R}^{(k)} = L^{-1} R^{(k)}$ but leaves the iterates $X^{(k)}$ unchanged. In practice, we take either $L = A_{pp}$ or $L = I$. The version given here is appropriate for the solution of equation (6.41); to handle equation (6.40), simply replace $X \to Y$ and $\hat{M} \to \hat{M}^\dagger$ wherever they occur.

[7] I am grateful to James Sexton for bringing this to my attention.

BiCGSTAB

```
$R^{(0)} = \phi - \hat{M}X^{(0)}$
$\tilde{R}^{(0)} = L^{-1}R^{(0)}$
$V = P = 0$
$\rho_0 = \alpha = \omega = 1$
for $k = 0, 1, 2, \ldots$ until convergence do {
    $\rho_{k+1} = \left(\tilde{R}^{(0)}\right)^{\dagger} \tilde{R}^{(k)}$
    $\beta = (\rho_{k+1}/\rho_k)(\alpha/\omega)$
    $P = \tilde{R}^{(k)} + \beta(P - \omega V)$
    $V = L^{-1}\hat{M}P$
    $\alpha = \rho_{k+1}/\left(\tilde{R}^{(0)}\right)^{\dagger} V$
    $S = \tilde{R}^{(k)} - \alpha V$
    $T = L^{-1}\hat{M}S$
    $\omega = T^{\dagger}S/T^{\dagger}T$
    $X^{(k+1)} = X^{(k)} + \omega S + \alpha P$
    $\tilde{R}^{(k+1)} = S - \omega T$
    $R^{(k+1)} = L\tilde{R}^{(k+1)}$                                    (6.46)
}
```

I found that this block Jacobi preconditioning typically reduced the number of iterations for BiCGSTAB to converge on equations (6.40) and (6.41) by some 15%. The improvement for Conjugate Gradient on equation (6.39) was even more significant, at some 25%. These savings in the number of iterations do translate into similar savings in wall-clock time for the following reasons.

Firstly, there is no significant overhead in the multiplication by the coefficient matrix. Using preconditioned BiCGSTAB to solve equation (6.41), the coefficient matrix

$$\hat{M}_{pp} = A_{pp} - \kappa^2 D_{p\bar{p}} A_{\bar{p}}^{-1} D_{\bar{p}p} \tag{6.47}$$

is replaced by

$$A_{pp}^{-1} \hat{M}_{pp} = I - \kappa^2 A_{pp}^{-1} D_{p\bar{p}} A_{\bar{p}}^{-1} D_{\bar{p}p} \tag{6.48}$$

so we have effectively traded a multiplication by A_{pp} for a multiplication by its inverse, and the costs of these two operations are approximately equal.[8] Similar remarks (but with different coefficient matrices) apply to BiCGSTAB on equation (6.40) and to CG on equation (6.39).

Secondly, it is possible to skip the updates of the residuals of the original system in equations (6.45) and (6.46). One must then be content to base convergence criteria on the preconditioned residuum, but this is reasonably satisfactory as the ratio of the two residua is typically of order unity. In my implementation, I only compute the residual of the original system after the preconditioned residuum is within $\sqrt{2}$ of the convergence target.

6.5 Implementation, verification and optimisation

6.5.1 Operational aspects

Run-times for dynamical fermion simulations are typically measured in months or even years. This simple fact has important consequences which informed the

[8]Actually, at the time of writing, the cost of multiplication by the inverse is actually less than that of the forward multiply, which has not yet been rewritten to exploit the trick of §(6.5.5).

design of our code. We wanted to make maximum use of the available computing resources, at the same time as minimising the impact of the inevitable system breaks. Furthermore, we tried to allow for improved modules to be swapped into the production code during a production run. I now describe three features of our implementation which did much to achieve these ends.

The first of these is checkpointing. The essential purpose of checkpointing is damage control; by recording the state of the simulation periodically we can recover from a system crash without losing too great an amount of precious computing time. At the end of each trajectory, the state of the simulation is completely defined by the fields U, P, the state of the random number generator, and the run-time options; during a trajectory, rather more information is required to specify the state. It is, therefore, both natural and convenient to implement checkpointing by saving to disk the state of the simulation at the end of an integral number of trajectories.[9] Lest a disk failure occur while a checkpoint is being written, we work with a cycle of (at least) two sets of checkpoint files.

The second feature is a facility to stop the simulation tidily and on demand. This was achieved by including in the code at the end of every trajectory a test for the existence of a flag file; if the file is found to exist, the program takes a checkpoint and stops. The operational flexibility that this feature provided to us proved extremely invaluable in numerous situations, including several that we had not foreseen.

The third feature is a robust and flexible parameter file reader. The program reads its run-time options from a free-format text file consisting of keyword and value pairs as well as optional embedded comments.[10] Such a flexible format does much to reduce time lost due to human error, as will be appreciated by anyone who has had to work with parameter files that depend on specifying options in

[9]On large volumes at light masses we generally checkpoint at the end of every trajectory, but the ability to checkpoint less frequently was used on cheaper simulations where a trajectory takes only a few minutes to complete.

[10]The currently supported keywords and their meanings are listed in D.

the correct sequence.[11] Moreover, it enables a new version of the code using new run-time options to be swapped in painlessly during a simulation.[12] Our parameter file strategy also facilitated the implementation of checkpointing; it was a simple matter to write to disk, as part of every checkpoint, a complete parameter file defining everything necessary to resume the simulation from the latest checkpoint; having written a checkpoint we also record the location of the associated parameter file. Similar parameter files are saved along with every gauge configuration that is stored for subsequent analysis; these serve to document the options used in the generation of the gauge configuration (including a definition of the file format) and to record information from which the integrity of the files themselves can be ascertained (time-sliced plaquette and checksums).

These three features are supplemented by a suite of shell scripts. Once a simulation is deemed to have equilibrated, little operator intervention is required other than occasionally to resume the run. The script that does this function (1) ensures that the simulation is not already running by testing for the non-existence of a lock file, and if no lock file is found, creates a lock file and continues,[13] (2) finds the latest checkpoint parameter file and takes a copy to be read by the GHMC code,[14] (3) invokes the GHMC code itself, and once this has completed (4) deletes the lock file created in (1), and finally, if no flag file exists to indicate that the simulation has been stopped on demand, (5) submits to the NQS queue a job which runs this script again. Step (5) ensures that full use is made of the machine's uptime. Other scripts in the suite fulfill essential housekeeping functions such as bundling and archiving of the generated gauge configurations and their conjugate momenta.

[11]Even worse is the all too common scenario in which options can only be changed by recompiling the code.

[12]An earlier methodology in UKQCD's MPP codes was to delegate all parameter file parsing to a separate program. This resulted in an unhealthy inertia, as even a minor change to the parameters of one application necessarily assumed system-wide proportions.

[13]Such precautions are unfortunately necessary on a shared machine.

[14]The parameter file reader is written to ignore duplicated keywords, so this is a convenient place to insert any parameter overrides.

6.5.2 Code verification

Verifying the correctness of a code of this magnitude is no simple task. For the most part, *unit testing* of the component modules presents no particular difficulty, although constructing test harnesses and test data is not always straightforward. The full scope of the problem becomes apparent once the individual modules have been integrated into a single code and the obvious bugs (those that prevent the code from running to completion) have been ironed out. Our ultimate concern is that the code produces the correct results. In the solver, we always had the ability to verify the solution by the simple expedient of substitution, but no such possibility exists here. We can gain some confidence in our code by reproducing published results, insofar as any are available, but sooner or later we shall have to venture into uncharted waters in parameter space.

Tests which depend on the correctness of the code as a whole are therefore extremely precious, and I identify some of these below.

1. *Reproducibility.* If the same code is run twice from the same starting conditions, the results of the second run should reproduce those of the first. This requirement, without which one could never develop satisfactory confidence in the code's correctness, is not as trivial as it might seem at first sight,[15] and actually influenced the design of our code. For example, in the interests of economy, we store on disk only two rows of the gauge field configurations, reconstructing the third as required from the condition that each link matrix is special and unitary; in order to ensure reproducibility it is necessary to reunitarise the gauge fields in memory whenever they are written to disk.[16] Similarly, although the conjugate momenta are reinitialised at the

[15]For example, changing the processor grid will induce tiny changes in global summations, and these will be magnified by the chaotic dynamics of GHMC. Moreover, although the gauge fields and conjugate momenta are stored in a format which is independent of the processor grid, the state of the the random number generator is not currently defined in a grid-independent way.

[16]In finite precision, reunitarising a reunitarised gauge field will induce in it a tiny but measurable change.

start of every trajectory in pure HMC, in our current implementation they must be included in every checkpoint to ensure reproducibility.[17]

2. *Energy conservation and scaling.* Although HMC defines a conservative dynamical system, the discrete integration scheme that we employ to compute the evolution of U and P means that the molecular dynamics trajectory will depart from the $\Delta\mathcal{H} = 0$ surface. A small $\Delta\mathcal{H}$ is necessary to ensure a satisfactory acceptance rate; we therefore require at the very least that our code can achieve a sufficiently small $\Delta\mathcal{H}$, if necessary after "reasonable" tuning of N_{md} and $d\tau$. More powerful tests can be designed around the observation that in any symplectic integration scheme $\Delta\mathcal{H}$ should scale (to leading order) as a power of the timestep $d\tau$; such tests were conducted on our code in [4] for the leapfrog scheme that we employ in production and for several higher-order integration schemes, and the results were found to be satisfactory.

3. *Reversibility.* In order to ensure detailed balance the molecular dynamics trajectory must be reversible. We measure violations of reversibility on an intial P and U by integrating the equations of motion forward through molecular dynamics time $N_{\mathrm{md}}d\tau$ to obtain P' and U' (the *forward trajectory*), then reversing the sign of P' and integrating through a further $N_{\mathrm{md}}d\tau$ to obtain P'' and U'' (the *reverse trajectory*), and finally computing

$$\delta\Delta U = \sqrt{\sum_{x,\mu}\sum_{i,j}\left|\left(U''_{\mu,x}\right)_{ij} - \left(U_{\mu,x}\right)_{ij}\right|^2} \tag{6.49}$$

$$\delta\Delta P = \sqrt{\sum_{x,\mu}\sum_{a=1}^{8}\left|\left(P''_{\mu,x}\right)_{a} + \left(P_{\mu,x}\right)_{a}\right|^2} \tag{6.50}$$

and

$$\delta\Delta\mathcal{H} = \mathcal{H}\left(U'', P''\right) - \mathcal{H}\left(U, P\right). \tag{6.51}$$

[17]The GHMC algorithm requires initial conjugate momenta; if no starting P is supplied our program will generate one, and the net effect is to boost the state of the random number generator, destroying reproducibility.

The plus sign in equation (6.50) compensates for the fact that P'' is the negative of what would be obtained by changing the sign of $d\tau$ instead of P'. The quality of these measurements should be interpreted with reference to the best that one could hope to achieve, given the lattice volume and precision used; for example, $\delta\Delta U$ less than

$$\epsilon\left(\sum_{x,\mu}\sum_{i,j}\left|(U_{\mu,x})_{ij}\right|^2\right)^{\frac{1}{2}}$$

where ϵ is the machine epsilon, is not realistically attainable. If all solves are done exactly, employing a symplectic integration scheme guarantees the reversibility of the trajectory up to machine precision. The impact of inexact solves on reversibility can be eliminated by adopting a strategy whereby the initial guesses at the solutions are chosen reversibly; in practice this means that the initial guesses for X_p and/or Y_p should be constant throughout the simulation, the zero vector being the obvious choice.[18]

4. The quantity $\langle\exp(-\Delta\mathcal{H})\rangle$, the average being taken over trajectories in an equilibrated Markov chain, should agree within errors to unity. This provides both a useful diagnostic for deciding whether a simulation has thermalised, and an extremely potent check of the code.

Although these tests are powerful, it is nonetheless possible to have an incorrect code that passes them all. During the early stages of system testing, we had a reproducible, energy-conserving, reversible code and a thermalised simulation, which failed to reproduce published results.[19] The average plaquette is one quan-

[18]It is for this reason that we decided in the design phase to eschew educated guessing techniques, amongst which the chronological inversion method [63] is the most sophisticated. However, I did experiment briefly with some non-reversible guessing strategies, including the simple expedient of taking for the initial guess(es) the solution(s) from the previous timestep; I saw significant savings in solver iterations at the expense of disturbingly large reversibility violations.

[19]The problem turned out to be that we were using the wrong variance when refreshing pseudofermions; the error could be swept up in a redefinition of β and $d\tau$, so that it appeared to all intents and purposes as if we were simulating at the wrong value of β.

tity that provides a convenient reference. We have run simulations in both the pure gauge ($\kappa = 0$) and pure Wilson ($\kappa \neq 0$, $C_{SW} = 0$), and reproduced published results for the plaquette, within errors. It was more difficult to verify our code for non-zero C_{SW}, owing to the lack of published data. We are indebted to Matt Wingate and the MILC collaboration for agreeing to run a small HMC simulation at non-zero C_{SW} using their own code, and to Karl Jansen for some suggestions on how to test the clover part of the action using a cumulant expansion.

6.5.3 Optimisation – architectural factors

Performance on the Cray-T3E is dominated by memory bandwidth. Although the memory system of the T3E is much improved relative to the T3D (streams, secondary cache and E-registers), the disproportionately faster processors[20] mean that memory optimisations are actually more important on the T3E than on the T3D. The fact that the processors support multiple instruction issue coupled with the increased complexity of the memory system makes it a good deal more difficult to take full advantage of the machine's capabilities. Typically, even highly optimised Fortran code can sustain only a small fraction of the theoretical peak speed, so key routines *must* be written in assembly language. As a by-product of the tender process that led to the acquisition of the Cray-T3E at Edinburgh, we were fortunate to obtain a set of highly optimised routines implementing multiplication by the fermion matrix. Since then a number of additional routines have been rewritten in assembler,[21] and currently less than 25% of the run-time is spent executing Fortran code.

A significant optimisation on the Cray-T3E is to use 32-bit instead of 64-bit floating point numbers to represent the gauge, conjugate momenta and pseudo-fermion fields. The improvement in speed (worth a factor of about 1.7 to us) is

[20]The Alpha EV5.6 processors are clocked at 450 MHz and are capable of issuing two floating point instructions in each clock cycle for a peak performance of 900 MFlops per processing element.

[21]We are indebted to Stephen Booth for several assembler routines manipulating 3×3 matrices, and for an assembly language version of the multiplication by A_{pp}^{-1}, using the trick described in §(6.5.5).

entirely due to the reduction in the time taken for memory loads and stores; the number of clock cycles for a register to register floating point operation is the same in both precisions. Obviously, the improved performance must be weighed against degradation of acceptance rate and less tangible considerations such as reversibility and the accuracy of energy calculations.

The Cray-T3E memory system is equipped with a number of *streams*, which supersede the read-ahead buffers of the Cray-T3D. By enabling streams we gain a further factor of about 2 in speed.[22]

Together these optimisations (assembly language, 32-bit precision and streams) account for an approximately seven-fold increase in speed.

6.5.4 Exponentiating the conjugate momenta

The gauge updates in the equations of motion require the exponential of the conjugate momenta on each link.

$$U_{\mu x}(\tau + d\tau) = \exp\left(id\tau P_{\mu x}(\tau + \frac{1}{2}d\tau)\right) U_{\mu x}(\tau) \tag{6.52}$$

We compute the exponential *exactly*, or rather, up to machine precision. Since the $P_{\mu x}$ are hermitian and traceless, this ensures that the $U_{\mu x}$ will remain unitary and special, again up to machine precision. We have two methods for computing the exponential.

The first method unitarily diagonalises the hermitian matrix $H = P_{\mu x}$,

$$H = Q^\dagger D Q \tag{6.53}$$

using the LAPACK library routine CHEEV, then computes

$$\exp(iHd\tau) = Q^\dagger \exp(iDd\tau)\, Q. \tag{6.54}$$

[22]Spare a thought for those compelled by circumstance to develop codes on earlier versions of the Cray-T3E which were not "stream-safe".

The other method works with the traceless, anti-hermitian matrix $G = iP_{\mu x}d\tau$ and proceeds along the following lines.[23] It follows from the Cayley-Hamilton theorem that any power series in the 3×3 matrix G can be expressed as a polynomial in G of order at most 2, and in particular that

$$\exp(G) = aI + bG + cG^2 \tag{6.55}$$

for some a, b and c which depend on G but *not* on the choice of basis. In a basis in which G is diagonal, $\exp(G)$ may be represented by

$$\begin{pmatrix} e^{i\alpha} & & \\ & e^{i\beta} & \\ & & e^{-i(\alpha+\beta)} \end{pmatrix} =$$

$$aI + b\begin{pmatrix} i\alpha & & \\ & i\beta & \\ & & -i(\alpha+\beta) \end{pmatrix} + c\begin{pmatrix} i\alpha & & \\ & i\beta & \\ & & -i(\alpha+\beta) \end{pmatrix}^2 \tag{6.56}$$

Here $i\alpha$, $i\beta$ and $-i(\alpha+\beta)$ are the pure imaginary eigenvalues of G, which we determine by solving the characteristic equation[24] using Cardano's formula and some trigonometry. Equation (6.56) can be re-expressed as a linear system with Vandermonde's matrix as the coefficient matrix

$$\begin{pmatrix} 1 & i\alpha & (i\alpha)^2 \\ 1 & i\beta & (i\beta)^2 \\ 1 & -i(\alpha+\beta) & (-i(\alpha+\beta))^2 \end{pmatrix}\begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} e^{i\alpha} \\ e^{i\beta} \\ e^{-i(\alpha+\beta)} \end{pmatrix} \tag{6.57}$$

which may be solved for a, b, and c. Special care must be taken if G has degenerate eigenvalues.

The two methods give results which do not differ significantly, but the second method, being some six times faster than the first, is the one we use in

[23]We are grateful to A. D. Kennedy [64] for providing details of this method.

[24]Computing the characteristic equation is simplified by expressing $\det(\lambda I - G)$ in terms of traces of G^2 and G^3.

production.[25]

6.5.5 The clover term

The operations of multiplying by the clover term A_{pp} and its inverse A_{pp}^{-1} can be simplified quite significantly by exploiting the block structure of the clover term itself.[26] Details are given in appendix C. The trick reduces the floating point operations count to construct the clover term by a factor of about 4, and the operations count to multiply by the clover term or its inverse by a factor slightly larger than 2. The memory requirements to store the clover term are halved.

6.5.6 The energy calculation

Since the energy $\mathcal{H}$ is an extensive quantity, constrained by the conservative dynamics to change in small part over the course of a trajectory, the calculation of $\Delta\mathcal{H}$ will become more and more problematic as the lattice volume is increased.

We calculate the energy on a site-by-site basis at the beginning and end of each trajectory, finally computing $\Delta\mathcal{H}$ by summing the site-by-site differences. This unnecessary refinement was originally intended to help reduce round-off error, but any benefits are entirely negligible due to the fact that global summations are performed in 64-bit precision. We note that on $V = 12^3 \times 24$ this method yields a $\Delta\mathcal{H}$ that does not differ significantly from the naïve result.

The pure gauge contribution S_{pg} to the energy is proportional to the average plaquette, and it can be calculated in similar fashion (see eg. [23]). Using properties of the Gell-Mann matrices, the kinetic term

$$\frac{1}{2}\mathrm{Tr}\; P_\mu^2 = \sum_{x,\mu}\sum_{a=1}^{8}\left(P_{\mu x}^a\right)^2 \tag{6.58}$$

[25]To be fair, one could hope to improve the performance of the first method by writing a diagonalisation routine optimised for hermitian, traceless, 3×3 matrices. I did not explore this possibility because, once the second method had been introduced, the fraction of run-time spent in exponentiating the conjugate momenta was so small that further attempts at optimisation would be unrewarding.

[26]We were made aware of this possibility by reference [65].

is also straightforward.

The fermionic contribution has two parts

$$S_{fe} = \phi^\dagger \left(\hat{M}^\dagger_{pp}\hat{M}_{pp}\right)^{-1} \phi - 2 \ln \det A_{\bar{p}\bar{p}}. \tag{6.59}$$

The first term can be computed by either

$$\phi^\dagger \left(\hat{M}^\dagger_{pp}\hat{M}_{pp}\right)^{-1} \phi = \phi^\dagger X_p \tag{6.60}$$

or

$$\phi^\dagger \left(\hat{M}^\dagger_{pp}\hat{M}_{pp}\right)^{-1} \phi = \phi^\dagger \hat{M}^{-1}_{pp}\hat{M}^{-\dagger}_{pp}\phi = Y_p{}^\dagger Y_p. \tag{6.61}$$

We use the second method (equation (6.61)) in practice.

The term $\ln \det A_{\bar{p}\bar{p}}$ is easily calculated by exploiting the $L^\dagger DL$ factorisation of $A_{\bar{p}\bar{p}}$ which is already calculated for use in the routine implementing multiplication by $A^{-1}_{\bar{p}\bar{p}}$. Since L is lower triangular with ones down the diagonal, we have

$$\det A_{\bar{p}\bar{p}} = \det \left(L^\dagger DL\right) = \det D, \tag{6.62}$$

and the determinant reduces to the product of the diagonal components, each of which can be shown to be real and positive.[27] When the inverse of the clover term is computed by the method described in appendix (C), the above calculation must be modified, but this is easily done. The remaining subtlety is the need to avoid underflow and overflow when taking the product of the diagonal components. We do this by rescaling the running product whenever it becomes dangerously small or dangerously large. On a parallel architecture, the need for a global product routine is obviated by first taking the logarithm of the determinant on the local lattice, then completing the calculation using existing routines for the global sum.

[27]Provided only that $A_{\bar{p}\bar{p}}$ is hermitian and positive definite.

6.5.7 The first solve in a trajectory

Recall that the pseudofermions ϕ are refreshed at the start of each trajectory from

$$\phi = \hat{M}_{pp}^{\dagger}\eta \tag{6.63}$$

for Gaussian noise η, and that, by equations (6.40) and (6.41), Y_p is given by

$$Y_p = \hat{M}_{pp}X = \hat{M}_{pp}^{-\dagger}\phi. \tag{6.64}$$

Thus, immediately after the refresh (before the gauge fields on which M depends have changed) we have

$$Y_p = \eta \tag{6.65}$$

and consequently there is no need to compute Y_p. If a two-step solve is being used, the very first solve for Y_p in each trajectory can be avoided. Moreover, avoiding this solve also avoids introducing an error into Y_p due to imperfect convergence.

A further consequence of this observation is that, in view of equation (6.61), the fermionic contribution to the initial energy is computable up to machine precision without the need for any solve.

As the accuracy of GHMC depends crucially on the accuracy of the calculation of the change in energy over a trajectory, computing the energy in this way seems highly desirable, and our code was designed to exploit this trick.

There is one potential drawback to the trick of skipping the initial solve for Y: it is manifestly not reversible – it is impossible to do the final solve for Y on the reverse trajectory exactly. It is worth noting that the trick will not affect measurements of $\delta\Delta U$, as the final gauge field update in the leapfrog integration scheme is performed before the final solve. It will, however, affect measurements of $\delta\Delta P$ (and hence $\delta\Delta\mathcal{H}$). The deleterious effect of the trick on reversibility can be made arbitrarily small by tightening the convergence criteria on the exterior solves. Although no significant reduction in reversibility was observed on $12^3 \times 24$ lattices when this trick was introduced, the possible impact on reversibility should not be forgotten.

6.5.8 Relaxing solver convergence criteria

Although the accuracy of the exterior (ie. first and last) solves in each trajectory is important inasmuch as it determines the accuracy of the calculation of $\Delta\mathcal{H}$ (and hence the correctness of a GHMC simulation), the accuracy of the interior solves is rather less important. It is even permissible to conduct the molecular dynamics using a guidance Hamiltonian $\mathcal{H}'$ which differs from the one used in the Metropolis step, but we did not explore this possibility.[28]

Provided that a reversible guessing strategy is adopted, the interior solves can be performed using looser convergence criteria than the exterior ones.[29] We will then be trading acceptance rate for solver iterations. Studies conducted on small volumes [4] found that quite promising savings in the work effort per accepted configuration could be had by this means.

As the lattice volume is increased and as the sea-quark mass is reduced, it becomes more difficult to control departures of the molecular dynamics from the $\Delta\mathcal{H} = 0$ surface, and we have essentially only three run-time options that we can tune to maintain a reasonable acceptance rate: $d\tau$, n_G and the convergence criteria for the internal solves. Now, whichever precision (32-bit or 64-bit) we are using for the fields U, ϕ, X, Y, will determine the tightest convergence criteria that we can use for the external solves. When 32-bit precision is used at lighter masses on larger lattices, it will often be the case that the convergence criteria used in the internal solves is the same as that used in the external solves.

From this it would be wrong to conclude that relaxing solver convergence criteria is not useful at light masses and large volumes. My view is that the problem is not so much being unable to relax the internal criteria with respect to the

[28]In principle, it is possible that a guidance Hamiltonian $\mathcal{H}' = \mathcal{H}(\beta', \kappa', C'_{SW})$ could yield a higher acceptance rate than $\mathcal{H}(\beta, \kappa, C_{SW})$. However, β', κ' and C'_{SW} must be tuned and this will not be cheap.

[29]This strategy was championed in [66] as a theoretically sound alternative to chronological inversion methods.

external, but rather being unable to tighten the external criteria with respect to the internal. The problem arises out of the fact that on any given architecture there is a limited choice of precision in which to work, and it is exacerbated by the fact that our codes make this choice at compile-time. The remedy that I propose is to use 32-bit precision for the internal solves (thereby benefiting from reduced memory accesses on the Cray-T3E), and 64-bit precision everywhere else. This would necessitate a major overhaul of existing codes,[30] but I recommend it highly nonetheless.

6.6 Computing observables "on the fly".

In a HMC simulation, gauge configurations will generally be stored less frequently than they are generated. This is a sensible strategy. Configurations separated by a single trajectory are usually highly correlated, and hadronic observables are typically calculated on configurations separated by intervals of order the (relevant) auto-correlation time.

However, in order to estimate the autocorrelation times, one needs observables calculated more frequently. We are interested, therefore, in observables which can be calculated cheaply on each "physical" gauge configuration. By cheaply I mean that the cost is significantly less than the cost of the trajectory, and by "physical", I mean those configurations which are accepted into the sample, ie. the starting configurations in each GHMC trajectory.

Observables obtainable as by-products of computations which must be done anyway are therefore desirable. These include

1. The 1×1 Wilson loop, or plaquette, which is proportional to the pure gauge part of the action. We do not, at present, have parallel codes to compute larger Wilson loops.

[30]Two versions, one for each precision, of each of a large number of modules would need to be invokable at run-time. Is there a way to achieve this without duplicating the source of each module? This consideration is especially important in an environment where all change-control mechanisms are anarchic.

2. The number of iterations of the solver.
3. The chiral condensate $\langle\bar{\psi}\psi\rangle$ is not itself cheap to compute, but a noisy estimator can be obtained as a by-product of the first solve in a GHMC trajectory.

Other quantities that we log at the end of each trajectory are the initial and final total energies, the kinetic, fermionic and pure gauge contributions thereto, and the result of the Metropolis decision. Such information enables us to monitor $\langle\Delta\mathcal{H}\rangle$, $\langle\exp(-\Delta\mathcal{H})\rangle$ and acceptance rate, and is useful in deciding whether a simulation has equilibrated, and whether it is behaving as expected.

6.6.1 The chiral condensate

The chiral condensate $\langle\bar{\psi}\psi\rangle$ is proportional to the trace of the inverse of the fermion matrix.

It is useful to relate $\mathrm{Tr}M^{-1}$ to the red-black preconditioned matrix. Formally, we have

$$\begin{aligned}
&\mathrm{Tr}\ M^{-1}\\
=\ &\mathrm{Tr}\left[(A-\kappa D)^{-1}\right]\\
=\ &\mathrm{Tr}\left[\left(I-\kappa A^{-1}D\right)^{-1}A^{-1}\right]\\
=\ &\mathrm{Tr}\left[\left(I+\left(\kappa A^{-1}D\right)+\left(\kappa A^{-1}D\right)^2+\left(\kappa A^{-1}D\right)^3+\ldots\right)A^{-1}\right] && (6.66)\\
=\ &\mathrm{Tr}\left[\left(I+\left(\kappa A^{-1}D\right)^2+\left(\kappa A^{-1}D\right)^4+\ldots\right)A^{-1}\right] && (6.67)\\
=\ &\mathrm{Tr}\left[\left(I-\left(\kappa A^{-1}D\right)^2\right)^{-1}A^{-1}\right] && (6.68)\\
=\ &\mathrm{Tr}\left[\left(A-\kappa^2 DA^{-1}D\right)^{-1}\right] && (6.69)
\end{aligned}$$

The Neumann series expansions in equations (6.66) and (6.68) converge if the spectral radius of $\kappa A^{-1}D$ is less than one. Equation (6.67) relies on the fact that the matrices

$$\left(A^{-1}D\right)^{2m+1} A^{-1}, \qquad m = 0, 1, 2, \ldots$$

are block off-diagonal and hence do not contribute to the trace. It remains to recognise that the matrix in equation (6.69)

$$A - \kappa^2 D A^{-1} D = \begin{pmatrix} \hat{M}_{ee} & 0 \\ 0 & \hat{M}_{oo} \end{pmatrix} \equiv \hat{M} \tag{6.70}$$

is nothing other than (both parities of) the red-black preconditioned fermion matrix. We have, therefore, that

$$\mathrm{Tr}\ M^{-1} = \mathrm{Tr}\ \hat{M}_{ee}^{-1} + \mathrm{Tr}\ \hat{M}_{oo}^{-1} \tag{6.71}$$

and, taking the ensemble average over gauge configurations,

$$\langle \mathrm{Tr}\ \hat{M}^{-1} \rangle = 2 \langle \mathrm{Tr}\ \hat{M}_{ee}^{-1} \rangle = 2 \langle \mathrm{Tr}\ \hat{M}_{oo}^{-1} \rangle, \tag{6.72}$$

since observables must be independent of the choice of parity.

A noisy estimate of $\mathrm{Tr}\, \hat{M}_{pp}^{-1}$ can be obtained in the following manner. If $\xi_1, \ldots, \xi_{N_E}$ are noise vectors with the property

$$\lim_{N_E \to \infty} \frac{1}{N_E} \sum_{n=1}^{N_E} (\xi_n)_i (\xi_n)_j^* = \delta_{ij} \tag{6.73}$$

then

$$\mathrm{Tr}\ \hat{M}_{pp}^{-1} = \lim_{N_E \to \infty} \frac{1}{N_E} \sum_{n=1}^{N_E} \xi_n^\dagger \hat{M}_{pp}^{-1} \xi_n. \tag{6.74}$$

We can now estimate $\langle \mathrm{Tr} \hat{M}_{pp}^{-1} \rangle$ by the sample mean over gauge configurations $U^{(1)}, \ldots, U^{(N_U)}$ produced in a GHMC simulation:

$$\langle \mathrm{Tr} \hat{M}_{pp}^{-1} \rangle \approx \frac{1}{N_U} \frac{1}{N_E} \sum_{m=1}^{N_U} \sum_{n=1}^{N_E} \left(\xi_n^{(m)}\right)^\dagger \hat{M}_{pp}^{-1} \left[U^{(m)}\right] \xi_n^{(m)}. \tag{6.75}$$

Ideally the number of estimators N_E on each gauge configuration should be large. However we can hope, appealing to the ergodicity of GHMC, that equation (6.75) will still provide a useful, albeit rather noisy, estimate even for $N_E = 1$.

The vectors η, used to refresh pseudofermions in GHMC, have components drawn at random from a Gaussian distribution of zero mean and unit variance, and hence possess property (6.73).[31] Furthermore, at the start of each trajectory, $Y_p = \eta$ and the gauge configuration is "physical", and so

$$\eta^\dagger \hat{M}_{pp}^{-1} \eta = \eta^\dagger \hat{M}_{pp}^{-1} Y_p = \eta^\dagger X_p. \tag{6.76}$$

In this way, we get one estimate almost for free, the only additional cost being a single inner product.

The purist will notice that this method is not strictly valid in view of the subtle correlations that exist between the noise vectors $\eta^{(m)}$ and the gauge configurations $U^{(m+1)}$. It is not known whether the bias that these correlations may cause is significant.

[31]Note that if an accurate estimate of $\mathrm{Tr}\hat{M}^{-1}$ on individual gauge field configurations is required, it is preferable to take several estimators using Z_2 noise [67] instead of Gaussian noise (see eg. [68]).

Chapter 7

GHMC: Pilot simulations

This chapter describes a series of pilot simulations using the GHMC code described in chapter 6. These were amongst the first simulations using two degenerate flavours of dynamical clover-improved Wilson fermions.

7.1 Run parameters

At the outset it was appreciated that simulations involving dynamical fermions necessarily require several orders of magnitude more computing time than quenched simulations. However, this leaves much room for uncertainty as to the computing time required per independent configuration at some β, κ and lattice volume.[1] Our aim then was to begin to map out the $\{\beta, \kappa, V\}$-parameter space for dynamical clover-improved Wilson fermions in preparation for later, more ambitious simulations. At the same time it was not intended that this be a throw-away calculation: the lattice volume in physical units would need to be sufficiently large for meaningful hadron spectroscopy. The choices of β and V were therefore informed by

- the experience of the UKQCD collaboration with clover-improved fermions in quenched simulations,
- previous dynamical simulations with unimproved Wilson fermions by other collaborations, particularly SESAM,

[1]In this chapter, the notation $\kappa = \kappa_{\text{sea}}$ shall always characterise a sea-quark mass in a GHMC simulation.

- trial simulations on 4^4 lattices and conservative estimates of scaling behaviour,
- estimates of the speed that our code would sustain on the new Cray-T3E at Edinburgh,[2] and
- political pressure to be delivering preliminary results on timescales of 4-6 months.

Taken together, these considerations implied that the lattice would of necessity be rather coarse. This was felt by some to be a good thing, as it is on coarse lattices that $\mathcal{O}(a)$-improvement should be seen to best advantage.

The clover coefficient C_{SW} was fixed in each case by the choice of β and the prescription of non-perturbative improvement. We remain indebted to the ALPHA collaboration for making available to us preliminary estimates of C_{SW} at $N_f = 2$.[3]

During the first weeks of production, there was some vacillation over the choice of β, but eventually most effort was given over to $\beta = 5.2$. Simulations at $\beta = 5.2$, $\kappa = 0.136, \ldots, 0.1395$, $V = 12^3 \times 24$ were followed by smaller volume runs ($V = 8^3 \times 24$) to enable finite-size effects to be studied. The $\beta = 5.2$ runs at larger volume $V = 16^3 \times 24$ came later, as did the lightest mass run at $\beta = 5.2$, $\kappa = 0.1398$, $V = 12^3 \times 24$. Rather smaller statistics were obtained at $\beta = 5.3$ and $\beta = 5.4$; for the most part these runs were scheduled to take advantage of computing capacity made spare by artificial constraints due to the job-queuing policy on the Cray-T3E.

Table 7.1 lists the parameters used in these simulations. The 'burned' column gives the trajectory numbers which were discarded for equilibration. 'pcsn' gives

[2]This was complicated by the fact that the delivery schedule was altered so that only a 16-node interim machine was available during the first few months.

[3]Actually, it is at the very range of β that we chose to explore ($\beta \leq 5.4$) that C_{SW} is most difficult to determine precisely. The estimate of $C_{SW} = 1.76$ at $\beta = 5.2$, for example, has since been revised upwards to 2.02.

the precision (32-bit or 64-bit), n_G was defined by equation (6.38), 'pre' indicates whether the block Jacobi preconditioning of §(6.4) was employed, and 'acc' gives the percentage acceptance rate.

In all cases the trajectory length was held fixed $N_{\mathrm{md}} d\tau = 1$. The GHMC mixing angle was also held fixed, $\theta = \pi/2$, so that it is standard Hybrid Monte Carlo that is being employed. Although it was recognised that tuning these parameters could potentially improve autocorrelation times, it was thought likely that the cost of the tuning exercise itself would outweigh any benefits that might ensue.

β	C_{SW}	$V/24$	κ_{sea}	Burned	Analysed	pcsn	N_{md}	n_G	pre	acc
5.2	1.76	8^3	.1360	1549–1649	1650–3000	32	50	1	y	79.3
5.2	1.76	8^3	.1370	3001–3099	3100–5004	32	50	1	y	79.0
5.2	1.76	8^3	.1380	5005–5599	5600–7000	32	50	1	y	76.8
5.2	1.76	8^3	.1390	7001–7699	7700–10056	32	50	1	y	70.3
5.2	1.76	8^3	.1395	10057–10149	10150–21150	32	50	2	y	68.3
5.2	1.76	12^3	.1360	1530–2210	2211–5190	64	50	2	n	77.1
5.2	1.76	12^3	.1370	1601–1999	2000–8007	64	50	2	n	73.1
5.2	1.76	12^3	.1380	1601–1999	2000–8001	64	50	2	n	71.5
5.2	1.76	12^3	.1390	2001–2399	2400–8019	64	50	2	n	61.1
5.2	1.76	12^3	.1395	5861–6299	6200–11305	64	64	2	n	65.9
5.2	1.76	12^3	.1398	10001–10609	10610–15118	64	100	2	y	81.4
5.2	1.76	16^3	.1390	1001–1309	1310–2924	32	100	2	y	83.4
5.2	1.76	16^3	.1395	2001–2309	2310–5503	32	100	2	y	77.5
5.3	1.72	12^3	.1340	1859–2349	2350–3000	32	50	1	y	59.0
5.3	1.72	12^3	.1350	3001–3399	3400–5000	32	64	1	y	77.3
5.3	1.72	12^3	.1360	4987–5499	5500–7205	32	64	1	y	75.6
5.3	1.72	12^3	.1370	7183–7299	7300–9228	32	64	2	y	81.8
5.3	1.72	12^3	.1380	9229–9499	9500–11638	32	64	2	y	75.4
5.4	1.69	12^3	.1320	601–899	900–1182	64	50	1	n	56.8
5.4	1.69	12^3	.1330	720–1199	1200–2378	64	64	1	n	78.1
5.4	1.69	12^3	.1340	901–1499	1500–2921	64	50	1	n	65.5
5.4	1.69	12^3	.1350	1421–1999	2000–4432	64	50	2	y	77.5
5.4	1.69	12^3	.1360	1381–1999	2000–4211	64	50	2	y	73.7
5.4	1.69	12^3	.1370	4212–5099	5100–8025	64	64	2	y	79.5

Table 7.1: GHMC run parameters. All runs used pure HMC, BiCGSTAB with a target residue of 10^{-7}, and $d\tau = 1/N_{\mathrm{md}}$.

7.2 Equilibration and the black arts

How do you decide whether a simulation has thermalised? What parameters do you vary and by how much in order to achieve a reasonable acceptance rate? These are questions which have no entirely satisfactory answers – there is much "black art" involved in running a GHMC simulation.

Here I try to convey something of the strategies that we followed and refined during these pilot simulations.

At each β and V, our usual approach was to start from a random configuration of the gauge fields, as this generally leads to much faster convergence than starting from the unit gauge configuration. The initial value of κ is chosen to be well below the range in which the critical value is suspected to lie. The initial value of the timestep is taken to be rather large, and the initial convergence criteria for the internal solves are loose. One then accumulates statistics until all gross trends in the monitored observables appear to have died away. One now makes a small increase in κ, and resumes from the last stored configuration. The process of incrementing κ and accumulating statistics is iterated until the sea-quark mass is light enough to be considered "interesting"; in the absence of any better criterion, the longer the solver takes to converge, the more interesting is the sea-quark mass. The purpose so far has been solely to obtain a gauge configuration near to the equilibrium distribution of the heaviest interesting quark mass, and hitherto one can have followed a fairly aggressive policy, not worrying overmuch about acceptance rate and full equilibration (thermalisation). But now it is time to be more careful.

Based on experience gained at the previous κ, it may be necessary to adjust one or more of $d\tau$, n_G and the solver's convergence criteria in order to achieve a satisfactory acceptance rate at the new κ. It is best to make any such adjustments as early as possible, since in principle one should allow for re-equilibration whenever any simulation parameters are altered. We generally aimed for an acceptance rate in the range of 70–80%, and with practice, we got better at getting this right without repeated parameter tweaking.

The question of equilibration is now an important, and delicate, one. Unfortunately, given the cost of simulations with dynamical fermions, any reliable diagnostic of equilibration is likely to require a level of statistics comparable to, or greater than, the level on which it is ultimately intended to base the analysis of physical observables. The policy that we followed was that

1. there should be no evidence of any trend (or cyclicity) remaining in monitored observables,
2. a further number of trajectories at least equal to several times our best guess at the worst case exponential autocorrelation time §(7.4) have been discarded (or 'burned'), and
3. the average of $\exp(-\Delta\mathcal{H})$ over the remaining trajectories should be consistent with one.

There are various potential traps here; for example, one could easily deceive oneself with respect to the last criterion by varying the window over which $\langle\exp(-\Delta\mathcal{H})\rangle$ is estimated.

By the time that one has determined the start of the range of trajectories which is deemed safe for analysis, one will already have accumulated a modest level of statistics. This may well be sufficient to provide a preliminary estimate of lattice spacing (which runs with κ), quark mass or other quantity. On the basis of this information, one can decide whether to move on to a lighter mass, or to accumulate further statistics at the current one.

Near the critical value of the sea-quark mass, we expect that the number of iterations N_{solver} that the solver takes to converge should obey a critical scaling law, and we make the *ansatz*

$$N_{\text{solver}} \propto \left(\frac{1}{\kappa} - \frac{1}{\kappa_c}\right)^{\delta}. \tag{7.1}$$

Once modest statistics have been accumulated at three values of κ, equation (7.1) may be used to obtain a preliminary estimate of κ_c, which itself can inform the choice of the next κ. The estimate can be refined with each new κ and with increased statistics. The final value, however, should be determined from an analysis of the hadron spectrum.

Using Conjugate Gradient at $\beta = 5.2$, $C_{SW} = 1.72$, $V = 12^3 \times 24$, the measured value of the exponent in equation (7.1) is $\delta = -0.612(2)$ [4, §4.2.3]. I am not aware of any simple model from which δ can be predicted.

7.3 Error Estimation

The problem with estimating the errors in observable quantities in this kind of Markov process is that the configurations from which they are determined will be correlated with each other, so the naïve estimate will in general underestimate the true error. The problem of estimating the error in an observable is closely related to the problem of determining the integrated autocorrelation time §(7.4) for that observable.

The standard practice in hadron spectroscopy is to compute observables on configurations which, it is hoped, can safely be considered independent. The usual criterion is that the configurations are separated by at least twice the typical integrated autocorrelation time for hadronic observables.

However, it is hard to believe that discarding the intermediate measurements could possibly improve the estimate of the central value, nor give a more reliable estimate of the error. Moreover, as will be seen, measuring the autocorrelation time is itself problematic.

Here we need to estimate observables from measurements taken at the end of every trajectory, and these will certainly not be independent.

There is no one "correct" method. The method I use here is inspired by the bootstrap[69]. Given an ordered set of k measurements (eg. of the average plaquette measured after the accept/reject step in a HMC trajectory) $x_1, \ldots, x_n$

we group the measurements into sub-sequences (or *bins*) of size b

$$\{x_1,\ldots,x_b\},\{x_{b+1},\ldots,x_{2b}\},\ldots,\{x_{mb+1},\ldots,x_n\}$$

so that

$$n = mb + r. \tag{7.2}$$

If $r = 0$, there are m bins each of size b, otherwise there are $m + 1$ bins, and the last bin has size $r < b$. Now take the average $\bar{x}_i$ of the values in the ith bin, and apply a bootstrap resampling procedure [69] to the data set $\{\bar{x}_1, \bar{x}_2, \ldots\}$, determining the 68% (say) confidence limits. To reduce any bias towards the last few measurements in the original data set, we choose the ith bin with probability equal to the size of the bin divided by n.

The procedure is repeated for $b = 1, 2, \ldots, b_{max}$ where hopefully b_{max} is large compared to the integrated autocorrelation time and small compared to the number of samples n. If these two conditions can be met simultaneously, we expect to see plateaux in the confidence limits once the bin size is large enough that the $\{\bar{x}_i\}$ can be considered independent. The upper and lower limits on the error bars are then read off from the ordinates of the plateaux.

The method is rather expensive (one needs to sort the resampled data sets at each bin size), but it is able to provide asymmetric error bars and is independent of any estimate of the autocorrelation time. Indeed, it can be used as a rough consistency check on a direct measurement of the autocorrelation time.

In figure 7.2 confidence limits calculated by this method are plotted against bin-size for the average plaquette (which is strongly correlated) and the initial kinetic energy at the start of each GHMC trajectory (which is completely uncorrelated, being a sum of normal deviates). The quality of the plateaux obtained by this method can be deceptive — contrast figure 7.3 in which the same information is displayed using logarithmic scales for the abscissae — so these error estimates should be viewed with some caution.

Table 7.2 shows the average plaquette, $\mathrm{Tr}\hat{M}_{pp}^{-1}/V$ and the number of BiCGSTAB iterations required to solve $\hat{M}_{pp}^{\dagger}Y_p = \phi$ with a target residue of 10^{-7}, for each of the simulations in table 7.1. In all cases, the error bars are consistent with being symmetric about the central value. Figures 7.4–7.6 display the same information in graphical form.

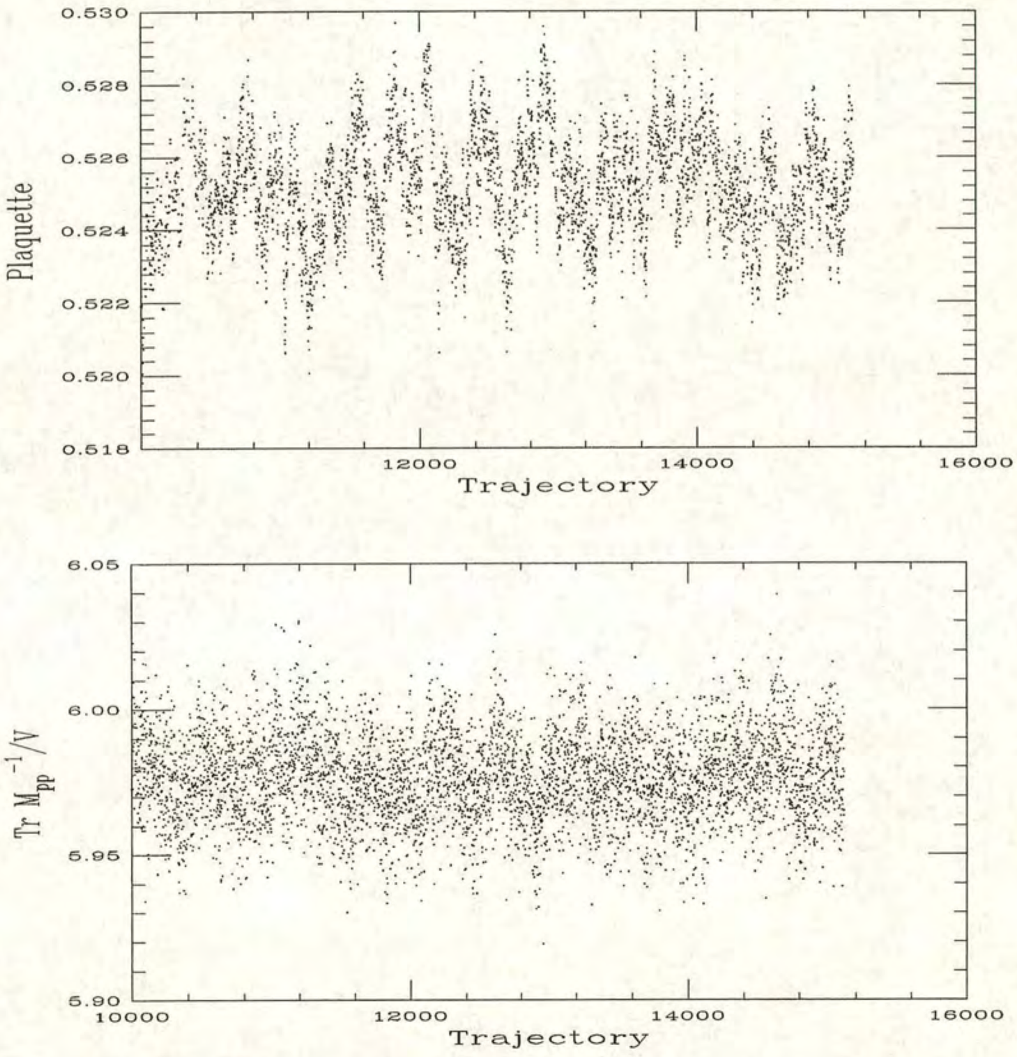


Figure 7.1: Sample HMC time series for plaquette (top) and $\mathrm{Tr}\hat{M}_{pp}^{-1}/V$ (bottom). Parameter values are $\beta = 5.2$, $C_{SW} = 1.76$, $\kappa = 0.1398$, $V = 12^3 \times 24$.

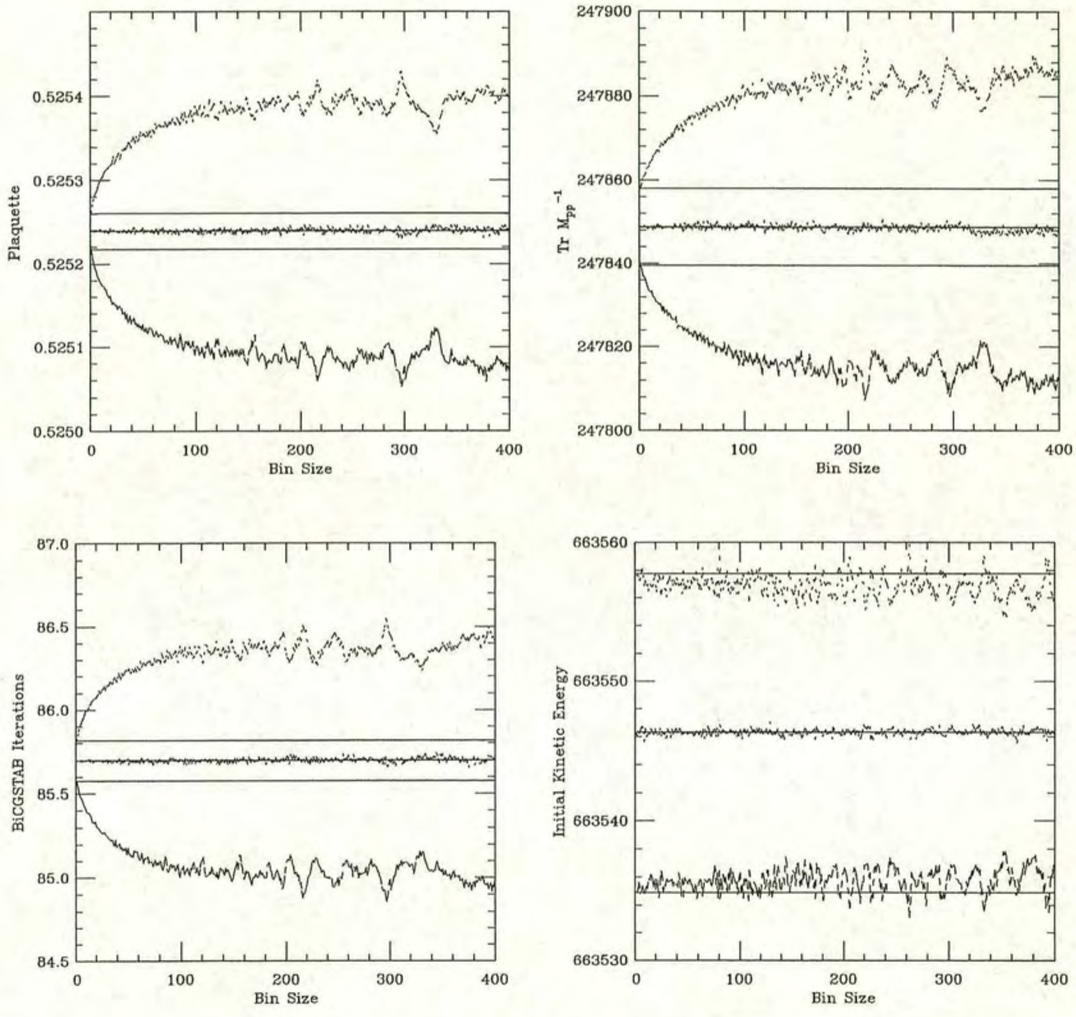


Figure 7.2: Upper and lower 68% confidence limits on the average plaquette (top left), $\mathrm{Tr}\hat{M}_{pp}^{-1}$ (top right), N_{BiCGSTAB} (bottom left) and initial kinetic energy (bottom right) as a function of bin-size. The central broken curve is the bootstrap median. The central solid line is the mean; the other solid lines show the naïve errors. Parameter values are $\beta = 5.2$, $C_{SW} = 1.76$, $\kappa = 0.1398$, $V = 12^3 \times 24$.

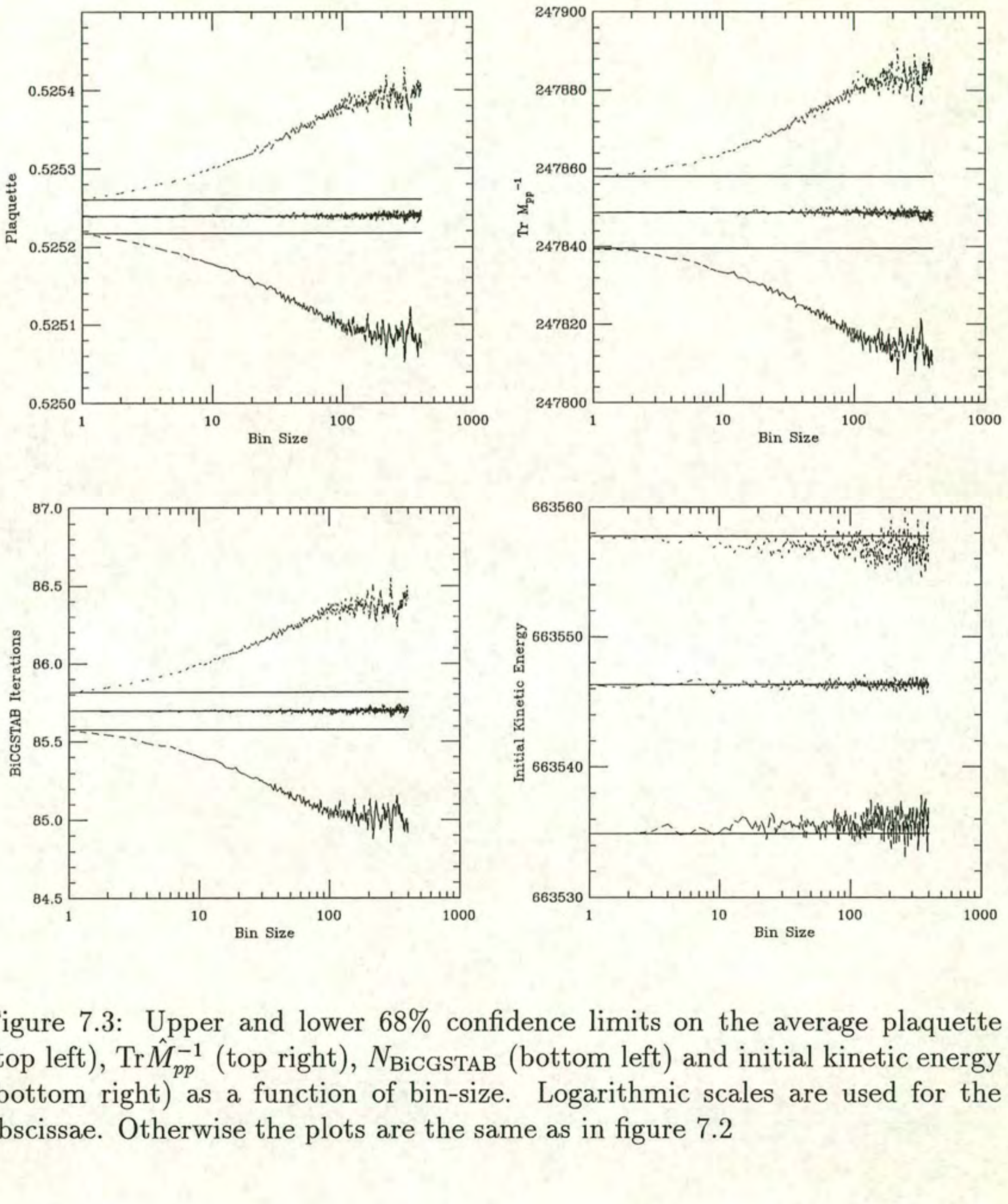


Figure 7.3: Upper and lower 68% confidence limits on the average plaquette (top left), $\mathrm{Tr}\hat{M}_{pp}^{-1}$ (top right), N_{BiCGSTAB} (bottom left) and initial kinetic energy (bottom right) as a function of bin-size. Logarithmic scales are used for the abscissae. Otherwise the plots are the same as in figure 7.2

β	C_{SW}	V	κ_{sea}	Plaquette	$\text{Tr}\hat{M}_{pp}^{-1}/V$	N_{BiCGSTAB}
5.2	1.76	$8^3 \times 24$	0.1360	0.4879(5)	6.1438(17)	23.1(1)
5.2	1.76	$12^3 \times 24$	0.1360	0.48744(16)		
5.2	1.76	$8^3 \times 24$	0.1370	0.4933(5)	6.1242(16)	26.41(15)
5.2	1.76	$12^3 \times 24$	0.1370	0.49461(12)		
5.2	1.76	$8^3 \times 24$	0.1380	0.5040(6)	6.0857(23)	33.5(3)
5.2	1.76	$12^3 \times 24$	0.1380	0.50390(15)		
5.2	1.76	$8^3 \times 24$	0.1390	0.5153(6)	6.0324(23)	48.8(7)
5.2	1.76	$12^3 \times 24$	0.1390	0.51558(18)		
5.2	1.76	$16^3 \times 24$	0.1390	0.51620(20)	6.0296(8)	49.75(30)
5.2	1.76	$8^3 \times 24$	0.1395	0.5231(3)	5.9897(15)	68.4(8)
5.2	1.76	$12^3 \times 24$	0.1395	0.52182(19)	5.9972(9)	76.2(1.2)(a)
5.2	1.76	$16^3 \times 24$	0.1395	0.52201(9)	5.9964(5)	67.8(3)
5.2	1.76	$12^3 \times 24$	0.1398	0.52524(15)	5.9762(8)	85.7(7)
5.3	1.72	$12^3 \times 24$	0.1340	0.52272(30)	6.0690(12)	25.4(1.2)
5.3	1.72	$12^3 \times 24$	0.1350	0.52842(20)	6.0428(7)	30.11(9)
5.3	1.72	$12^3 \times 24$	0.1360	0.53311(16)	6.0172(7)	36.81(12)
5.3	1.72	$12^3 \times 24$	0.1370	0.53787(15)	5.9867(6)	48.52(21)
5.3	1.72	$12^3 \times 24$	0.1380	0.54166(13)	5.9554(7)	71.4(5)
5.4	1.69	$12^3 \times 24$	0.1320	0.54489(20)		
5.4	1.69	$12^3 \times 24$	0.1330	0.54746(12)	6.0168(4)	
5.4	1.69	$12^3 \times 24$	0.1340	0.54979(9)	6.0000(5)	
5.4	1.69	$12^3 \times 24$	0.1350	0.55219(8)	5.9783(4)	40.31(7)
5.4	1.69	$12^3 \times 24$	0.1360	0.55448(8)	5.9551(5)	51.82(13)
5.4	1.69	$12^3 \times 24$	0.1370	0.55664(8)	5.9300(6)	73.65(23)

Table 7.2:
The average plaquette, $\text{Tr}\hat{M}_{pp}^{-1}/V$ and N_{BiCGSTAB} for various β, κ_{sea} and lattice size.
(a) not preconditioned.

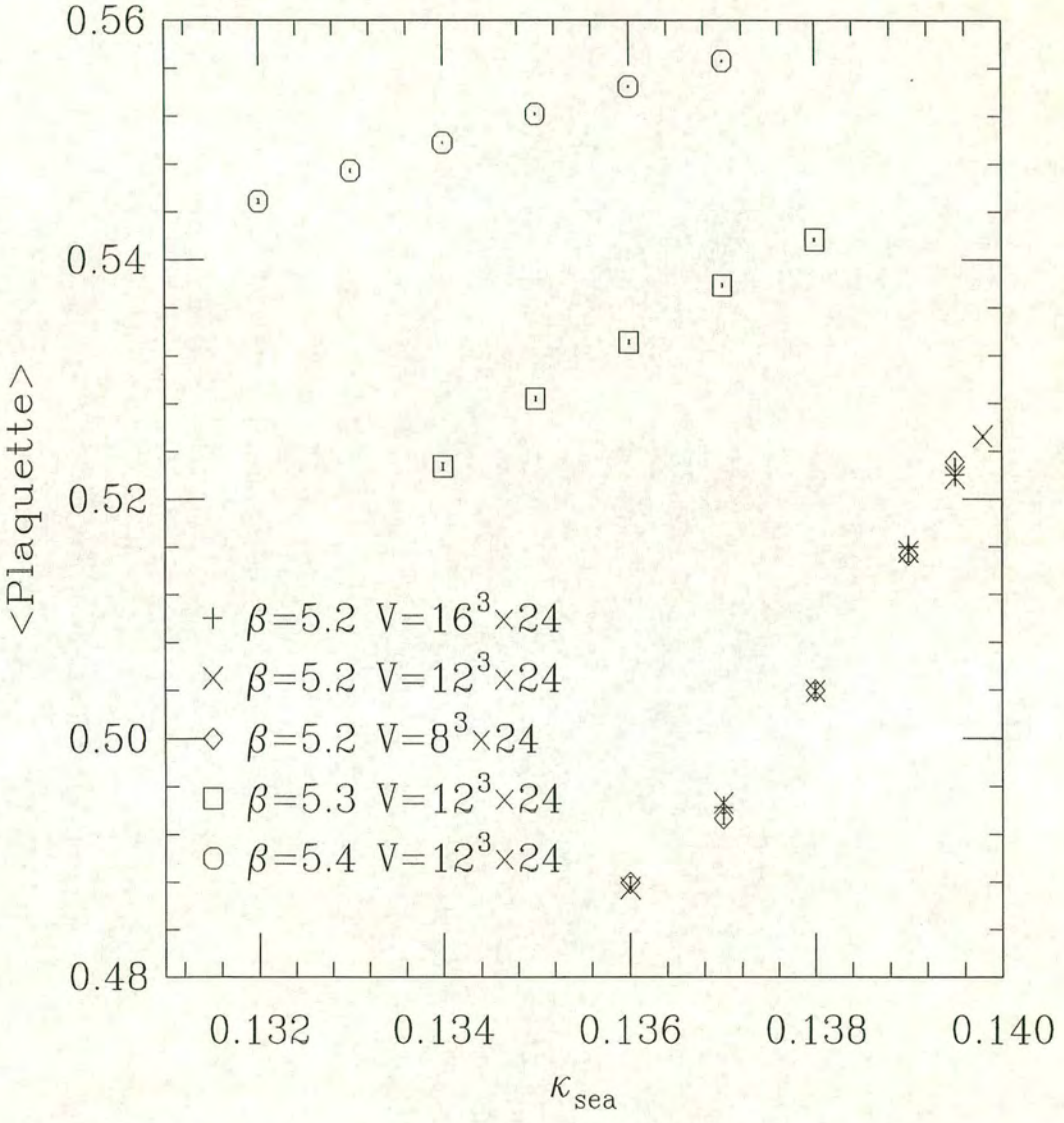


Figure 7.4: The average plaquette as a function of κ_{sea}. Error bars, indicating the 68% confidence levels, are smaller than the plotting symbols.

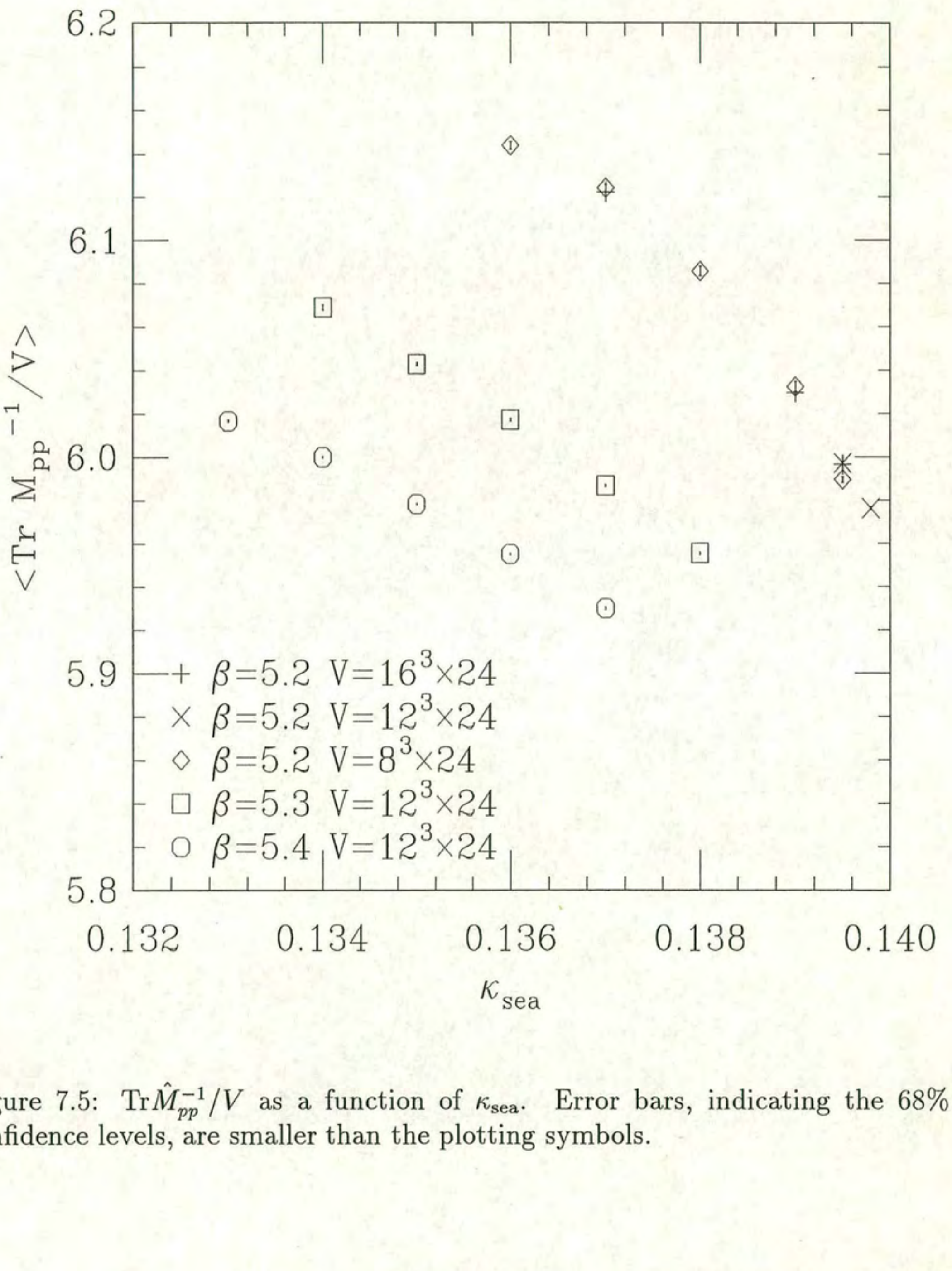


Figure 7.5: $\mathrm{Tr}\hat{M}_{pp}^{-1}/V$ as a function of $\kappa_{\rm sea}$. Error bars, indicating the 68% confidence levels, are smaller than the plotting symbols.

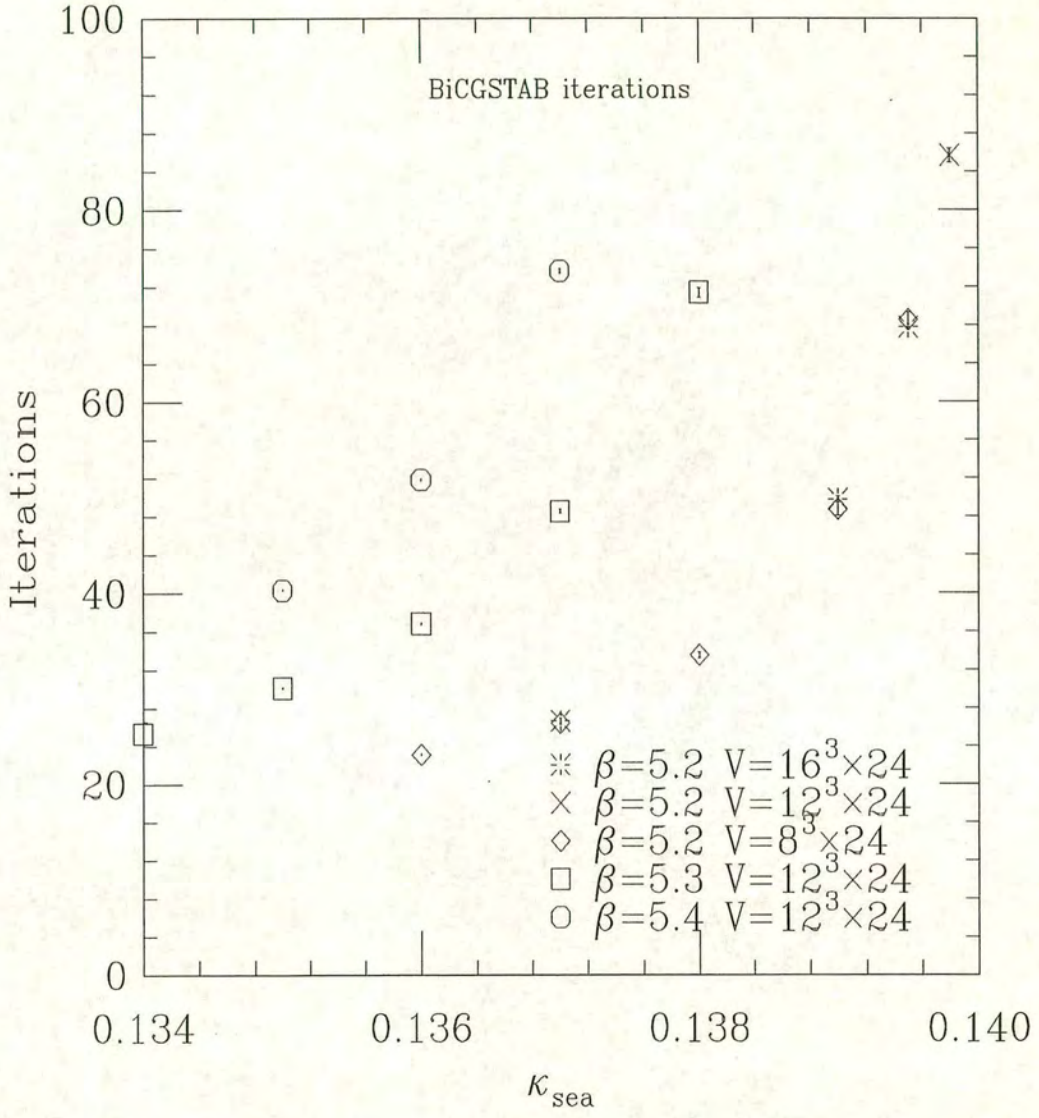


Figure 7.6: The number of BiCGSTAB iterations required to solve $\hat{M}^{\dagger}_{pp}Y_p = \phi$ to a target residue of 10^{-7} as a function of κ_{sea}. Error bars, indicating the 68% confidence levels, are smaller than the plotting symbols.

7.4 Autocorrelations

The sample autocorrelation function on a sequence of measurements $\{x_1, x_2, \ldots, x_n\}$ on (a realisation of) a time series [70, 71] is given by

$$\rho(t) = \frac{\gamma(t)}{\gamma(0)} \tag{7.3}$$

where $\gamma(t)$ is the *sample autocovariance at lag t*

$$\gamma(t) = \frac{1}{n}\sum_{s=1}^{n-t}(x_s - \bar{x})(x_{s+t} - \bar{x}) \tag{7.4}$$

and $\bar{x}$ is the sample mean.[4]

The *exponential autocorrelation time* τ_{exp} characterises the approach of a simulation to the equilibrium distribution. In practice, τ_{exp} is extracted by fitting the autocorrelation function to the model

$$\rho(t) = \exp\left(-\frac{t}{\tau_{\text{exp}}}\right) \tag{7.5}$$

over some range of the lag.

The *integrated autocorrelation time* τ_{int} is defined by

$$\tau_{\text{int}} = \lim_{T\to\infty} \tau_{\text{cum}}(T), \tag{7.6}$$

where

$$\tau_{\text{cum}}(T) = \frac{1}{2} + \sum_{t=1}^{T}\rho(t) \tag{7.7}$$

is sometimes called the *cumulative autocorrelation time*. In practice we have a finite number of measurements and must estimate τ_{int} by looking for a plateau in

[4]$\gamma(t)$ is a biased estimator of a property of the time-series itself, to which we have access only through measurements $\{x_1, x_2, \ldots, x_n\}$ on one particular realisation. Other estimators are possible; the choice made here (and in [4]) of $\frac{1}{n}$ instead of $\frac{1}{n-t}$ in equation (7.4) ensures that the sample auto-correlation matrix is non-negative definite [71, §7.2].

a plot of the cumulative autocorrelation time as a function of T. That this might sometimes be problematic can be seen by inspecting figure 7.7. The integrated autocorrelation time quantifies the effect of autocorrelations on the statistical quality of a sample: the effective number of independent measurements in a sample of n correlated measurements is $n/2\tau_{\text{int}}$ [3, page 384].

Estimates of τ_{exp} and τ_{int} for the plaquette were calculated along these lines in [4], and I shall not reproduce that work here. Instead, I shall use the errors determined in §(7.3) to quote "ball-park estimates" on the plaquette integrated autocorrelation time. The estimates in table 7.3 were calculated from

$$\tau_{\text{int}} \approx \frac{1}{2}\left(\frac{\sigma(m)}{\sigma(1)}\right)^2 \tag{7.8}$$

where $\sigma(k)$ is the error determined by the method of §(7.3) at a bin-size of k, and the bin-size m lies within the plateau. In the absence of any reliable estimate of the uncertainties in the plaquette error bars, these numbers should be taken with a very large grain of salt. Comparing these estimates with the values reported in [4], one sees that [4] is slightly more conservative, but that, within errors, the two sets are consistent. It should be pointed out that there was some variation in the acceptance rate in these simulations, and that the acceptance rate must influence autocorrelation times.[5]

[5]Autocorrelation times diverge as the acceptance rate goes to zero.

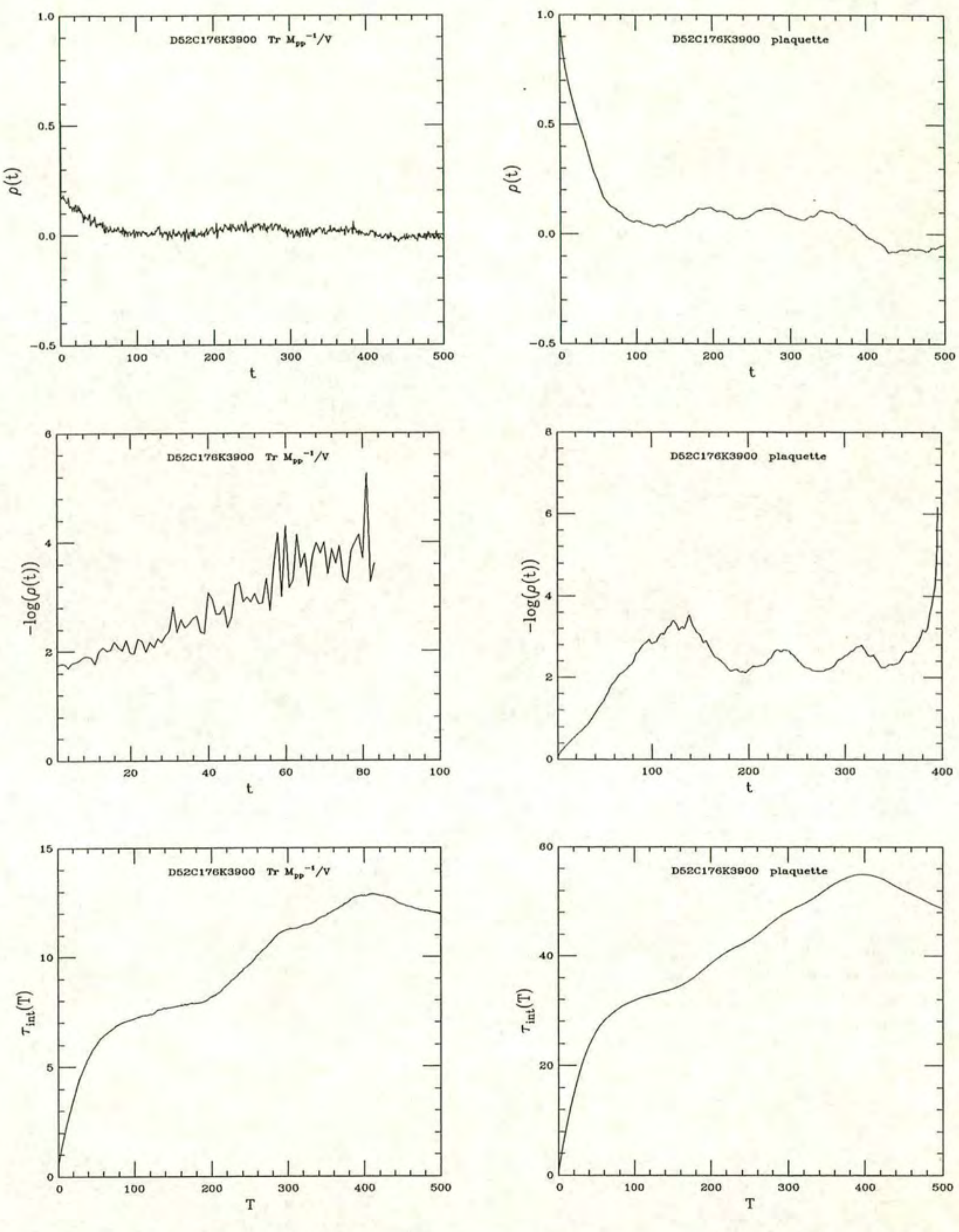


Figure 7.7: Autocorrelation functions (top), their logarithms (middle) and cumulative (bottom) autocorrelation functions for $\mathrm{Tr}\hat{M}_{pp}^{-1}$ (left) and plaquette (right). $\beta = 5.2$, $C_{SW} = 1.76$, $\kappa_{\rm sea} = 0.139$, $V = 12^3 \times 24$.

β	$V/24$	.132	.133	.134	.135	.136	.137	.138	.139	.1395	.1398
5.2	8^3					23	28	29	39	58	
5.2	12^3					16	20	25	35	34	24
5.2	16^3								30	18	
5.3	12^3			13	21	18	17	15			
5.4	12^3	18	11	8	10	10	14				

Table 7.3:
Estimates of integrated autocorrelation times for the plaquette for various β, κ_{sea} and lattice size. These should be taken *cum magno grano salis.*

Given that the integrated autocorrelation times in table 7.3 are rather poorly determined, it would be dangerous to infer any trend of τ_{int} with β, κ or V.

Nonetheless, taking the data at face value, there does appear to be a systematic decrease in τ_{int} as the coupling becomes weaker.[6] To test this tentative hypothesis, reliable determinations of τ_{int} at $\beta = 5.3$ or 5.4 would be required, which in turn would require a substantial increase in statistics. If this trend were confirmed, it would help to account for the fact that we find much longer autocorrelation times at $\beta = 5.2$ than did the SESAM collaboration at $\beta = 5.6$ [72, 73, 68].

[6]The dependence of τ_{int} on β should properly be studied at fixed sea-quark mass. However, the critical value of κ_{sea} decreases as β increases, and we do not have reliable estimates of either sea-quark masses or the critical value of κ_{sea} at $\beta = 5.3$ and $\beta = 5.4$. Instead, we can be guided by the number of solver iterations required for convergence, interpreting table 7.3 in the light of figure 7.6. Thus there is a rough correspondence between $(\beta, \kappa) = (5.2, 0.1395)$, $(5.3, 0.138)$, $(5.4, 0.137)$, and similarly between $(\beta, \kappa) = (5.2, 0.139)$, $(5.3, 0.137)$, $(5.4, 0.136)$.

Chapter 8

Diagonalisation of M — the Lanczos method

8.1 Overview

In this chapter, we describe a method for diagonalising the non-hermitian fermion matrix M.[1]

In outline, the method consists of the following steps.

1. Use the non-hermitian Lanczos method to find a similarity transformation that tridiagonalises M:
$$X^{-1}MX = T.$$

2. Use the QL algorithm with implicit shifts [78] to diagonalise T:
$$S^{-1}TS = \Delta. \tag{8.1}$$
The columns of XS become the (right) eigenvectors of M:
$$M\,(XS) = (XS)\,\Delta. \tag{8.2}$$

3. Normalise the eigenvectors and create an index table [78] for the eigenvalues according to the ordering rule (B.13), so that pairs of eigenvectors $(r_i, \tilde{r}_i)$ corresponding to complex conjugate eigenvalues (ρ_i, ρ_i^*) can be identified.

Steps 1 and 2 are essentially as described in [75, 76]. To the best of my knowledge,

[1]If one is interested only in the eigenvalues of $Q = \gamma_5 M$, then the method of Cullum and Willoughby [74, 75, 76, 77] is more appropriate.

no serious attempt has yet been made to utilise the non-orthogonal basis of eigenvectors $\{r_1, \ldots, r_n\}$ in the calculation of observables.

In view of the spectral expansion for M that was established in §(3.2.2), and in view of equation (3.36), this method yields an all-to-all propagator at almost any value of $\kappa = \kappa_{\text{valence}}$

$$M^{-1}(\kappa) = \sum_{i=1}^{n} \frac{1}{\rho_i(\kappa)} \frac{r_i \tilde{r_i}^\dagger \gamma_5}{\tilde{r}_i^\dagger \gamma_5 r_i}. \tag{8.3}$$

Thus, in principle at least, diagonalising M provides direct access to a wealth of interesting quantities, such as $\det(M)$, $\mathrm{Tr}[M^{-1}]$, $\mathrm{Tr}[\gamma_5 M^{-1}]$ and any hadronic correlation function. Properties of the eigenvectors themselves, especially those corresponding to real eigenvalues, are relevant to the subject of lattice topology.

In practice, the usefulness of this approach is severely limited. The huge memory requirements[2] restrict the applicability of the method to very small lattices, too small, in fact, for any meaningful hadron spectroscopy. As will be seen, the computational effort is completely dominated by the (unfortunately necessary) reorthogonalisation of the Lanczos vectors. The complexity is therefore n^3, which is no better than most dense direct methods.

8.2 Lanczos tridiagonalisation

In this section, we outline a non-hermitian Lanczos method for tridiagonalising the fermion matrix M.

We begin by demanding that there exist a similarity transformation

$$X^{-1}MX = T \tag{8.4}$$

[2]Storage must be allocated for $n = 12V$ vectors, each having n complex components.

where T is complex, symmetric and tridiagonal.

$$T = \begin{pmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \beta_2 & \alpha_3 & \ddots & \\ & & \ddots & \ddots & \beta_{n-1} \\ & & & \beta_{n-1} & \alpha_n \end{pmatrix} \tag{8.5}$$

The reason for insisting that T be symmetric is that this greatly simplifies the anticipated diagonalisation of T using the QL algorithm.

We write

$$X = (x_1, x_2, \dots x_n) \tag{8.6}$$
$$Y = (y_1, y_2, \dots y_n) \tag{8.7}$$

and we require that

$$Y^\dagger = X^{-1}. \tag{8.8}$$

Equation (8.8) gives immediately the biorthogonality property

$$y_i^\dagger x_j = \delta_{ij}, \tag{8.9}$$

which we will make good use of below. From equations (8.4) and (8.8) we have

$$MX = XT \tag{8.10}$$
$$M^\dagger Y = YT^*. \tag{8.11}$$

Performing the matrix multiplications in equations (8.10) and (8.11), inspecting the kth column of the results, and rearranging, leads to

$$\beta_k x_{k+1} = Mx_k - \alpha_k x_k - \beta_{k-1} x_{k-1} \equiv \tilde{x}_{k+1} \tag{8.12}$$
$$\beta_k^* y_{k+1} = M^\dagger y_k - \alpha_k^* y_k - \beta_{k-1}^* y_{k-1} \equiv \tilde{y}_{k+1}. \tag{8.13}$$

An expression for α_k can be obtained by left-multiplying equation (8.12) by $y_k^\dagger$ and using biorthogonality.

$$\alpha_k = y_k^\dagger M x_k. \tag{8.14}$$

Equations (8.12) and (8.13) also provide an expression for β_k^2

$$\beta_k^2 = \tilde{y}_{k+1}^\dagger \tilde{x}_{k+1}. \tag{8.15}$$

Equations (8.12) and (8.13) provide recurrence relations for x_k and y_k, which we supplement with the initial conditions

$$\begin{aligned} x_0 &= y_0 = 0 && (8.16)\\ \beta_0^2 &= \tilde{y}_1^\dagger \tilde{x}_1 && (8.17)\\ x_1 &= \tilde{x}_1/\beta_0 && (8.18)\\ y_1 &= \tilde{y}_1/\beta_0^*. && (8.19) \end{aligned}$$

$\tilde{x}_1$ and $\tilde{y}_1$ may be chosen freely, with the proviso that $\tilde{y}_1^\dagger \tilde{x}_1 \neq 0$.

With these recurrence relations and initial conditions, we have everything that is necessary to write down the algorithm. But we have not yet made use of the γ_5-symmetry of M. Given our experience with BiCG and QMR, we should expect that γ_5-symmetry can be exploited so as to halve the work effort and storage requirements. Here, however, we cannot simply set y_k equal to $\gamma_5 x_k$, because, in general, $x_k^\dagger \gamma_5 x_k$ can be either positive or negative and we must respect equation (8.9). The remedy is to set

$$y_1 = s_1 \gamma_5 x_1 \tag{8.20}$$

where $s_k = \pm 1$ is the sign of $x_k^\dagger \gamma_5 x_k$, and then to show that

$$y_k = s_k \gamma_5 x_k \tag{8.21}$$

implies that

$$y_{k+1} = s_{k+1}\gamma_5 x_{k+1}. \tag{8.22}$$

For the induction to succeed, we need to supplement equations (8.20) and (8.21) by a recurrence relation for the s_k and an initial condition for s_0. A moment's thought reveals that the missing ingredients are

$$\begin{aligned} s_{k+1} &= \frac{\beta_k}{\beta_k^*} s_k, && (8.23) \\ s_0 &= 1. && (8.24) \end{aligned}$$

Proceeding with the induction, one finds that the α_k have become real, that the β_k are either real or purely imaginary, and that the recurrence relation (8.23) is consistent with biorthogonality

$$s_i x_i^\dagger \gamma_5 x_j = \delta_{ij}. \tag{8.25}$$

Before writing down the γ_5-symmetric form of the algorithm, we make a few remarks.

The first remark is that, in exact arithmetic, β_n must vanish. This is because $\tilde{x}_{n+1}$ is orthogonal to a vector space of dimension n and therefore must equal the zero vector. The magnitude of β_n provides one simple diagnostic of convergence — we will consider others below. It is also possible for the algorithm to terminate prematurely with a vanishing β_k, $k < n$. This is exactly the same kind of Lanczos breakdown that arose in the context of solver algorithms in chapters 4 and 5. Exact breakdowns appear to be exceedingly rare on typical gauge configurations, provided that a sensible choice is made for x_1. A near-breakdown can lead to fatal loss of precision.

The second remark is that, in view of biorthogonality, we have more than one way of calculating α_k

$$\alpha_k = s_k x_k^\dagger M x_k \tag{8.26}$$

$$= s_k x_k^\dagger M \left(x_k - x_{k-1}\beta_{k-1}\right). \tag{8.27}$$

These are equivalent in exact arithmetic, but may have different numerical properties in the finite precision that must be used in practice.

The third remark is that round-off errors in finite precision lead inevitably to a loss of biorthogonality between the latest right Lanczos vectors $\{x_k, x_{k+1}, \ldots\}$ and the earliest left Lanczos vectors $\{y_1, y_2, \ldots\}$. Although equation (8.27) seems to perform moderately better in this regard than does equation (8.26), it is usually necessary to re-orthogonalise x_{k+1} against all previous y_j, $j = 1, \ldots, k$ in order to control the magnitude of β_n. This seems to be unavoidable even on lattices as small as $V = 4^4$.

The fourth, and final, remark is that the same algorithm can be used for any γ_5-symmetric matrix. In particular, it can be used to good effect with the red-black preconditioned matrix

$$\hat{M} = (A + \kappa D)\, A^{-1} (A - \kappa D) = \left(\begin{array}{cc} \hat{M}_{ee} & 0 \\ 0 & \hat{M}_{oo} \end{array} \right). \tag{8.28}$$

In view of the decoupling of even sites and odd sites, the eigenspectra of $\hat{M}_{ee}$ and $\hat{M}_{oo}$ can be calculated independently.

The algorithm is given below.

Lanczos Tridiagonalisation of $M = \gamma_5 M^\dagger \gamma_5$

```
$x_0 = 0$
$s_0 = 1$
Choose $\tilde{x}_1$ such that $\tilde{x}_1^\dagger \gamma_5 \tilde{x}_1 \neq 0$
$s_1 = \mathrm{sign}\left(\tilde{x}_1^\dagger \gamma_5 \tilde{x}_1\right)$
$\beta_0 = \begin{cases} \sqrt{\tilde{x}_1^\dagger \gamma_5 \tilde{x}_1} & \text{if } s_1 > 0 \\ \mathrm{i}\sqrt{-\tilde{x}_1^\dagger \gamma_5 \tilde{x}_1} & \text{if } s_1 < 0 \end{cases}$
$x_1 = \tilde{x}_1 / \beta_0$
for $k = 1, 2, \ldots, n$ do {
    $t = M x_k - x_{k-1} \beta_{k-1}$
    $\alpha_k = s_k x_k^\dagger \gamma_5 t$
    $\tilde{x}_{k+1} = t - x_k \alpha_k$
    $\delta_{k+1} = s_k \tilde{x}_{k+1}^\dagger \gamma_5 \tilde{x}_{k+1}$
    $s_{k+1} = s_k \mathrm{sign}(\delta_{k+1})$
    $\beta_k = \begin{cases} \sqrt{\delta_{k+1}} & \text{if } \delta_{k+1} > 0 \\ \mathrm{i}\sqrt{-\delta_{k+1}} & \text{if } \delta_{k+1} < 0 \end{cases}$
    $x_{k+1} = \tilde{x}_{k+1} / \beta_k$
    reorthogonalise $x_{k+1}$ against all previous $y_j = s_j \gamma_5 x_j, \ j = 1, \ldots, k$
    rescale $x_{k+1}$ so that $x_{k+1}^\dagger \gamma_5 x_{k+1} = s_{k+1}$
}
```

The reorthogonalisation procedure is set out below. As the lattice volume is increased the computational effort required by the reorthogonalisation soon becomes the dominant contribution. Even on lattices as small as $V = 4^4$, all other operations, including the n matrix-vector multiplications, are entirely negligible. Therefore, any efforts at performance optimisation should be concentrated on the reorthogonalisation routine. The purpose of the multiplications by γ_5 in equations (8.29) and (8.30) is to permit the use of efficient BLAS routines to perform the dot products inside the loop.

Reorthogonalisation

```
$x_{k+1} = \gamma_5 x_{k+1}$                                    (8.29)
$w = 0$
for $j = 1, 2, \ldots, k$ do {
    $w = (s_j x_j^\dagger x_{k+1}) x_j + w$
}
$x_{k+1} = \gamma_5 x_{k+1}$                                    (8.30)
$x_{k+1} = -w + x_{k+1}$
```

8.3 Implementation

Most of the operations required by the Lanczos tridiagonalisation routine are the same matrix-vector and vector-vector operations required by the solver algorithms of chapter 4. It is therefore natural to use the same geometric decomposition of the lattice that was used in the other codes discussed in this thesis. Even the reorthogonalisation procedure can be built out of the same elementary building blocks. Indeed, this is one reason why the Lanczos method was attractive in the first place. On a parallel architecture, the tridiagonalisation routine naturally leads to a situation in which every processor has its own copy of the tridiagonal matrix T. This gives rise to a memory overhead equivalent to $2\,(N_{PE}-1)$ additional vectors, N_{PE} being the number of processor elements; this overhead is usually insignificant compared to the n Lanczos vectors.

The diagonalisation of T is not an operation that we have encountered previously in this thesis. I use the complex QL algorithm with implicit shifts to accomplish the diagonalisation.[3] I do not describe the QL algorithm here, but I do make two remarks.

The QL algorithm accomplishes the diagonalisation through a sequence of elementary rotations; these same rotations can be applied simultaneously to the Lanczos vectors, so that, when the diagonalisation is complete, the Lanczos vectors have been overwritten by the eigenvectors of M.

The diagonalisation of T itself does not parallelise in any natural way, but this is of little consequence as the cost of the entire operation is negligible compared to reorthogonalisation. However, the extra processors do make their presence felt during the back-transformation of the Lanczos vectors, which turns out to be embarrassingly parallel.

The creation of the index table in step 3 is an interesting problem. In exact arithmetic, the solution would be trivial: sort the eigenvalues by their real part, using first the modulus and then the sign of the imaginary parts to break ties. In the absence of any degenerate eigenvalues, this simple approach would be entirely sufficient to identify complex-conjugate partners — they will appear next to each other after sorting.

The problem is not so trivial in finite arithmetic, because there will be some error in the determination of each eigenvalue. Sorting the eigenvalues by their real part is still useful, and will serve to bring complex conjugate partners close

[3] I am indebted to Christine Davies for providing the source of a serial code to perform this operation.

together, but they are no longer guaranteed to be adjacent. Some refinement is possible by imposing cuts in numerical tests of equality, so that two numbers are considered equal if the absolute value of their difference is smaller than some predefined tolerance. My current implementation proceeds along these lines, and usually succeeds, more by good luck than good management, on lattices of size $V = 4^3 \times 2$.

On larger lattices the problem becomes more difficult for two reasons. Firstly, the eigenvalues in the centre of the spectrum will tend to have slightly larger errors. Secondly, the real parts of the eigenvalues become more dense in the real line. Thus it becomes more likely that two or more complex conjugate pairs of eigenvalues will be interleaved after sorting. My current implementation usually fails to identify complex conjugate partners correctly on lattices of size $V = 4^3 \times 8$, and more sophisticated logic is required. It is possible to devise a reliable algorithm which uses the sort only to identify candidate pairs of eigenvectors (r_i, r_j), and confirms or refutes their complex conjugacy by computing $r_i^\dagger \gamma_5 r_j$. This quantity is typically of order 10^{-1} if ρ_i and ρ_j are genuinely complex conjugate partners, but of order 10^{-10} otherwise.

8.4 Verification

There are various means of verifying the correctness of the eigenvalues and eigenvectors computed by the method.

The magnitude of β_n provides a direct quantitative check on the convergence of the Lanczos tridiagonalisation routine. It should be consistent with zero.

Since $\sigma_{\mu\nu} F_{\mu\nu}$ and the hopping term are both traceless, summing the eigenvalues provides a simple consistency check. One should find

$$\mathrm{Tr}\, M = \sum_{i=1}^{n} \rho_i = n. \tag{8.31}$$

Direct tests of the quality of the eigenvalues and eigenvectors are easily performed by computing $||Mr_i - \rho_i r_i||$. The overhead is considerable, but nonetheless small compared to reorthogonalisation.

If the method is applied to the full fermion matrix and to both parities of the red-black preconditioned matrix, equation (6.71) suggests checking that

$$\mathrm{Tr}\ M^{-1} - \mathrm{Tr}\ \hat{M}_{ee}^{-1} - \mathrm{Tr}\ \hat{M}_{oo}^{-1}$$

is consistent with zero.

The quantity

$$\sum_{i=1}^{n} \frac{\tilde{r_i}^{\dagger} r_i}{\tilde{r}_i^{\dagger} \gamma_5 r_i}$$

provides a potent consistency check on the eigenvectors. As shown in section §(3.2.2), this quantity should equal zero. It tests simultaneously the quality of the eigenvectors and whether the matching of complex conjugate pairs of eigenvalues has been performed correctly. On a $V = 4^3 \times 2$ lattice, this quantity is typically of order 10^{-5}, but it will jump by 10 orders of magnitude or more if any misidentification of complex conjugate partners is made.

Eigenvalue spectra calculated by this method on a $4^3 \times 8$ lattice can be seen in figures 8.1 and 8.2. The former shows the spectrum of M, and the latter the spectra of $\hat{M}_{ee}$ and $\hat{M}_{oo}$. Both were calculated on the same gauge configuration, which was generated by the HMC method of chapter 6 with $N_f = 2$, $\beta = 5.2$, and $C_{SW} = 2.02$. The quark mass is rather heavy at $\kappa_{\text{sea}} = \kappa_{\text{valence}} = 0.13$.

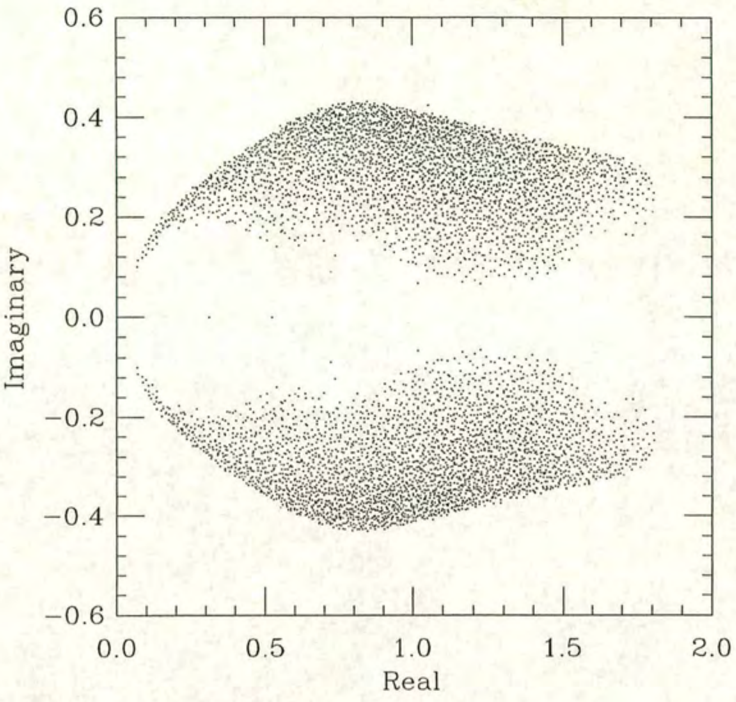
The eigenvalue spectrum in figure 8.3 was calculated on another dynamical gauge configuration at the same coupling but lighter quark mass ($\kappa_{\text{sea}} = \kappa_{\text{valence}} = 0.134$). The lighter quark mass does appear to induce some subtle differences,[4] but I caution the reader against inferring too much from a single gauge configu-

[4]The small eigenvalues are closer to the origin and to the imaginary axis at the lighter mass, and there appears to be a greater suppression of eigenvalues near the real axis at the lighter mass. Whether this second effect is genuinely significant, or merely a statistical fluctuation, or symptomatic of incomplete equilibration in the simulation at the heavier mass, is not known to me. Greatly increased statistics would be required to clarify this point.

ration.

The distribution of the small eigenvalues of M (or $M^\dagger M$) is relevant to the convergence behaviour of the iterative methods discussed in chapters 4 and 5. The condition number of the fermion matrix depends sensitively on the location of the smallest eigenvalue. Whether block algorithms provide any significant speed-up depends on the blocking factor s and the location of the s smallest eigenvalues. The benefits of red-black preconditioning can clearly be seen by comparing figure 8.2 to figure 8.1; the eigenvalues of $\hat{M}$ are confined to a smaller ellipse than are those of M, and the smallest eigenvalues of $\hat{M}$ are roughly twice as large as those of M.

The matrix whose eigenvalues are plotted in figure 8.1 possesses two genuinely real eigenvalues, confirmed by the non-zero chiralities of the corresponding eigenvectors. Now, in simulations with dynamical fermions, gauge field configurations on which the fermion matrix possesses zero modes at $\kappa_{\text{valence}} = \kappa_{\text{sea}}$ are expected to be suppressed by the presence of the fermionic determinant in the effective action. However, as pointed out in chapter 3, a real eigenvalue can be shifted to zero by varying κ_{valence}. Computing propagators at $\kappa_{\text{valence}} \neq \kappa_{\text{sea}}$ is *partially* quenched, yet the practice is not uncommon in hadron spectroscopy with dynamical fermions. Under these circumstances it is still possible to encounter exceptional configurations, the plague of the quenched approximation; the Modified Quenched Approximation [79] is designed to address this problem by shifting the poles of the quark propagator in a controlled way.



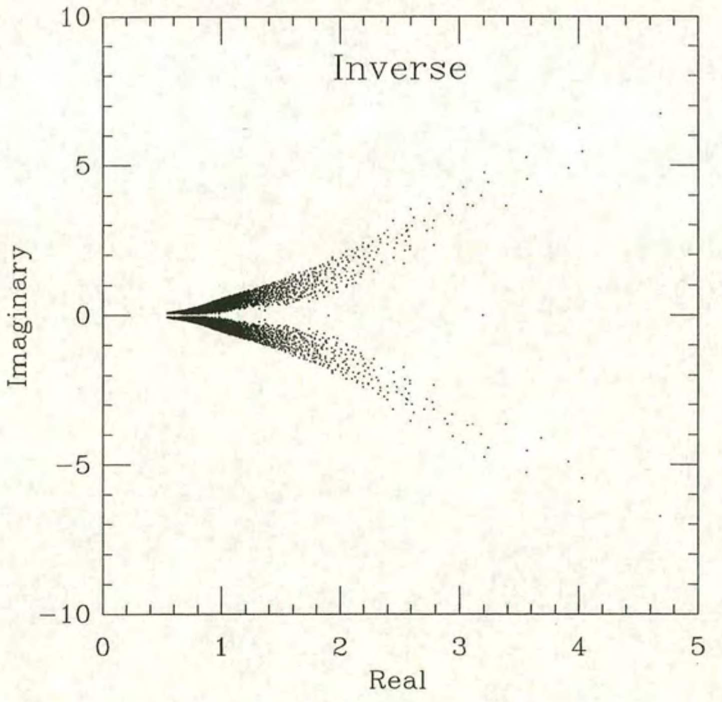


Figure 8.1: Eigenvalue spectra of the Wilson fermion matrix (top) and its inverse (bottom). $V = 4^3 \times 8$, $\beta = 5.2$, $C_{SW} = 2.02$ (non-perturbatively improved), $N_f = 2$, $\kappa_{\text{sea}} = \kappa_{\text{valence}} = 0.13$.

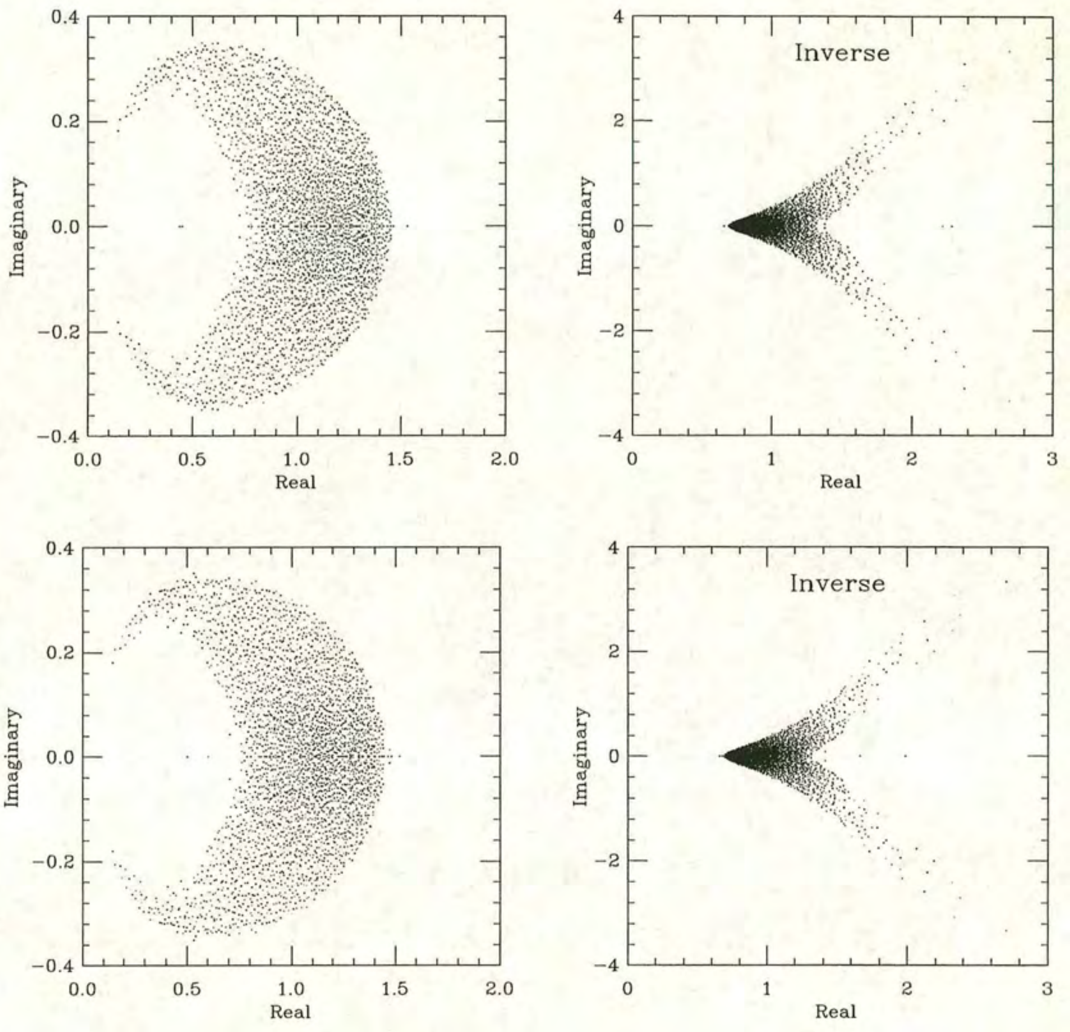


Figure 8.2: Eigenvalue spectra of the red-black preconditioned fermion matrix on the even parity (top left) and the odd parity (bottom left). The spectra of the corresponding inverse matrices are shown to the right. $N_f = 2$, $\beta = 5.2$, $C_{SW} = 2.02$ (non-perturbatively improved), $N_f = 2$, $\kappa_{\text{sea}} = \kappa_{\text{valence}} = 0.13$. The gauge configuration is the same as for figure 8.1.

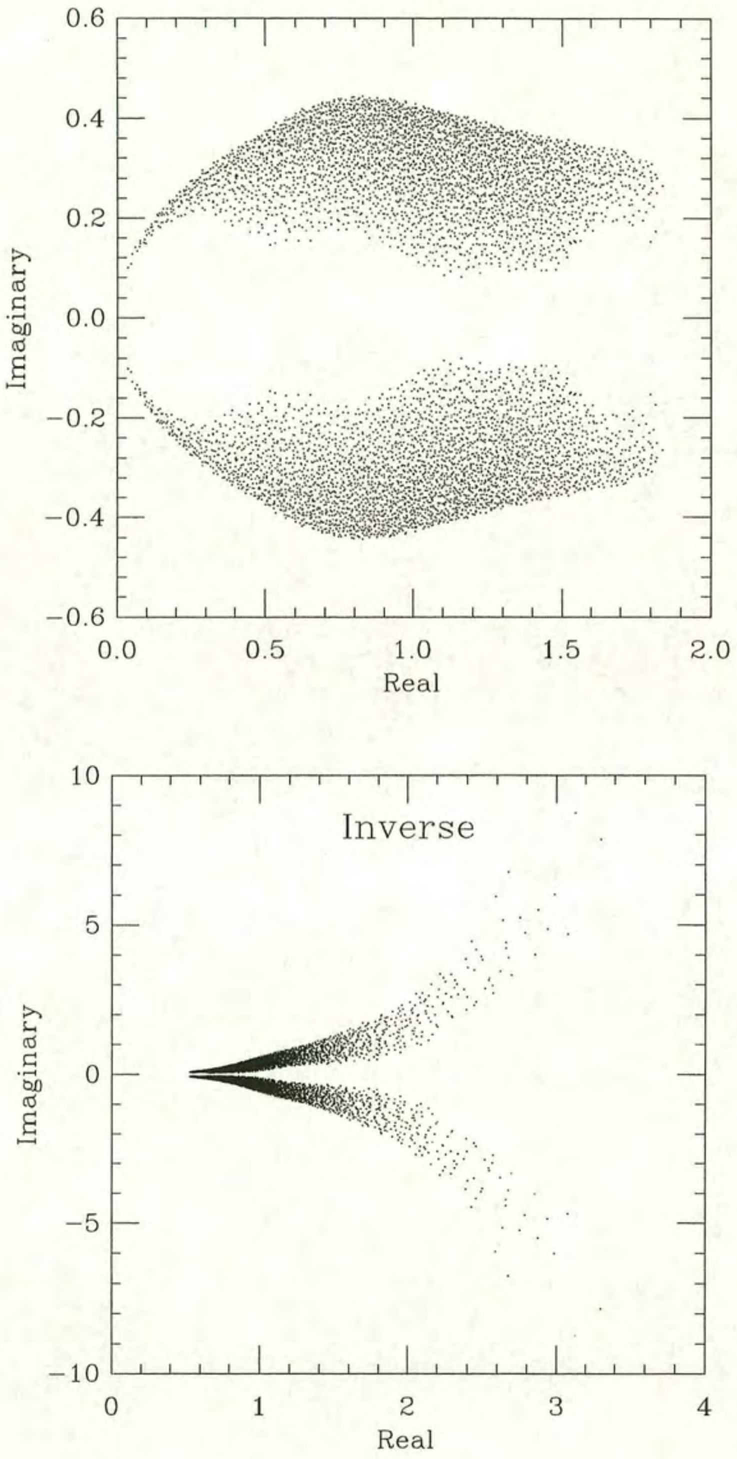


Figure 8.3: Eigenvalue spectra of the Wilson fermion matrix (top) and its inverse (bottom). $V = 4^3 \times 8$, $\beta = 5.2$, $C_{SW} = 2.02$ (non-perturbatively improved), $N_f = 2$, $\kappa_{\text{sea}} = \kappa_{\text{valence}} = 0.134$. Contrast figure 8.1.

Chapter 9

Conclusions and recommendations

A large part of this thesis concentrated on algorithms for solving linear systems involving the Wilson fermion matrix. Most attention was given to the case of clover-improvement and red-black preconditioning. I described and implemented various algorithms, including BiCG(γ_5), QMR(γ_5), BiCGSTAB and BiCGSTAB2, and compared their performance with each other and with a pre-existing implementation of MR. BiCGSTAB was found to be the method of choice in most of the regimes studied. There remain, however, places for MR, QMR(γ_5) and even CGNR in any fully equipped armoury of tools for quark propagator calculation.

It is unlikely that future research in Krylov-subspace methods will lead to any dramatic improvements over the methods considered here. Those attempting to accelerate quark propagator calculations are advised to spend their energies in areas where research is less mature. At present, the search for better preconditioners seems to hold much promise. LL-SSOR has already emerged as the preconditioner to beat.[1]

Chapter 5 was concerned with block algorithms, iterative methods in which the solutions of linear systems with multiple right-hand sides are determined simultaneously. I described, implemented and compared four block algorithms: B-CGNR, B-Lanczos, B-BiCG(γ_5) and B-QMR(γ_5). Of these, only the first two

[1]Readers who are interested in experimenting with LL-SSOR and who have access to the source of the UKQCD MPP codes, should note that a LL-SSOR preconditioned version of MR may be found in `mr_solver_ssor_1.F`, courtesy of its author, Gero Ritzenhöfer. The matrix multiplication is not optimised, and it is restricted to the unimproved case $C_{SW} = 0$. A version using BiCGSTAB may be found in `bicgstab_solver_ssor.F`.

had been studied previously in lattice QCD. It was found that B-BiCG(γ_5) and B-QMR(γ_5) both outperform B-CGNR and B-Lanczos very significantly, a fact which is attributed to the superior spectral properties of M compared with $M^\dagger M$ and $\gamma_5 M$. However, none of these methods performed sufficiently well to be preferred to BiCGSTAB at the weak couplings and large lattices that characterise the UKQCD collaboration's current programme of hadron spectroscopy. If block algorithms have a rôle to play in the future of lattice QCD, it is most likely to be at stronger coupling and lighter mass; the possibility of block algorithms should be borne in mind by anyone contemplating simulations with highly-improved fermionic actions on coarse lattices.

Chapter 6 described an implementation of the Generalised Hybrid Monte Carlo algorithm, and much attention was given to methods for improving performance. Although hardware-specific optimisations were most significant in this regard, I focused on more portable, algorithmic improvements. Of the various tricks and techniques that were discussed in chapter 6, four in particular are worthy of recapitulating here.

Replacing one Conjugate Gradient solve by two BiCGSTAB solves was found to result in savings of up to 40%, but it is not known whether this will remain true at much lighter sea-quark masses. Once it is accepted that BiCGSTAB should be used to perform the solves, the wisdom of formulating GHMC in terms of the determinant of the red-black preconditioned matrix can be called into question. If instead one defines pseudofermions on both parities and works with the full fermion matrix, a wider variety of preconditioning strategies, including LL-SSOR, becomes available. Moreover, it has been suggested [68] that defining pseudofermions on half the lattice sites could have an adverse effect on autocorrelation times.

The next two tricks are related to the clover term. Exploiting a symmetry in the block structure of the clover term halved the memory required to store the $L^\dagger DL$ decomposition of A, and reduced the effort required to compute the decomposition by a factor of four. The cost of multiplying by the inverse of A

was approximately halved. This trick has since been incorporated into the solver code.

It was found that applying a further block Jacobi preconditioner to the red-black preconditioned system yielded a modest improvement in the convergence rate of solvers in GHMC. This preconditioner has not yet been retrofitted to the solver code, but the effort should be worthwhile. On quenched configurations, improvements to MR of up to 30 per cent have been observed.[2] BiCGSTAB can be handled in exactly the same way as in the GHMC code. It *is* possible to apply block Jacobi preconditioning to the γ_5-symmetric algorithms BiCG(γ_5) and QMR(γ_5) as a *central* preconditioner, because the matrix $\tilde{M}_{pp} = A_{pp}^{-\frac{1}{2}} \hat{M}_{pp} A_{pp}^{-\frac{1}{2}}$ *is* γ_5-symmetric. There will be two ways to proceed. The obvious way is to work with $\tilde{M}_{pp}$ directly, but multiplications by $\tilde{M}_{pp}$ will clearly be more expensive than multiplications by $\hat{M}_{pp}$. The alternative is to absorb the factors of $A_{pp}^{-\frac{1}{2}}$ into redefinitions of the various vectors that appear in the recurrence relations; one should find that all explicit occurrences of $A_{pp}^{-\frac{1}{2}}$ can be eliminated, but that inner products will now involve A_{pp}.[3] In either case, the preconditioner will involve some additional overhead to offset any gains from faster convergence, and the net improvement may be too small to justify the coding effort.

The fourth technique is nothing other than our mechanisms for checkpointing and resuming a GHMC simulation. These mechanisms enabled us to make maximum use of the available computing resources and to minimise the impact of system failures, and may validly be regarded as an important optimisation.

Chapter 7 described our first experiences with Hybrid Monte Carlo simulations using two mass-degenerate flavours of dynamical clover-improved Wilson fermions. It raises questions concerning the effect of the coupling constant on autocorrelation times.

[2]Users with access to the source of the UKQCD MPP codes can find an example using the MR algorithm in `mr_solver_2.F`.

[3]O'Leary's formulation of B-BiCG in §(5.2.4) applied central preconditioning along these very lines.

Chapter 8 announced the existence of a parallel Lanczos code to diagonalise the fermion matrix. It is my hope that this code will prove useful.

Appendix A

Notation and Conventions

A.1 Matrix notation

Suppose A is an arbitrary $m \times n$ matrix. Then a_{ij} shall denote the component of A in the ith row and jth column, and A_k denotes the kth column of A. Sometimes we write $A = (a_{ij})$. A^T and $A^\dagger$ denote respectively the transpose and hermitian conjugate of A, ie.

$$A^T = (a_{ji}), \; A^\dagger = \left(a_{ji}^*\right).$$

Iteration numbers on a matrix or vector are indicated by superscripts in parentheses, eg. $A^{(k)}$ denotes the matrix A in the kth iteration. As the inverse of the transpose (hermitian conjugate) of a square matrix equals the transpose (hermitian conjugate) of its inverse, we also write unambiguously

$$\left(A^T\right)^{-1} = \left(A^{-1}\right)^T \equiv A^{-T}, \; \left(A^\dagger\right)^{-1} = \left(A^{-1}\right)^\dagger \equiv A^{-\dagger}.$$

I trust my readers to correctly infer the dimension of the identity matrix, which I denote by I, from the context in which it occurs. The notation δ_{ij} is reserved for the Kronecker delta.

A.2 Dirac matrices

We use the following representation of the Euclidean gamma matrices:

$$\gamma_1 = \begin{pmatrix} 0 & 0 & 0 & i \\ 0 & 0 & i & 0 \\ 0 & -i & 0 & 0 \\ -i & 0 & 0 & 0 \end{pmatrix} \quad \gamma_2 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

$$\gamma_3 = \begin{pmatrix} 0 & 0 & i & 0 \\ 0 & 0 & 0 & -i \\ -i & 0 & 0 & 0 \\ 0 & i & 0 & 0 \end{pmatrix} \quad \gamma_4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

and

$$\gamma_5 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \tag{A.1}$$

Here the spinor indices μ, ν run from 1 to 4, corresponding to x, y, z, t.

$$\sigma_{\mu\nu} = \frac{i}{2}\left[\gamma_\mu, \gamma_\nu\right] \tag{A.2}$$

Explicitly,

$$\begin{aligned}
\sigma_{12} &= \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} & \sigma_{13} &= \begin{pmatrix} 0 & -i & 0 & 0 \\ i & 0 & 0 & 0 \\ 0 & 0 & 0 & -i \\ 0 & 0 & i & 0 \end{pmatrix} \\
\sigma_{14} &= \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} & \sigma_{23} &= \begin{pmatrix} 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \end{pmatrix} \\
\sigma_{24} &= \begin{pmatrix} 0 & 0 & 0 & -i \\ 0 & 0 & i & 0 \\ 0 & -i & 0 & 0 \\ i & 0 & 0 & 0 \end{pmatrix} & \sigma_{34} &= \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{pmatrix}
\end{aligned} \tag{A.3}$$

A.3 Gell-Mann matrices

We adopt the same conventions for the Gell-Mann matrices as given in [80, pages 516-7], that is

$$\begin{aligned}
\lambda_1 &= \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} & \lambda_2 &= \begin{pmatrix} 0 & -i & 0 \\ i & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} & \lambda_3 &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \\
\lambda_4 &= \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} & \lambda_5 &= \begin{pmatrix} 0 & 0 & -i \\ 0 & 0 & 0 \\ i & 0 & 0 \end{pmatrix} & \lambda_6 &= \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \\
\lambda_7 &= \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -i \\ 0 & i & 0 \end{pmatrix} & \lambda_8 &= \frac{1}{\sqrt{3}}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -2 \end{pmatrix}
\end{aligned} \tag{A.4}$$

These are traceless, hermitian and normalised such that

$$\mathrm{Tr}\ [\lambda_a \lambda_b] = 2\delta_{ab}. \tag{A.5}$$

Appendix B

Eigenvectors of J-hermitian matrices

Let $A \in C^{N\times N}$ be a J-hermitian matrix. That is, there exists a non-singular matrix $J \in C^{N\times N}$ such that

$$JA = A^{\dagger}J. \tag{B.6}$$

Since J is invertible, A and $A^{\dagger}$ are similar matrices and have the same Jordan form, trace, determinant, characteristic polynomial and minimum polynomial.

Considering the characteristic polynomial of A

$$\begin{aligned}
\det(A-\lambda I) &= \det(J)\det(A-\lambda I)\det\left(J^{-1}\right) && \text{(B.7)}\\
&= \det\left(JAJ^{-1}-\lambda I\right) && \text{(B.8)}\\
&= \det\left(A^{\dagger}-\lambda I\right) && \text{(B.9)}\\
&= \det\left(A^{*}-\lambda I\right) && \text{(B.10)}\\
&= \det\left(A-\lambda^{*} I\right)^{*} && \text{(B.11)}
\end{aligned}$$

we observe that if λ is a zero of the characteristic polynomial, then λ^* is too. Consequently the eigenvalues of A are real or occur in complex conjugate pairs. Moreover, λ_i and λ_i^* have the same multiplicity.

We assume that A is diagonalisable, that is there exist N eigenvalues λ_i (not necessarily distinct) corresponding to N linearly independent eigenvectors v_i. That is,

$$Av_i = \lambda_i v_i. \tag{B.12}$$

In general, the v_i will not be orthogonal. We do not require that the v_i be normalised.

Without loss of generality, let the eigenpairs (λ_i, v_i) be sorted on key

$$(\mathrm{Re}\ \lambda_i, |\mathrm{Im}\ \lambda_i|, \mathrm{sign}(\mathrm{Im}\ \lambda_i)). \tag{B.13}$$

By this we mean that $i \leq j$ implies

$$\begin{aligned} \mathrm{Re}\ \lambda_i &\leq \mathrm{Re}\ \lambda_j \\ \mathrm{Re}\ \lambda_i = \mathrm{Re}\ \lambda_j &\implies |\mathrm{Im}\ \lambda_i| \leq |\mathrm{Im}\ \lambda_j| \\ \mathrm{Re}\ \lambda_i = \mathrm{Re}\ \lambda_j \text{ and } |\mathrm{Im}\ \lambda_i| = |\mathrm{Im}\ \lambda_j| &\implies \mathrm{sign}(\mathrm{Im}\ \lambda_i) \leq \mathrm{sign}(\mathrm{Im}\ \lambda_j). \end{aligned}$$

Degenerate eigenvalues are ordered arbitrarily, but consecutively. Forming the matrices

$$V = (v_1, \ldots, v_N) \tag{B.14}$$

$$\Lambda = \mathrm{diag}(\lambda_i) \tag{B.15}$$

we can re-write the eigenvalue equation (B.12) in matrix form

$$AV = V\Lambda. \tag{B.16}$$

The assumption of linearly independent eigenvectors means that V has rank N and is consequently non-singular.

Because the eigenvalues occur in complex conjugate pairs, the sets $\{\lambda_i^*\}$ and $\{\lambda_i\}$ are the same. We can write

$$A\tilde{v}_i = \lambda_i^* \tilde{v}_i \tag{B.17}$$

and arrange for the eigenpairs $(\lambda_i^*, \tilde{v}_i)$ to be ordered by the same rule as above, ie. we sort these eigenpairs on key

$$(\mathrm{Re}\ \lambda_i, |\mathrm{Im}\ \lambda_i^*|, \mathrm{sign}(\mathrm{Im}\ \lambda_i^*)),$$

but with the additional constraint that degenerate eigenpairs appear in the same order in both schemes. If λ_i is real and simple, then v_i and $\tilde{v}_i$ coincide.

In terms of the matrix

$$\tilde{V} = (\tilde{v}_1, \ldots, \tilde{v}_N) \tag{B.18}$$

we have

$$A\tilde{V} = \tilde{V}\Lambda^*. \tag{B.19}$$

Multiplying equation (B.16) on the left by J and using equation (B.6) one finds that the i-th column of JV is an eigenvector of $A^\dagger$ with eigenvalue λ_i. Doing the same with equation (B.17) gives that the i-th column of $J\tilde{V}$ is an eigenvector of $A^\dagger$ with eigenvalue λ_i^*.

Notice that, on the one hand,

$$\tilde{v}_i^\dagger J A v_j = \lambda_j \tilde{v}_i^\dagger J v_j \tag{B.20}$$

but on the other hand

$$\begin{aligned}
\tilde{v}_i^\dagger J A v_j &= \tilde{v}_i^\dagger A^\dagger J v_j && \text{(B.21)}\\
&= (A\tilde{v}_i)^\dagger J v_j && \text{(B.22)}\\
&= (\lambda_i^* \tilde{v}_i)^\dagger J v_j && \text{(B.23)}\\
&= \lambda_i \tilde{v}_i^\dagger J v_j. && \text{(B.24)}
\end{aligned}$$

So if λ_i and λ_j are distinct, then $\tilde{v}_i^\dagger J v_j = 0$.

The matrix form of this result is

$$\left[\Lambda, \tilde{V}^\dagger J V\right] = 0. \tag{B.25}$$

This means that the matrix $\tilde{V} J V$, which has components

$$\left(\tilde{V}^\dagger J V\right)_{ij} = \tilde{v}_i^\dagger J v_j \tag{B.26}$$

is block diagonal. There is one block for each *distinct* eigenvalue, and the size of the block is given by the multiplicity of the corresponding eigenvalue. We now observe that any linear combination of eigenvectors corresponding to the same eigenvalue λ is itself an eigenvector with eigenvalue λ, and we set about exploiting this freedom to find matrices of eigenvectors W and $\tilde{W}$ such that $\tilde{W}^\dagger JW$ *is* diagonal, at the same time as satisfying

$$\begin{aligned} AW &= W\Lambda && \text{(B.27)} \\ A\tilde{W} &= \tilde{W}\Lambda. && \text{(B.28)} \end{aligned}$$

The procedure is analogous to finding an orthonormal set of eigenvectors corresponding to a multiple eigenvalue of an hermitian matrix, except that here the eigenvectors (the columns of W) are to be bi-orthogonal with respect to another set of vectors (the columns of $J^\dagger\tilde{W}$).

Let B_j denote the block corresponding to the j-th distinct eigenvalue μ_j of multiplicity m_j. Given the ordering rule (B.13) on the λ_i, we have

$$\mu_j = \lambda_{i_j} = \ldots = \lambda_{i_j+m_j-1} \tag{B.29}$$

for some i_j. Defining

$$\begin{aligned} V_j &\equiv \left(v_{i_j}, \ldots, v_{i_j+m_j-1}\right) && \text{(B.30)} \\ \tilde{V}_j &\equiv \left(\tilde{v}_{i_j}, \ldots, \tilde{v}_{i_j+m_j-1}\right) && \text{(B.31)} \end{aligned}$$

we have

$$B_j = \tilde{V}_j^\dagger JV_j \tag{B.32}$$

and we may take its LDU decomposition

$$\tilde{V}_j^\dagger JV_j = L_jD_jU_j. \tag{B.33}$$

Here, D_j is diagonal, L_j is lower triangular and U_j is upper triangular. Both L_j and U_j have ones down the diagonal and are therefore invertible. The LDU

decomposition of $\tilde{V}^\dagger JV$ is just the direct sum of the $L_j D_j U_j$.

Taking

$$\begin{aligned} W_j &= V_j U_j^{-1} && \text{(B.34)} \\ \tilde{W}_j &= \tilde{V}_j L_j^{-\dagger}. && \text{(B.35)} \end{aligned}$$

we have by construction that

$$\tilde{W}_j^\dagger J W_j = D_j \tag{B.36}$$

which is diagonal, and plainly

$$\begin{aligned} AW_j = AV_j U_j^{-1} &= V_j \mu_j I U_j^{-1} = W_j \mu_j && \text{(B.37)} \\ A\tilde{W}_j = A\tilde{V}_j L_j^{-\dagger} &= \tilde{V}_j \mu_j^* I L_j^{-\dagger} = \tilde{W}_j \mu_j^*. && \text{(B.38)} \end{aligned}$$

It is now plain that

$$\begin{aligned} W &= (W_1, \ldots, W_l) \equiv (w_1, \ldots, w_N) && \text{(B.39)} \\ \tilde{W} &= \left(\tilde{W}_1, \ldots, \tilde{W}_l\right) \equiv (\tilde{w}_1, \ldots, \tilde{w}_N) && \text{(B.40)} \end{aligned}$$

satisfy equation (B.27), and furthermore

$$\tilde{W}^\dagger J W = \bigoplus_{j=1}^{l} B_j = \bigoplus_{j=1}^{l} D_j = D. \tag{B.41}$$

Now the left hand side of equation (B.41) is the product of non-singular matrices and hence non-singular. Therefore D is also non-singular, and

$$W^{-1} = D^{-1} \tilde{W}^\dagger J. \tag{B.42}$$

Multiplying equation (B.27) from the right by W^{-1} we obtain a spectral decomposition for A:

$$A = W \Lambda D^{-1} \tilde{W}^\dagger J. \tag{B.43}$$

Observing that

$$\tilde{w}_i{}^\dagger J w_i = D_{ii} \tag{B.44}$$

equation (B.43) may also be expressed in the form:

$$A = \sum_{i=1}^{N} \lambda_i \frac{w_i \tilde{w}_i^\dagger J}{\tilde{w}_i^\dagger J w_i}. \tag{B.45}$$

This equation (B.45) is our main result. We observe that if J is the identity matrix, then A is hermitian, $\tilde{w}_i = w_i$ and equation (B.45) reduces to the familiar spectral decomposition for an hermitian matrix.

It should be pointed out that the ordered set $\{\tilde{w}_i\}$ is *not* in general a permutation of $\{w_i\}$, unless all the eigenvalues of A are simple (in which case the LDU decomposition is trivial and $W = V$ and $\tilde{W} = \tilde{V}$). However, this is remedied automatically if J has certain hermiticity properties as we shall now show.

We assume now that J is hermitian[4]. Consider the j-th distinct eigenvalue μ_j and its corresponding block $B_j = \tilde{V}_j^\dagger J V_j$.

Suppose first that μ_j is real. By the ordering rule, $\tilde{V}_j = V_j$, and by the hermiticity of J, B_j is hermitian. Then the LDU decomposition of B_j reduces to the $LDL^\dagger$ decomposition. We find that

$$W_j = V_j L_j^{-\dagger} \tag{B.46}$$

and

$$\tilde{W}_j = \tilde{V}_j L_j^{-\dagger} = V_j L_j^{-\dagger} = W_j. \tag{B.47}$$

Notice that if J were not hermitian, each column of $\tilde{W}_j$ would be some linear combination of the columns of W_j.

Now suppose that μ_j is not real. Without loss of generality, we may take Im $\mu_j < 0$. Perform the LDU decomposition of the corresponding block.

$$B_j = \tilde{V}_j^\dagger J V_j = L_j D_j U_j. \tag{B.48}$$

[4]If J is anti-hermitian, work with the hermitian matrix $J' = iJ$ instead.

By the ordering rule the next block B_{j+1} corresponds to $\mu_{j+1} = \mu_j^*$, and furthermore

$$\tilde{V}_j = V_{j+1}, \qquad \tilde{V}_{j+1} = V_j. \tag{B.49}$$

Therefore

$$B_{j+1} = \tilde{V}_{j+1}^\dagger J V_{j+1} = V_j^\dagger J \tilde{V}_j = V_j^\dagger J^\dagger \tilde{V}_j = B_j^\dagger. \tag{B.50}$$

Thus the hermiticity of J permits us to set consistently

$$\tilde{W}_{j+1} = W_j = V_j U_j^{-1} \tag{B.51}$$

$$W_{j+1} = \tilde{W}_j = \tilde{V}_j L_j^{-\dagger} \tag{B.52}$$

and we only need to perform *one* LDU decomposition.

The practical importance of the ordering rule is now clear. Given an hermitian matrix J, a J-hermitian matrix A, and any complete set of eigenvalues and corresponding eigenvectors, we can apply an ordering rule to construct two numberings of the eigenpairs, and then construct, in place, another set of eigenpairs $\{(\lambda_i, w_i)\}$ satisfying equation (B.45). Moreover, there is no need to store the $\{(\lambda_i^*, \tilde{v}_i)\}$ as these are given in terms of the $\{(\lambda_i, w_i)\}$ by the ordering rule.

One might question the usefulness of this result. After all, if J is hermitian, JA is hermitian, and A can be expanded in terms of the eigenvectors of JA. However, it may be the case that one is interested in properties of the eigenvectors of A. Alternatively, it may be the case that A is a shifted matrix and JA is not, and that one is interested in the properties of A or A^{-1} at several values of the shift. The Wilson fermion matrix in lattice QCD is such a matrix.

Appendix C

The structure of the clover term

On each site, the clover term A is given by

$$A = 1 - \frac{1}{2}\kappa C_{SW}\sigma_{\mu\nu}F_{\mu\nu} \tag{C.53}$$

In the obvious basis (colour indices run faster than spin), $\frac{1}{2}\sigma_{\mu\nu}F_{\mu\nu}$ has the block form

$$\frac{1}{2}\sigma_{\mu\nu}F_{\mu\nu} = -\begin{pmatrix} W_1 & W_2 & W_3 & W_4 \\ W_2^\dagger & -W_1 & W_4^\dagger & -W_3 \\ W_3 & W_4 & W_1 & W_2 \\ W_4^\dagger & -W_3 & W_2^\dagger & -W_1 \end{pmatrix} \tag{C.54}$$

where the 3×3 matrices $W_1 \ldots W_4$ are given by

$$\begin{aligned} W_1 &= F_{12} \\ W_2 &= iF_{13} + F_{23} \\ W_3 &= -F_{34} \\ W_4 &= iF_{24} - F_{14} \end{aligned} \tag{C.55}$$

Note that W_1 and W_3 are hermitian.

It is now apparent that the clover term can be written in the form

$$A = \begin{pmatrix} B & C \\ C & B \end{pmatrix} \tag{C.56}$$

where

$$B = \begin{pmatrix} 1 + \kappa C_{SW} W_1 & \kappa C_{SW} W_2 \\ \kappa C_{SW} W_2^\dagger & 1 - \kappa C_{SW} W_1 \end{pmatrix} \tag{C.57}$$

$$C = \begin{pmatrix} \kappa C_{SW} W_3 & \kappa C_{SW} W_4 \\ \kappa C_{SW} W_4^\dagger & -\kappa C_{SW} W_3 \end{pmatrix} \tag{C.58}$$

We now seek a similarity transformation S such that $S^{-1}AS$ is block diagonal for arbitrary B, C. These requirements do not uniquely fix S, but the choice

$$S = \frac{1}{\sqrt{2}} \begin{pmatrix} I & I \\ I & -I \end{pmatrix} \tag{C.59}$$

gives

$$S^{-1}AS = \begin{pmatrix} B + C & 0 \\ 0 & B - C \end{pmatrix} \tag{C.60}$$

This choice is a convenient one, because then

$$S = S^{-1} = S^\dagger = S^T. \tag{C.61}$$

It should not come as a complete surprise that the columns of S are the eigenvectors of γ_5.

We can use this to reduce the computational cost of multiplying a 12-component column vector by A, thus:

$$\begin{aligned}
& A \begin{pmatrix} u \\ v \end{pmatrix} \\
= \; & \frac{1}{2} \begin{pmatrix} I & I \\ I & -I \end{pmatrix} \begin{pmatrix} B+C & 0 \\ 0 & B-C \end{pmatrix} \begin{pmatrix} I & I \\ I & -I \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} \\
= \; & \frac{1}{2} \begin{pmatrix} x+y \\ x-y \end{pmatrix}
\end{aligned}$$

where

$$\begin{aligned}
x &= (B+C)(u+v) \\
y &= (B-C)(u-v)
\end{aligned}$$

Instead of one 12×12 matrix-vector multiply, we now have two 6×6 matrix-vector multiplies, plus a small overhead of two 6-component vector additions.

The same trick can be used to reduce the computational cost of multiplication by the inverse of the clover term.

$$\begin{aligned}
& A^{-1} \begin{pmatrix} u \\ v \end{pmatrix} \\
= \; & \frac{1}{2} \begin{pmatrix} I & I \\ I & -I \end{pmatrix} \begin{pmatrix} (B+C)^{-1} & 0 \\ 0 & (B-C)^{-1} \end{pmatrix} \begin{pmatrix} I & I \\ I & -I \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} \\
= \; & \frac{1}{2} \begin{pmatrix} x+y \\ x-y \end{pmatrix}
\end{aligned}$$

where now

$$\begin{aligned}
x &= (B+C)^{-1}(u+v) \\
y &= (B-C)^{-1}(u-v)
\end{aligned}$$

In both the GHMC and solver codes, it is necessary to do many multiplications by $A^{-1} = A^{-1}[U, \kappa, C_{SW}]$ during which U, κ and C_{SW} remain constant. In practice, we first compute the $L^\dagger DL$ decompositions of $B + C$ and $B - C$ and use these to implement multiplication by their inverses. Note that

- L is lower triangular with ones on the diagonal, and D is diagonal
- B and C are hermitian so the decompositions exist
- $L^\dagger DL$ works just as well as the more usual $LDL^\dagger$
- it is just as efficient and generally more accurate to compute $\left(L^\dagger DL\right)^{-1} y$ by solving $L^\dagger DLx = y$ (this requires only a back substitution, scaling, and forward substitution). See the discussion in *Numerical Recipes* [78, section 2.3] on the related topic of the LU decomposition.

The trick reduces the memory required to store the decomposition by a factor of 2, and the operation count for multiplication by A^{-1} by a factor slightly better than 2. The operation count for computing the decomposition is reduced by a factor of about 4.

Appendix D

GHMC run-time options

The GHMC code reads its run time options from a text file carrying the suffix *.eri*. Every stored gauge configuration is accompanied by a file carrying the suffix *.par*; this file has the same format as the *.eri* file, and contains the complete set of run-time options to resume the GHMC simulation from the gauge configuration it accompanies.

Options are specified by keyword and value pairs, the value being separated from the keyword by one or more spaces (tabs are *not* equivalent to spaces and should be avoided). An exclamation point at the start of a line introduces a comment, and trailing comments, introduced by an exclamation point following the value string, are also allowed. The order in which options appear in the parameter file is insignificant, except that if an option is specified twice it is the *first* instance that survives.

The allowed keywords and their meanings are specified overleaf.

latt[X] Size of lattice in X-dimension.

latt[Y] Size of lattice in Y-dimension.

latt[Z] Size of lattice in Z-dimension.

latt[T] Size of lattice in T-dimension.

start_type Allowed values are:

> **0** old: load in an old gauge configuration
>
> **1** cold: set all gauge links to the identity matrix
>
> **2** hot: start from a random gauge configuration

start_sweep Trajectory number to start from.

rng_seed Positive integer used to re-seed the random number generator. If zero, the state of the random number generator (RNG) will be loaded from the *.rng* file associated with the gauge configuration. As the state of the RNG depends on the processor grid, it is necessary to re-seed when the number of processors is changed. *rng_seed* must be non-zero for cold or hot starts.

bc[X] Fermionic boundary condition in X-direction, default: 'periodic'.

bc[Y] Fermionic boundary condition in Y-direction, default: 'periodic'.

bc[Z] Fermionic boundary condition in Z-direction, default: 'periodic'.

bc[T] Fermionic boundary condition in T-direction, default: 'antiperiodic'.

calc_par Parity, 0 (even) or 1 (odd), on which pseudo-fermions are defined.

beta The simulation parameter β.

kappa Sea κ.

clover C_{SW}.

beta_shift Default 0. Not yet implemented. The guidance Hamiltonian for the molecular dynamics can in principle use a different value $\beta' = \beta + \delta\beta$; *beta_shift* would specify $\delta\beta$.

kappa_shift Default 0. Not yet implemented. As *beta_shift*.

clover_shift Default 0. Not yet implemented. As *beta_shift*.

mixing_parameter $\frac{2\theta}{\pi}$ where θ is the GHMC mixing angle (see §(6.2)). Pure HMC is specified by the value 1.0.

timestep The size of the molecular dynamics timestep $d\tau$.

sweep_length The trajectory length N_{md}.

gauge_updates $n_{\mathcal{G}}$, usually 1 or 2. See §(6.3.2).

num_sweeps The number of trajectories to perform this time.

save_int Determines the frequency at which gauge configurations etc. are to be saved to disk. The GHMC code saves data for future analysis whenever the remainder on dividing the trajectory number by *save_int* is zero. A zero value of *save_int* is defined to mean "never save".

checkpoint_int Determines the frequency at which checkpoints are to be taken, by the same recipe as for *save_int*. The value 0 disables checkpointing.

checkpoint_cycle_len The number of checkpoints in a checkpoint cycle. Should be at least 2, but there is no compelling reason for more than 2.

next_checkpoint_num The number of the next checkpoint to write. Should not exceed *checkpoint_cycle_len*.

time_limit_in_mins The GHMC code will take a checkpoint and stop if starting the next trajectory is likely to cause the elapsed run time to exceed *time_limit_in_mins*. This feature is especially useful when jobs are submitted through a queueing system which imposes a time limit on jobs.

load_momenta Determines whether the conjugate momenta should be loaded from disk. Allowed values are 'yes' and 'no'. If resuming from an 'old' configuration, always set *load_momenta* to 'yes'. Even in pure HMC, the old conjugate momenta are required to ensure reproducibility.

solver_type Inversion algorithm to use. Allowed values are:

- **'cg'** Conjugate Gradient. This choice implies a 1-step solve.
- **'bicgstab'** BiCGSTAB. This choice implies a 2-step solve.

restart_solver_type Should always be 'cg' at present.

guess_strategy_x Strategy to employ when choosing the initial guess $X^{(0)}$. Allowed values are:

- **zero** $X^{(0)} = 0$ (recommended).
- **noise** $X^{(0)} = \eta$ (dubious).
- **source** $X^{(0)} = \phi$ (not recommended).
- **last** The solution from the previous timestep is taken as the initial guess. Faster than *zero*, but manifestly non-reversible and not advised.

guess_strategy_y Strategy to employ when choosing the initial guess $Y^{(0)}$. Allowed values are as for *guess_strategy_x*.

restarts Number of restarts. 2 is recommended.

min_iter Minimum number of solver iterations.

max_iter Maximum number of solver iterations. Set *max_iter* to about 50% more than the average number of iterations required for convergence at this *kappa* and *solver_type*.

print_freq Set this to 0 unless you really want to monitor the size of the residuum in every solve. If *print_freq* $= n > 0$, the solver will report the value of the residuum $||r^{(k)}||$ whenever $\mathrm{mod}(k, n) = 0$, k being the iteration number.

omega Over-relaxation parameter, only used for MR. Default: 1.1. This can be tuned to accelerate convergence. A reasonable range is $1.05 < \omega < 1.4$. Not currently used (MR is not currently implemented in the GHMC code).

target_residue The stopping criterion for the exterior (first and last) solves in each trajectory. The algorithm is deemed to have converged on the solution (Y or X) when the equation residuum is smaller than *target_residue*. As the exterior solves control the accuracy of the energy calculation, I advise against any attempts at economy here. Use 0.1E-06 in 32-bit precision and don't be afraid to ask for more accuracy (say 0.1E-09) in 64-bit precision. See the discussion in §(6.5.7).

relax_md_residue_by The convergence criteria for the interior solves (during the molecular dynamics) may be relaxed with respect to the exterior solves. See the discussion in §(6.5.8). The target residue for the interior solves is given by *target_residue* × *relax_md_residue_by*.

precondition Allowed values are 'yes' and 'no'. If 'yes', the solver will use the block Jacobi preconditioning described in §(6.4), except on the last restart.

input_gauge_size Allowed values: 4, 8. Size, in bytes, of each floating point number in the input files containing the gauge configuration and their conjugate momenta.

byte_swap_input_gauge Allowed values are 'no', 'yes'. This option determines whether any special endianness conversion is to be done when loading gauge configurations (and their conjugate momenta) from disk.

output_gauge_size Allowed values: 4, 8. Size, in bytes, of each floating point number in the output files containing gauge configurations and their conjugate momenta.

byte_swap_output_gauge Allowed values are 'no', 'yes'. This option determines whether any special endianness conversion is to be done when saving gauge configurations (and their conjugate momenta) to disk.

output_gauge_path Name of the directory to contain the gauge configurations stored by the GHMC code for subsequent analysis.

output_fe_path Name of the directory to contain the *.par* and *.rng* files associated with stored gauge configurations.

output_mom_path As *output_gauge_path* but for conjugate momenta.

input_gauge_path Name of the directory containing the old gauge configuration, if any, to be loaded from disk.

input_fe_path Name of the directory containing the *.par* and *.rng* files associated with the input gauge configuration, if any.

input_mom_path As *input_gauge_path* but for conjugate momenta.

checkpoint_path Name of the directory used for saving checkpoint information.

input_stem The root file names for the input gauge configuration and conjugate momenta are constructed by appending predefined strings to *input_stem*.

output_prefix The root file names for the output gauge configuration and conjugate momenta are constructed by appending predefined strings to *output_prefix*. *input_stem* contains the trajectory number but *output_prefix* does not.

validate_plaquettes Allowed values are 'yes' and 'no'. If 'yes', the average plaquette of the gauge configuration loaded from disk will be computed and compared with the values specified in

plaquette_real real part of average plaquette

plaquette_imag imaginary part of average plaquette

The program will abort if the values do not match. These numbers are computed when a gauge configuration is generated, and stored in *.edi* files by the pure gauge code (quenched) or *.par* files by the Hybrid Monte Carlo code (usually dynamical). This option, and the options below, provide a useful check on the integrity of a gauge configuration.

validate_tplaquettes Allowed values are 'yes' and 'no'. As *validate_plaquettes*, but the average is taken over each time-slice separately. Values are specified in

tplaquette_real[t]

tplaquette_imag[t]

where t denotes the time-slice, $0 < t <$ *latt[T]* -1.

validate_tcsum Allowed values are 'yes' and 'no'. If 'yes', a checksum will be computed for each time-slice of the gauge configuration as it is loaded, and compared against values specified in

tcsum[t] An integer, $0 < t <$ *latt[T]* -1.

plaquette_real See *validate_plaquettes*

plaquette_imag See *validate_plaquettes*

tplaquette_real[t] See *validate_tplaquettes.*

tplaquette_imag[t] See *validate_tplaquettes.*

tcsum[t] See *validate_tcsum.*

Appendix E

Glossary

Block algorithm An iterative method for solving linear systems with *multiple* right-hand sides simultaneously. Contrast *point algorithm.*

BLAS Basic Linear Algebra Subroutines. A library of subroutines for performing basic vector-vector and matrix-vector operations. Available on a wide variety of platforms, and often highly optimised for the target architecture.

Condition number The condition number (with respect to inversion), cond (A) of a square matrix A is given by

$$\mathrm{cond}\,(A) = \|A\| \|A^{-1}\|.$$

For a common choice of norm,

$$\mathrm{cond}\,(A) = \frac{\max\limits_{\lambda} |\lambda|}{\min\limits_{\lambda} |\lambda|}$$

cond (A) provides a crude measure of the difficulty of solving a linear system of the form

$$Ax = b,$$

the larger the condition number, the more difficult the problem.

Dublin plot A plot of $1/\kappa$ (or m_q) as a function of β. Useful for discussing continuum extrapolations at constant a.

Krylov subspace The k-th Krylov subspace generated by a matrix A and vector

v is given by

$$K_k(A, v) = \text{span}\{v, Av, \ldots, A^{k-1}v\}$$

Krylov subspace method An iterative method for solving a linear system

$$Ax = b$$

which utilises a sequence of Krylov subspaces

$$\left\{K_k\left(A, b - Ax^{(0)}\right)\right\},$$

in order to construct successive approximations $x^{(k)}$ to $x = A^{-1}b$.

LAPACK LAPACK (Linear Algebra Package) is a library of linear algebra and eigenproblem solvers. It is available on a wide variety of platforms, and supersedes the earlier packages LINPACK and EISPACK.

Machine epsilon The machine epsilon ϵ is the smallest number such that the floating point representation of $1.0 + \epsilon$ is distinguishable from 1.0. ϵ is typically $O(10^{-7})$ in 32-bit precision and $O(10^{-15})$ in 64-bit precision.

Point algorithm Any iterative method for solving linear systems with a *single* right-hand side. Contrast *block algorithm.*

QR decomposition The QR decomposition of a matrix A is the factorisation $A = QR$, where R is upper triangular and Q satisfies $Q^{\dagger}Q = I$.

restart Having obtained an approximate solution to the matrix equation $Ax = b$ using an iterative method, it is sometimes useful to *restart.* The solution is repeated using the latest approximation, $x^{(k)}$ say, as the initial guess $x^{(0)}$. The algorithm used in the restart need not necessarily be the same as in the original solve. Usually the restart will converge in much fewer iterations than the original solve, therefore yielding a more reliable final residual.

saxpy A subroutine in the BLAS library which performs the operation

$$y = ax + y$$

for real vectors x, y and real scalar a. More generally, the name given to the class of vector-vector operations of this form, for any vectors x and y and any scalar a.

Sparse matrix A matrix $A = (a_{ij})$ for which most of the components a_{ij} are zero. There is no clear division between "sparse" and "dense", but a good operational definition is that a matrix is sparse if it pays to exploit the zeroes. The fermion matrix becomes more sparse as the lattice volume is increased.

References

[1] M. Creutz. Quarks, Gluons and Lattices. Cambridge University Press, Cambridge, 1983.

[2] Heinz J. Rothe. Lattice Gauge Theories: An Introduction. World Scientific, Singapore, 1992.

[3] I. Montvay and G. Münster. Quantum Fields on a Lattice. Cambridge University Press, Cambridge, 1994.

[4] Zbigniew Sroczynski. Taking Lattice QCD Beyond the Quenched Approximation. PhD thesis, University of Edinburgh, 1998.

[5] UKQCD Collaboration. Light hadron spectroscopy with $O(a)$ improved dynamical fermions. *in preparation*, (1998).

[6] K. G. Wilson. Confinement of quarks. *Phys. Rev. D*, **10**, 2445–59, (1974).

[7] B. Sheikholeslami and R. Wohlert. Improved continuum limit lattice action for QCD. *Nucl. Phys. B*, **259**, 572–596, (1985).

[8] G. Heatlie et al. The improvement of hadronic matrix elements in lattice QCD. *Nucl. Phys. B*, **352**, 266–288, (1991).

[9] M. Lüscher, S. Sint, R. Sommer and P. Weisz. Chiral symmetry and $O(a)$ improvement. *Nucl. Phys. B*, **478**, 365–400, (1996).

[10] M. Creutz. Evaluating grassmann integrals. hep-lat/9806037, 1998.

[11] M. Creutz. Grassmann integrals by machine. hep-lat/9804021, 1998. Presented at 16th International Symposium on Lattice Field Theory (LATTICE 98), Boulder, CO, 13-18 Jul 1998.

[12] C. R. Gattringer, I. Hip, C. B. Lang. Topological Charge and the Spectrum of the Fermion Matrix in Lattice-QED_2. *Nucl. Phys. B*, **508**, 329–352, (1997).

[13] S. Itoh, Y. Iwasaki and T. Yoshié. U(1) problem and topological excitations on a lattice. *Phys. Rev. D*, **36**, 527–45, (1987).

[14] J. Smit and J. Vink. Remnants of the index theorem on the lattice. *Nucl. Phys. B*, **286**, 485–508, (1987).

[15] J. Smit and J. Vink. Topological charge and fermions in the two-dimensional lattice U(1) model (I). Staggered fermions. *Nucl. Phys. B*, **303**, 36–56, (1988).

[16] J. Vink. Topological charge and fermions in the two-dimensional lattice U(1) model (II). Wilson fermions. *Nucl. Phys. B*, **307**, 549–70, (1988).

[17] P. Hernández. On the Index Theorem for Wilson Fermions. hep-lat/9801035, 1998. CERN preprint no. CERN-TH/97-370.

[18] F. Chatelin. Eigenvalues of Matrices. Wiley, Baffins Lane, Chichester, 1993.

[19] G. H. Golub and C. F. Van Loan. Matrix Computations. John Hopkins University Press, Baltimore, Maryland, third edition, 1996.

[20] R. Barrett et. al. Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition. SIAM, Philadelphia, 1994.

[21] A. Frommer. Linear systems solvers – recent developments and implications for Lattice computations. *Nucl. Phys. B (Proc. Suppl.)*, **53**, 120–126, (1997). hep-lat/9608074 (1996).

[22] A. D. Simpson. Algorithms for Lattice QCD. PhD thesis, University of Edinburgh, 1991.

[23] N. P. Stanford. Portable lattice QCD software for massively parallel processor systems. PhD thesis, University of Edinburgh, 1994.

[24] R. Gupta et. al. QCD with dynamical Wilson fermions. *Phys. Rev. D*, **40**, 2072–84, (1989).

[25] A. Frommer et al. Many Masses on One Stroke: Economic Computation of Quark Propagators. *Int. J. Mod. Phys.*, **C6**, 627–638, (1995). Wuppertal University Preprint WUB 95-12, Technical Note HLRZ-95-20, hep-lat/9504020.

[26] U. Glässner. How to compute Green's Functions for entire Mass Trajectories within Krylov Solvers. hep-lat/9605008, 1996.

[27] B. Jegerlehner. Multiple Mass Solvers. *Nucl. Phys. B (Proc. Suppl.)*, (1997). hep-lat/9708029.

[28] S. Fischer et al. A Parallel SSOR Preconditioner for Lattice QCD. *Comput. Phys. Commun.*, **98**, 20–34, (1996). Wuppertal University Preprint WUB 96-1, HLRZ 4/96.

[29] S. M. Pickles. Solver code: a status report. available on the UKQCD collaboration's internal web pages, 1997.

[30] M. R. Hestenes and E. Stiefel. Methods of Conjugate Gradients for Solving Linear Systems. *J. Res. Nat. Bur. Stand.*, **49**, 409–36, (1952).

[31] V. Faber and T. Manteuffel. Necessary and sufficient conditions for the existence of a conjugate gradient method. *SIAM J. Numer. Anal.*, **21**, 352–362, (1984).

[32] A. Boriçi and Ph. de Forcrand. Fast Krylov Space Method for Calculation of Quark Propagator. hep-lat/9405001, 1994. IPS Research Report No. 94-03.

[33] C. Lanczos. Solution of systems of linear equations by minimized iterations. *J. Res. Natl. Bur. Stand.*, **49**, 33–53, (1952).

[34] R. Fletcher. Conjugate Gradient methods for indefinite systems. In G. Watson, editor, Numerical Analysis, pages 73–89. Springer-Verlag, Berlin, 1975.

[35] R. W. Freund. Lanczos-Type Algorithms for Structured Non-Hermitian Eigenvalue Problems. In D. Brown et al., editor, Proceedings of the Cornelius Lanczos International Centenary Conference, pages 243–245, Philadelphia, 1993. SIAM.

[36] M. Plagge. Investigation of the biconjugate gradient algorithm for the inversion of fermion matrices. Technical Report PRINT-93-0042, Münster University, Dec 1992. hep-lat/9212007.

[37] R. W. Freund and N. Nachtigal. QMR: a quasi-minimal residual method for non-Hermitian linear systems. *Numer. Math.*, **60**, 315–339, (1991).

[38] B. N. Parlett, D. R. Taylor and Z. A. Liu. A look-ahead Lanczos algorithm for unsymmetric matrices. *Math. Comp.*, **44**, 105–124, (1985).

[39] P. Sonneveld. CGS, a fast Lanczos-type solver for nonsymmetric linear systems. *SIAM J. Sci Statist. Comput.*, **10**, 36–52, (1989).

[40] H. A. Van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Statist Comput.*, **13**, 631–644, (1992).

[41] M. H. Gutknecht. Variants of Bicgstab for Matrices with Complex Spectrum. *SIAM J. Sci. Comput.*, **14**, 1020–1033, (1993).

[42] A. Frommer et al. Accelerating Wilson Fermion Matrix Inversions by Means of the Stabilized Biconjugate Gradient Algorithm. *Int. J. Mod. Phys.*, **C5**, 1073–1088, (1994). Wuppertal University Preprint WUB 94-10, hep-lat/9404013.

[43] G. Cella et. al. Efficiency of different matrix inversion methods applied to Wilson fermions. *Int. J. Mod. Phys. C*, **7**, 787–809, (1996). hep-lat/9606003.

[44] P. A. Rowland. Light Hadron Spectroscopy in Quenched QCD. PhD thesis, University of Edinburgh, 1994.

[45] R. Frezzotti and K. Jansen. A Polynomial Hybrid Monte Carlo Algorithm. *Phys. Lett. B*, **402**, 328–334, (1997).

[46] R. Frezzotti and K. Jansen. Experiences with the Polynomial Hybrid Monte Carlo Algorithm. In C. T. H. Davies et. al., editor, Nucl. Phys. B (Proc. Suppl.), volume 63, pages 943–945, 1998. hep-lat/9709033. Talk given by R.F. at the International Symposium on Lattice Field Theory, 22-26 July 1997, Edinburgh, Scotland.

[47] T. Kalkreuter and H. Simma. An Accelerated Conjugate Gradient Algorithm to Compute Low-Lying Eigenvalues — a Study for the Dirac Operator in SU(2) Lattice QCD. *Comput. Phys. Commun.*, **93**, 33–47, (1996).

[48] D. Smith, H. Simma, M. Teper (UKQCD Collaboration). Topological structure of the SU(3) vacuum and exceptional eigenmodes of the improved Wilson-Dirac operator. In C. T. H. Davies et. al., editor, Nucl. Phys. B (Proc. Suppl.), volume 63, pages 558–560, 1998. hep-lat/9709128. Talk given by D.S. at the International Symposium on Lattice Field Theory, 22-26 July 1997, Edinburgh, Scotland.

[49] H. Simma, D. Smith (UKQCD collaboration). Low-lying Eigenvalues of the improved Wilson-Dirac Operator in QCD. hep-lat/9801025, 1998.

[50] J. F. McCarthy. The Block Conjugate Gradient Method. *Phys. Rev. D*, **40**, 2149–2152, (1989). Indiana University Preprint IUHET 160 (1989).

[51] D. Henty, R. Setoodeh and C. T. H. Davies. A Study of Block Algorithms for Fermion Matrix Inversion. *Nucl. Phys. B*, **337**, 487–508, (1990).

[52] P. D. O'Leary. The Block Conjugate Gradient Algorithm and Related Methods. *Lin. Algebra Appl.*, **29**, 293–322, (1980).

[53] W. E. Boyse and A. A. Seidl. A Block QMR Method for Computing Multiple Simultaneous Solutions to Complex Symmetric Systems. *SIAM J. Sci. Comput.*, **17**, 263–274, (1996).

[54] R. W. Freund and M. Malhotra. A Block QMR Algorithm for Non-Hermitian Linear Systems with Multiple Right-Hand Sides. *AT&T Bell Laboratories, Numerical Analysis Manuscript No. 95–09*, (1995). Available from `http://cm.bell-labs.com/cs/doc/95`.

[55] P. Fiebach, A. Frommer and R. W. Freund. Variants of the Block-QMR Method and Applications in Quantum Chromodynamics. *AT&T Bell Laboratories, Numerical Analysis Manuscript No. 97–3–01*, (1997). Available from `http://cm.bell-labs.com/cs/doc/97`.

[56] A. D. Kennedy, R. Edwards, H. Mino and B. Pendleton. Tuning the generalized Hybrid Monte Carlo algorithm. *Nucl. Phys. B (Proc. Suppl.)*, **47**, 781–784, (1996).

[57] Z. Sroczynski, S. M. Pickles and S. P. Booth. UKQCD Dynamical Fermions Project, 1997.

[58] S. Duane, A. D. Kennedy, B. J. Pendleton and D. Roweth. Hybrid Monte Carlo. *Phys. Lett. B*, **195**, 216–222, (1987).

[59] A. Horowitz. The second order Langevin equation and numerical simulations. *Nucl. Phys.*, **B280[FS18](3)**, 510–522, (1987).

[60] A. Horowitz. A generalized guided Monte Carlo algorithm. *Phys. Lett. B*, **268**, 247–252, (1991).

[61] Gottlieb et al. Hybrid-molecular-dynamics algorithms for the numerical simulation of quantum chromodynamics. *Phys. Rev. D.*, **35**, 2531–2542, (1987).

[62] J. C. Sexton and D. H. Weingarten. Hamiltonian evolution for the hybrid Monte Carlo algorithm. *Nucl. Phys. B*, **380**, 665–677, (1992).

[63] R. Brower et al. Chronological Inversion Method for the Dirac Matrix in Hybrid Monte Carlo. *Nucl. Phys. B*, **484**, 353–374, (1997). hep-lat/9509012v2 (1996).

[64] A. D. Kennedy. Exact exponential mapping for su(3). private communication, 1990.

[65] M. Göckeler et. al. First results with non-perturbative fermion improvement. In Nucl. Phys. B (Proc. Suppl.), volume 53, pages 312–314, 1997. hep-lat/9608081. Talk given by P. Stephenson at the International Symposium on Lattice Field Theory, 1996, St. Louis.

[66] R. G. Edwards, I. Horváth and A. D. Kennedy. Non-Reversibility of Molecular Dynamics Trajectories. *Nucl. Phys. B*, **484**, 375–402, (1997).

[67] S. J. Dong and K. F. Liu. Stochastic Estimation with Z_2 Noise. *Phys. Lett. B*, **328**, 130–136, (1994). hep-lat/9308015.

[68] G. Ritzenhöfer. Flavour Singlet Operators: A First Calculation of Quark-Loop Insertions in Full QCD. PhD thesis, Wuppertal University, 1997.

[69] B. Efron. The jacknife, the bootstrap and other resampling plans. *Society for Industrial and Applied Mathematics*, (1982).

[70] G. E. P. Box and G. M. Jenkins. Time Series Analysis: Forecasting and Control. Prentice Hall, New Jersey, third edition, 1994.

[71] P. J. Brockwell and R. A. Davis. Time Series: Theory and Methods. Springer-Verlag, New York, second edition, 1991.

[72] SESAM Collaboration. QCD with dynamical Wilson fermions — first results from SESAM. In Nucl. Phys. B (Proc. Suppl.), volume 47, pages 386–393, 1996. hep-lat/9510001. Contributions by Th.Lippert and H.Hoeber to LAT95, Melbourne, July 1995.

[73] Th. Lippert et al. SESAM and TXL Results for Wilson Action — A Status Report. In Nucl. Phys. Proc. Suppl., volume 60A, pages 311–334, 1998. hep-lat/9707004.

[74] J. Cullum and R. A. Willoughby. Computing eigenvalues of very large symmetric matrices – an implementation of a Lanczos algorithm with no reorthogonalisation. *J. Comp. Phys*, **44**, 329, (1981).

[75] I. M. Barbour et al. The Lanczos Method in Lattice Gauge Theories. In The Recursion Method and Its Applications, pages 149–164. Springer-Verlag, 1985.

[76] R. Setoodeh, C. T. H. Davies and I. M. Barbour. Wilson fermions on the lattice – a study of the eigenvalue spectrum. *Phys. Lett. B*, **213**, 195–202, (1988).

[77] T. Kalkreuter. Study of Cullum's and Willoughby's Lanczos method for Wilson fermions. *Nucl. Phys. B (Proc. Suppl.)*, **49**, 168–173, (1996). hep-lat/9509071.

[78] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery. Numerical Recipes in Fortran. Cambridge University Press, Cambridge, second edition, 1992.

[79] W. Bardeen et. al. Light Quarks, Zero Modes, and Exceptional Configurations. *Phys. Rev. D*, **57**, 1633–1641, (1998). hep-lat/9705008.

[80] C. Itzykson and J. Zuber. Quantum Field Theory. McGraw-Hill, New York, International Edition edition, 1985.

List of Tables

List of Figures

Index