About the inversion of the Dirac operator

Version 1.3

Abstract

In this note we would like to specify a bit what is done when we invert the Dirac operator in lattice QCD.
Contents

1 Introductive remark .................................................. 2

2 Notations .................................................................. 3
   2.1 The gauge fields and gauge configurations .............. 4
   2.2 General properties of the Dirac operator matrix ....... 5
   2.3 Wilson-twisted Dirac operator .............................. 6

3 Even-odd preconditionning .................................. 9
   3.1 implementation of even-odd preconditionning ............ 10
      3.1.1 Conjugate Gradient ................................... 11

4 Inexact deflation ..................................................... 15
   4.1 Deflation method .................................................. 15
   4.2 Deflation applied to the Dirac Operator .................. 16
   4.3 Deflation FAPP ..................................................... 16
   4.4 Deflation : theory ................................................ 18
      4.4.1 Some properties of $P_{p}$ and $P_{d}$ ................. 18
      4.4.2 The Little Dirac Operator ............................. 19

5 Mathematic tools .................................................... 20

6 Samples from ETMC codes .................................. 22
Chapter 1

Introductive remark

The inversion of the Dirac operator is an important step during the building of a statistical sample of gauge configurations: indeed in the HMC algorithm it appears in the expression of what is called "the fermionic force" used to update the momenta associated with the gauge fields along a trajectory. It is the most expensive part in overall computation time of a lattice simulation with dynamical fermions because it is done many times.

Actually the computation of the quark propagator, i.e. the inverse of the Dirac operator, is already necessary to compute a simple 2pts correlation function from which one extracts a hadron mass or its decay constant: indeed that correlation function is roughly the trace (in the matricial language) of the product of 2 such propagators.
We are interested in the study of strong interaction, one of the 4 forces governing the Universe. In the Standard Model, the matter fields sensitive to the strong interaction are quarks; those particles have a spin 1/2. They are described by "Dirac spinors" i.e. 4-D complex vectors. The space in which The Dirac operator is written is a vector space which is the tensor product of three (six) vector spaces.

- The "spin space", $S$, which is a 4-dimension complex vector space. The vector of this space are called "spinors". In this space are defined "gamma matrices" which are constant 4*4 hermitian matrices. There are 4 gamma matrices corresponding to the four space-time directions: $\gamma_x, \gamma_y, \gamma_z, \gamma_t$ often written in the same order as $\gamma_1, \gamma_2, \gamma_3, \gamma_4$. $\gamma_4$ is sometimes named $\gamma_0$. A $\gamma_5$ is also used. They verify the properties

$$\gamma^2 = I_{4x4}, \quad \gamma_\mu \gamma_\nu + \gamma_\nu \gamma_\mu = 2\delta_{\mu\nu} I_{4x4} \quad (2.1)$$

for $\mu, \nu = 1, 4$ where $I_{4x4}$ is the identity 4*4 matrix.

- The "color space", $C$, which is a 3-dimension complex vector space. In this space are defined the gluon field matrices (or "gauge matrices") $U$ which are $SU(3)$ matrices id est special unitary 3*3 matrices. One such $U$ matrix is defined on every link of the lattice. Contrary to the gamma matrices these $U$ matrices change from link to link and many times during the runs. The Dirac operator in this "color" space is governed by these $U$ matrices as we shall see.

- The lattice space, $V$ of dimension $N$ (the number of lattice sites). This space itself is a tensor product of the $X$ space of dimension $L_x$ (length of the lattice in the direction $x$), $Y$ space of dimension $L_y$ (length of the lattice in the direction $y$), $Z$ (dimension $L_z$) and $T$ space (dimension $L_t$). It results that $N$ is the product

$$N = L_x L_y L_z L_t \quad (2.2)$$
We may denote by $p$ a generic site (or point) on the lattice of coordinates $(x, y, z, t)$. All these linear spaces are periodic, i.e., $x$ ranges from 1 to $L_x$ but $L_x + 1 = 1$.

**The full vector space** To summarize the total vector space in which the Dirac operator acts is

$$W = S \otimes C \otimes X \otimes Y \otimes Z \otimes T$$

(2.3)

of dimension $12N^4$. Vectors of this space will be called quark fields (sometimes one calls them "Wilson vectors"). In the following we will note the quark fields with the Greek letters $\phi, \psi, \omega, ...$ Physicists use to write them $\psi_\alpha^a(p)$ but we may as well to meet the request of our colleagues write them as $\psi(a, \alpha, x, y, z, t)$ (2.4)

where $a = 1, 3$ labels the color space (we will always use latin letters $a, b, c$ for color), $\alpha = 1, 4$ labels the spin space (we will always use greek letters $\alpha, \beta, ..., \gamma$ for spin) and $x = 1, L_x, y = 1, L_y$ etc label the site $p$ on the lattice. The Dirac operator is a $12N \times 12N$ matrix in the full vector space. It will be written in general as

$$D(a, \alpha, x, y, z, t; a', \alpha', x', y', z', t')$$

(2.5)

Or, in short-hand notations, $D$ will represent the $12N \times 12N$ matrix in the full vector space. We will give a more detailed description of it below. For practical use we now define in the full vector space

$$I_{12N \times 12N}, \quad \text{and} \quad \gamma_{12N \times 12N}$$

(2.6)

which are the identity matrix in the full space and the the $\gamma$ matrix in the spin state times the identity in all other components of the tensor product Eq. (2.3).

**2.1 The gauge fields and gauge configurations**

Physicists usually denote the matrices by $U_\mu(p)$ for the matrix which is on the link starting from the site $p$ in the direction $\mu$ ($\mu = x$ or $\mu = y$ or $z$ or $t$). In other words $U_\mu(p)$ is on the link between the point $p$ of coordinate $(x, y, z, t)$ and the point $p'$, $(x + 1, y, z, t)$. We could also write the matrix $U(p')$ as $U(p', p)$. The links are oriented. By definition

$$U(p', p) \equiv U^\dagger(p, p')$$

(2.7)

In the case of twisted quarks we write together the $u$ and $d$ quark into the isospin space. This leads in fact to a twice larger dimension of the vector space, $24 \times N$.

In the case of twisted quarks it is a $24 \times N \times 24N$ matrix.
We call "gauge configuration" a generic function which, to every link of the lattice, associates one $SU(3)$ matrix according to the rule Eq. (2.7). The first major step of lattice calculations is to generate a large sample of gauge configurations according to a well defined probability law. In the problem of the inversion the gauge configuration is given and does not change all along the computation. It has to be stored. Its size in bytes if $9 \times 16 \times 4 \times N$.

About 400kB for a lattice $L_x = L_y = L_z = 24, L_t = 48$.

2.2 General properties of the Dirac operator matrix

There are several forms of the Dirac operator on the lattice. All have their good and bad points, but all point towards the same limit in the continuum, i.e. when the lattice spacing vanishes. To mention the most popular ones: Wilson with the best variants called "Wilson-clover" and "Wilson-twisted" (the one used by ETMC collaboration); Staggered; overlap; domain-wall. Let us consider the Wilson class. The Dirac matrix is sparse. Indeed, $D(a, \alpha, x, y, z, t; a', \alpha', x', y', z', t') = 0$ except in nine cases:

- Same sites: $x = x', y = y', z = z', t = t'$
- Nearest neighbours sites or hopping terms:
  - $x = x' \pm 1, y = y', z = z', t = t'$,
  - $x = x', y = y' \pm 1, z = z', t = t'$, $z' \pm 1, t = t'$ and $x = x', y = y', z = z', t = t' \pm 1$.

The routine "Hopping Matrix" computes the action of these nearest neighbours terms. Example of the Wilson-twisted Dirac operator.

By "Inversion" we mean inverting partially this matrix i.e. solving a set of equations of the type

$$D \psi = \eta$$

(2.8)

where $D$ and $\eta$ are given and $\psi$ is unknown. $D$ is a $12N \times 12N$ complex matrix and $\eta$ and $\psi$ are $12N$ complex vectors. The solution is often called "quark propagator". Computing these is the major goal for the first year of petaQCD.
2.3 Wilson-twisted Dirac operator

In the case of Wilson-twisted the quarks are associated with doublets. We consider for example together the quarks "u" and "d". In this example, we assume that they have the same mass. The doublet implies that now the full space of the Dirac operator is a tensor product

$$W_t = I \otimes S \otimes C \otimes X \otimes Y \otimes Z \otimes T$$  \hfill (2.9)

where the additional space, $I$ is named "isospin" and has dimensions 2. The vector has two components, corresponding respectively to the $u$ and $d$ quark.

We will use only the identity matrix $I$ in isospin space and the matrix

$$\tau_3 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$  \hfill (2.10)

The upper 1 acts on $u$ quark the lower on $d$ quark. \hfill return to even-odd implementation

The full vector space is now of dimension $24N$. We write the Dirac operator as

$$D(i, a, \alpha, x, y, z, t; i', a', \alpha', x', y', z', t')$$  \hfill (2.11)

where the new indices $i, i' = 1, 2$ correspond to isospin. For the sake of simplicity we will write the Dirac operator by block of dimension $12N \times 12N$ each, the blocks corresponding to the isospin.

$$D = \begin{pmatrix} D^+ & 0 \\ 0 & D^- \end{pmatrix}$$  \hfill (2.12)

This matrix by block is diagonal because we are in a special case in which we use only the identity isospin matrix and the diagonal $\tau_3$, Eq. (2.10). In more complex cases the twisted Dirac operator contains also non-diagonal contributions in isospin space, but we will not consider them now. The Dirac operator $D^+$ ($D^-$) is the Dirac operator of the $u$ ($d$) quark.

Now we define all the non-zero matrix elements of the Dirac operator. We will write the identity matrix $I_C, I_S, I_x, I_y, I_z, I_t$ respectively for isospin, color, spin, $x, y, z, t$ spaces. It is the tensor product of all the preceding ones.

- same site: $x' = x, y' = y, z' = z, t' = t$. This term can be written as

$$D_{\text{same site}}^\pm = I_{12N \times 12N} \pm 2i\kappa\mu\gamma_5$$  \hfill (2.13)

where we have used \text{these notations} $\kappa$ is a parameter (typically between 0.1 and 0.2) of the run and $\mu$ is the quark mass in units of the lattice. The quark mass is a small number (a few $10^{-3}$) for light quarks. It can also be written as follows

$$D_{\text{same site}}^\pm(a, \alpha, x, y, z, t; i, a, \alpha', x, y, z, t) = \delta_{\alpha\alpha'} \pm 2i\kappa\mu(\gamma_5)_{\alpha\alpha'}$$  \hfill (2.14)
where the Kronecker symbol \( \delta_{\alpha\alpha'} \) vanishes until \( \alpha = \alpha' \) and where we write the matrix elements of \( \tau_1 \) and \( \gamma_5 \) as indices of the matrix name in the order line column. Notice that the color is trivial, identity in color space, and the spin is matrix \( \gamma_5 \) is the same on all sites. This makes this "same-site" part of Dirac operator rather trivial, which is used in the even-odd treatment. These terms can be found in the routine \text{tm\_operators.c}. See for example \text{mul\_one\_pm\_imu\_sub\_mul\_gamma5}. From now on we consider "hopping terms" i.e. interactions to the nearest neighbour. They are treated in the routine \text{Hopping\_Matrix.c}.

\begin{itemize}
  \item Case \( x' = x - 1 \).

  \[ D_{x' = x - 1}^{\pm} = -\kappa(I_{4 \times 4} + \gamma_x) \otimes U(p, p') \otimes I_y \otimes I_z \otimes I_t \]  

  (2.15)

  with the coordinates of \( p \) (\( p' \)) equal to \( x, y, z, t \) \((x' = x - 1, y, z, t)\). This can also be written

  \[ D_{x' = x - 1}^{\pm}(a, \alpha, x, y, z, t; i, a, \alpha', x - 1, y, z, t) = -\kappa(\delta_{\alpha\alpha'} + (\gamma_x)_{\alpha\alpha'})U(p, p')_{a,\alpha'} \]  

  (2.16)

  The color structure of this part is non trivial. It depends on the gauge configuration. In an inversion the \( U(p, p') \) do not change during the calculation but they are different from one couple \( p, p' \) to another \( q, q' \). See for instance the piece of code in \text{Hopping\_Matrix.c}.

  \item Case \( x' = x + 1 \).

  \[ D_{x' = x + 1}^{\pm} = -\kappa(I_{4 \times 4} - \gamma_x) \otimes U(p, p') \otimes I_y \otimes I_z \otimes I_t \]  

  (2.17)

  with the coordinates of \( p \) (\( p' \)) equal to \( x, y, z, t \) \((x' = x + 1, y, z, t)\). This can also be written

  \[ D_{x' = x + 1}^{\pm}(a, \alpha, x, y, z, t; i, a, \alpha', x + 1, y, z, t) = -\kappa(\delta_{\alpha\alpha'} - (\gamma_x)_{\alpha\alpha'})U(p, p')_{a,\alpha'} \]  

  (2.18)

  \item Case \( y' = y \pm 1 \)

  \[ D_{y' = y \pm 1}^{\pm}(a, \alpha, x, y, z, t; i, a, \alpha', x, y' = y \pm 1, z, t) = -\kappa(\delta_{\alpha\alpha'} \mp (\gamma_y)_{\alpha\alpha'})U(p, p')_{a,\alpha'} \]  

  (2.19)

  with the coordinates of \( p \) (\( p' \)) equal to \( x, y, z, t \) \((x, y = y \pm 1, z, t)\).

  \item Case \( z' = z \pm 1 \)

  \[ D_{z' = z \pm 1}^{\pm}(a, \alpha, x, y, z, t; i, a, \alpha', x, y, z' = z \pm 1, t) = -\kappa(\delta_{\alpha\alpha'} \mp (\gamma_z)_{\alpha\alpha'})U(p, p')_{a,\alpha'} \]  

  (2.20)

  with the coordinates of \( p \) (\( p' \)) equal to \( x, y, z, t \) \((x, y, z' = z \pm 1, t)\).
\end{itemize}
Case $t' = t \pm 1$

$$D_{t' = t \pm 1}^{\pm}(a, \alpha, x, y, z, t; i, a, \alpha', x, y, z, t' = t \pm 1) = -\kappa(\delta_{\alpha\alpha'} \mp (\gamma_{t})_{\alpha\alpha'})U(p, p')_{a, a'}$$

(2.21)

with the coordinates of $p$ ($p'$) equal to $x, y, z, t$ ($x, y, z, t' = t \pm 1$).

All other matrix elements of the Dirac operator are zero. This is why the Dirac operator is a sparse matrix. The number or matrix elements for one isospin is equal to $(4 \ast 3)^2 \ast N^2$. The number of non vanishing matrix elements is $588 \ast N = 12 \ast N + 8 \ast (4 + 4) \ast 9 \ast N$, where the first term is for the ”same site” and the second term is for the 8 directions, $4 + 4$ for the sum of two Dirac matrices times 9 for the generic $SU(3)$ matrix.

\textbf{return to general properties of Dirac operator}
Chapter 3

Even-odd preconditioning

The lattice sites can be classified according to the parity of $x + y + z + t$. The "same-site" matrix elements of the Dirac operator relate into the same parity. As already shown (see Wilson-twisted Dirac operator) they are very simple. The hopping terms relate sites of opposite parity. The complexity lies there. The vector space in which the Dirac operator acts has dimension $12N$. Separating the even and odd sites we have two subspaces of dimension $12N/2$. This is used to perform a preconditioning named "even-odd preconditioning. In the following we will write the Dirac operator by blocks according to this decomposition.

The general idea of preconditioning consists of multiplying both side of the initial system like eq. 2.8 by a preconditioning (regular) matrix $P$ to obtain a new system: $PD\psi = P\eta$ whose matrix $PD$ is supposed to have a smaller condition number $|\lambda_{\text{max}}/\lambda_{\text{min}}|$ than the original one $D$.

The Wilson action is such that only the nearest neighbours are concerned by the interaction (the matrix is sparse) and we can rewrite eq. 2.8 as (we will forget the color-spin indices from now)

\[
\begin{pmatrix}
D(ee) & D(eo) \\
D(oe) & D(oo)
\end{pmatrix}
\begin{pmatrix}
\psi(e) \\
\psi(o)
\end{pmatrix}
= 
\begin{pmatrix}
\phi(e) \\
\phi(o)
\end{pmatrix}
\] (3.1)

where e and o sets all the lattice sites which are "even" and "odd". The size of the subblocks is $12N/2 \times 12N/2$. For the Wilson and Wilson-twisted Dirac operators, $D(ee)$ and $D(oo)$ are diagonal in the volume space. Consequently those submatrices are very easy to invert. Multiplying both sides of Eq. 3.1 by the preconditioning matrix

\[
P = \begin{pmatrix}
D^{-1}(ee) & 0 \\
-D(oe)D^{-1}(ee) & 1
\end{pmatrix}
\] (3.2)
We get
\[
\begin{pmatrix}
I_{6N \times 6N} & D(ce)^{-1}D(eo) \\
0 & D(oo) - D(oe)D(ce)^{-1}D(eo)
\end{pmatrix}
\begin{pmatrix}
\psi(e) \\
\psi(o)
\end{pmatrix} = 
\begin{pmatrix}
D(ce)^{-1}\phi(e) \\
\phi(o) - D(oe)D(ce)^{-1}\phi(e)
\end{pmatrix}
\] (3.3)
This leads to Eq. (3.1) as
\[
\psi(e) = D^{-1}(ce)[\phi(e) - D(eo)\psi(o)]
\] (3.4)
simple to solve when \(D^{-1}(ce)\) is simple, and
\[
[D(oo) - D(oe)D^{-1}(ce)D(eo)]\psi(o) = \phi(o) - D(oe)D^{-1}(ce)\phi(e)
\] (3.5)
where the right-hand side is again easy to compute, but inverting the matrix on the left-hand side is non trivial. The auxiliary system to solve is then
\[
\hat{D}(oo)\psi(o) = \phi'(o)
\] (3.6)
where \(\hat{D}(oo) = D(oo) - D(oe)D^{-1}(ce)D(eo)\) and \(\phi'(o) = \phi(o) - D(oe)D^{-1}(ce)\phi(e)\).
We have diminished the size of the system to solve by 2. Note that it works only because \(D(ce)\) is diagonal in space for Wilson (and Wilson-twisted) fermions.

\textit{Note also that the even-odd preconditioning has a drawback, dixit Philippe: it does not allow “multisolvers” i.e. codes which solve different quark masses in one stroke. We now show a practical implementation of the even-odd preconditioning in the ETMC package.}

### 3.1 implementation of even-odd preconditioning

In fact, the Wilson-twisted Dirac operator in the full \(24N \times (24N)\) vector space is modified as follows:
\[
\begin{pmatrix}
Q^+ & 0 \\
0 & Q^-
\end{pmatrix} \equiv \begin{pmatrix}
\gamma_{12N\times 12N} & 0 \\
0 & \gamma_{12N\times 12N}
\end{pmatrix}
\begin{pmatrix}
D^+ \\
D^-
\end{pmatrix}
\] (3.7)
where we use the notations Eq. (2.6).

We use the fact that the matrices \(D\) and \(Q\) are diagonal by block \(\tau_3\) is diagonal in the isospin space \(I\). So the upper (lower) block of dimension \(12N \times (12N)\) corresponds to the \(u\) (\(d\)) quark. From Eq. (3.1) we get
\[
Q^\pm = \gamma_{12N\times 12N}
\begin{pmatrix}
D^\pm_{ee} & D_{eo} \\
D_{oo} & D^\pm_{oo}
\end{pmatrix}
\] (3.8)
\[
= \begin{pmatrix}
\gamma_{6N\times 6N}D^\pm_{ee} & 0 \\
\gamma_{6N\times 6N}D_{ee} & \gamma_{6N\times 6N}
\end{pmatrix}
\begin{pmatrix}
I_{6N\times 6N} & (D^\pm_{ee})^{-1}D_{eo} \\
0 & (D^\pm_{oo} - D_{ee}(D^\pm_{ee})^{-1}D_{eo})
\end{pmatrix}
\]
where the definition of $\gamma_{56N \times 6N}$ and $I_{6N \times 6N}$ are similar in the even/odd subspaces to that of $\gamma_{512N \times 12N}$ and $I_{12N \times 12N}$ in the full space and where we have multiplied $Q$ by a preconditioning matrix

$$
\begin{pmatrix}
P^+ & 0 \\
0 & P^-
\end{pmatrix} \gamma_{512N \times 12N}
$$

(3.9)

where $P^\pm$ is defined in Eq. (3.2) and where we use the fact that $\gamma_{56N \times 6N}$ commutes with $D_{ee}, D_{oe}, D_{eo}, D_{oo}$. One can check that

$$
\gamma_{56N \times 6N} \left( (D_{ee}^\pm)^{-1} - D_{oe}^{-1} (D_{ee}) \right) = \left( D_{ee}^\pm \right)^{-1} \gamma_{512N \times 12N}
$$

(3.10)

Using the fact that $D_{ee}^\pm = D_{oo}^\pm = I_{6N \times 6N}$ and that $(\gamma_{56N \times 6N})^2 = I_{6N \times 6N}$ one finds that the very simple result

$$
(D_{ee}^\pm)^{-1} = \frac{1}{1 + \mu^2} \left( I_{6N \times 6N} \mp i \gamma_{56N \times 6N} \right)
$$

(3.11)

We have to solve Eq. (3.1) which amounts to

$$
\hat{Q} \psi(\pm) = \gamma_{56N \times 6N} \left( \phi (\pm) - D_{oe} \left( D_{ee}^\pm \right)^{-1} \phi (\pm) \right)
$$

(3.12)

where

$$
\hat{Q} \psi(\pm) = \gamma_{56N \times 6N} \left( D_{oo}^\pm - D_{oe} \left( D_{ee}^\pm \right)^{-1} D_{eo} \right)
$$

$$
\psi(\pm) = \left( D_{ee}^\pm \right)^{-1} \left[ \gamma_{56N \times 6N} \phi (\pm) - D_{eo} \psi (\pm) \right]
$$

(3.13)

We have used the fact that $D_{oe}$ and $D_{eo}$ are identical for the $u$ (+) and $d$ (-) quark. Notice that $\hat{Q}$ is only defined on the odd sites. To solve Eq. (3.1) let us consider the Conjugate Gradient algorithm.

**3.1.1 Conjugate Gradient**

So, we have to solve systems like

$$
\hat{Q}^\pm X = \phi_0'
$$

(3.14)
However, the $\hat{Q}^\pm$ matrix does not have the needed proprieties to be solved by the Conjugate Gradient (CG) algorithm. We thus solve the equation

$$\hat{Q}^+ \hat{Q}^- X = \Phi_0, \quad \Phi_0 = \hat{Q}^- \phi_0$$

(3.17)

Since $\hat{Q}^+, \hat{Q}^-$ is Hermitian it can be made by conjugate gradient.

The general conjugate gradient algorithm (for a symmetric - or hermitian- positive definite matrix), for a system $Ax = b$ is the following. One can shown that the solution of the initial system minimize the bilinear functionnal $J(u) = \frac{1}{2}(Au, u) - (b, u)$ ($\langle .., \rangle$ denoting a scalar product). This strictly convex functionnal has suited properties, in order to characterize a unique solution of the minimization problem. Starting from any initial vector $x^0$, the solution can be written in terms of an orthogonal basis $(d^k)$ :

$$x - x^0 = \sum_{k=0}^{n-1} \alpha^k d^k$$

(3.18)

reporting into the functionnal :

$$J(x) = J(x^0) + \sum_{k=0}^{n-1} \alpha^k \langle Ax^0 - b, d^k \rangle + \frac{1}{2} \sum_{k=0}^{n-1} (\alpha^k)^2 \langle Ad^k, d^k \rangle$$

(3.19)

Writing the vanishing of the derivates $\frac{\partial J}{\partial \alpha^k}$, one get :

$$x = x^0 + \sum_{k=0}^{n-1} \alpha^k d^k = x^0 + \sum_{k=0}^{n-1} \frac{(b - Ax^0, d^k)}{(Ad^k, d^k)} d^k$$

(3.20)

From what the following algorithm can be deduced :

- initialization

$$r^0 = b - Ax^0$$

$$d^0 = r^0$$

(3.21)

- iterations

minimization step :

$$\alpha^k = \frac{\langle r^k, r^k \rangle}{\langle Ad^k, d^k \rangle}$$

$$x^{k+1} = x^k + \alpha^k d^k$$

$$r^{k+1} = r^k - \alpha^k Ad^k$$

(3.22)

computation of a new direction and orthogonalization :
\[
\begin{align*}
\beta^{k+1} &= \left( r^{k+1}, r^{k+1} \right) / \left( r^k, r^k \right) \\
d^{k+1} &= r^{k+1} + \beta^{k+1} d^k
\end{align*}
\] (3.23)

In HMC, the conjugate gradient (for hermitian matrices) is coded in the method cg::her (file /solver/cg::her.c). The product matrix vector is referenced by the pointer argument “matrix_mult f”. For instance, cg::her is called in invert::co.c in the following manner:

```c
iter = cg::her(Odd_new, g_spinor_field[DUM_DERI], max_iter, precision, rel_prec, VOLUME/2, &Qtm_pm_psi, sub_evs_flag, 1000);
```

The product matrix by vector is coded in this case by : Qtm_pm_psi. The code is the following:

```c
void Qtm_pm_psi(spinor * const l, spinor * const k){
/* Q_(-) */
  Hopping_Matrix(EO, g_spinor_field[DUM_MATRIX+1], k);
  mul_one_pm_imu_inv(g_spinor_field[DUM_MATRIX+1], -1.);
  Hopping_Matrix(OE, g_spinor_field[DUM_MATRIX], g_spinor_field[DUM_MATRIX+1]);
  mul_one_pm_imu_sub_mul_gamma5(g_spinor_field[DUM_MATRIX], k, g_spinor_field[DUM_MATRIX+1]);
/* Q_($) */
  Hopping_Matrix(EO, 1, g_spinor_field[DUM_MATRIX]);
  mul_one_pm_imu_inv(1, +1.);
  Hopping_Matrix(OE, g_spinor_field[DUM_MATRIX+1], 1);
  mul_one_pm_imu_sub_mul_gamma5(1, g_spinor_field[DUM_MATRIX], g_spinor_field[DUM_MATRIX+1]);
}
```

In the main loop of cg::her one can recognize the CG algorithm:

```c
/* main loop */
for(iteration=0; iteration<max_iter; iteration++){
  f(g_spinor_field[DUM_SOLVER+4], g_spinor_field[DUM_SOLVER+2]);
  if((subtract_ev == 1) && (iteration%modulo == 0)) {
    sub_lowest_eigenvalues(g_spinor_field[DUM_SOLVER+4], g_spinor_field[DUM_SOLVER+2],
  }
  c=scalar_prod(&g_ev[0*VOLUME], g_spinor_field[DUM_SOLVER+4]);
  printf("%e, %e\n", c.re, c.im); */
  pro=scalar_prod_r(g_spinor_field[DUM_SOLVER+2], g_spinor_field[DUM_SOLVER+4], N);
  /* Compute alpha_cg(i+1) */
  alpha_cg=normsq/pro;
  /* Compute x_+(i+1) = x_i + alpha_cg(i+1) p_i */
```
assign_add_mul_r(g_spinor_field[DUM_SOLVER], g_spinor_field[DUM_SOLVER+2], alpha_cg, /* Compute r_(i+1) = r_i - alpha_cg(i+1) Qp_i */
assign_add_mul_r(g_spinor_field[DUM_SOLVER+1], g_spinor_field[DUM_SOLVER+4], -alpha_cg, /* Check whether the precision is reached ... */
err=square_norm(g_spinor_field[DUM_SOLVER+1], N);
if(g_debug_level > 0 \&\& g_proc_id == g_stdio_proc) {
  printf("cg_her: %d\t% 23.16e\n",iteration,err); fflush( stdout);
}
if(((err <= eps_sq) \&\& (rel_prec == 0)) || ((err <= eps_sq*squarenorm) \&\& (rel_prec == 10)\nif((subtract_ev == 1)){
  assign_add_invert_subtracted_part(g_spinor_field[DUM_SOLVER], Q, 10, N);
}
assign(P, g_spinor_field[DUM_SOLVER], N);
f(g_spinor_field[DUM_SOLVER+2], P);
diff(g_spinor_field[DUM_SOLVER+3], g_spinor_field[DUM_SOLVER+2], Q, N);
err = square_norm(g_spinor_field[DUM_SOLVER+3], N);
if(g_debug_level > 0 \&\& g_proc_id == g_stdio_proc) {
  printf("cg_her: true residue %d\t% 23.16e\t\n",iteration, err); fflush( stdout);
}
g_sloppy_precision = 0;
return(iteration+1);
}

return to co implementation

14
Chapter 4

Inexact deflation

The deflation method is a general method to solve efficiently a linear system with eigenvalues of small magnitude. It is somehow a generalization of the preconditioning method.

Let $A$ be an invertible matrix, consider the linear system

$$Ax = b$$

(4.1)

the preconditioning method amounts finding an invertible matrix $P$ with the property that $PA$ is better conditioned than $A$ (i.e., the conditioning number of $PA$ is lower) and to solve

$$PAx = Pb$$

This is easy to propose, but finding $P$ is a big problem (I would suggest $P = A^{-1}$).

The deflation method uses a projector $P$ i.e., an operator satisfying $P^2 = P$ and therefore non-invertible (except for the identity) and then solve

$$\begin{cases} PAx = Pb \\ \overline{P}Ax = \overline{P}b \end{cases}$$

(4.2)

where $\overline{P} = 1 - P$. Each of these two linear systems has an infinite number of solutions, but it is still possible to reconstruct the solution of Eq. (4.1). Note that it is quite paradoxical that solving the two non-invertible systems (infinite condition number, see below) is more favorable than solving a single invertible system (finite condition number).

4.1 Deflation method

Again let $A$ be an invertible operator and $P_L$ a projector, $P_L^2 = P_L$ therefore $P_L$ is not invertible (except the identity), there exists a projector $P_R$ such that $P_LA = A P_R$. Indeed $P_R = A^{-1} P_L A$, one has $P_R^2 = A^{-1} P_L A A^{-1} P_L A = A^{-1} P_L^2 A = A^{-1} P_L A = P_R$. We note $\overline{P}_L = 1 - P_L$ and $\overline{P}_R = 1 - P_R$. 

15
Consider the system

\[ P_L Ax = Pb \]  

(4.3)

and let \( x_1 \) be a solution, take \( y \in \ker P_R \) (ie \( P_R y = 0 \))

\[ P_L A(x_1 + y) = P_L Ax_1 + P_L Ay \]

\[ = Pb + AP_R y \]

\[ = Pb \]

therefore \( x_1 + y \) is also a solution: the system Eq. (4.3) has an infinite number of solution.

Consider now \( x_1 \) a solution of \( P_L Ax = Pb \) then

\[ x = P_R x + P_R x_1 \]

is the solution of Eq. (4.4). Indeed

\[ Ax = AP_R x_1 + AP_R x_1 = P_R Ax_1 + P_R x_1 = P_R b + P_R b = b \]

All the difficulty is to find a smart projector \( P_L \). This is what Luscher did \cite{luscher}, and his paper is presented below.

### 4.2 Deflation applied to the Dirac Operator

What is presented below is just a rephrasing of the Luscher paper \cite{luscher}. First the practical aspect are presented, the justification comes after. The key point is a smart choice of the projector. The image of this projector is called the deflated subspace and the restriction of the Dirac operator the restricted Dirac operator. The restriction of the Dirac operator to the kernel of this projector is called the little Dirac operator.

### 4.3 Deflation FAPP

This paragraph should be enough to program the deflation algorithm as a recipe. The aim is to solve the inhomogeneous Dirac equation.

\[ D\psi = \eta \]

1. Divide the 4 dimensional lattice into \( N_b \) blocks,
2. Choose \( N_s \) random vectors \( \psi_{0,l} \) (component \textit{iid}d density ?) \cite{luscher},
3. compute in some way \( \psi_l = D^{-1}\psi_{0,l} \) (approximate and cheap inverse),

\[ ^1\text{In subroutine solver/generate/dfl_subspace.c in HMC/ETMC package, v5.0; random_fields(Ns) } \]
4. define the $N_s N_b$ vectors $\psi_l$ the restriction of $\psi_l$ to the $b^{th}$ block (all components for sites not in the $b^{th}$ block are set to zero). Note that it is not necessary to use more memory than $N_b \times 12 N$.

5. orthogonalize this $N_s N_b$ vectors. Note than the Gram-Schmidt (or other procedure) can be applied into the blocks separately since between blocks the orthogonality is automatic.

6. Compute the $N_s N_b \times N_s N_b$ matrix $A$, called little Dirac operator, with entry $A_{lb}^{l'b'} = \langle \psi_{l'}^b | D | \psi_{l}^b \rangle$. Note that it requires not more than $N_s$ $D$–application and scalar product (there is some extra memory requirement for the sites of the boundary of the block).

7. Invert $A$, to a good accuracy,

8. compute $|X\rangle = \sum_{k,l=1}^{N_s N_b} (A^{-1})_{kl} \langle \psi_l^b | \psi_k^b \rangle D | \psi_k^b \rangle$ (the index of the block has been skipped for clarity). again only $N_s$ scalar product are necessary,

9. find, iteratively (no other way !) one solution of the deflated Dirac operator $P_L D$, and apply $P_R$ to get a unique and well defined vector $|Y\rangle$

10. The desired solution is $|X\rangle + |Y\rangle$. That’s all folks!

Notes

• One has

\[ P_L |\psi\rangle = \sum_{k,l=1}^{N_s N_b} (A^{-1})_{kl} \langle \psi_l^b | \psi_k^b \rangle D | \psi_k^b \rangle \quad (4.4) \]

\[ P_R |\psi\rangle = \sum_{k,l=1}^{N_s N_b} (A^{-1})_{kl} \langle \psi_l^b | D \psi_k^b \rangle | \psi_k^b \rangle \quad (4.5) \]

Note that $P_R$ is much longer to apply than $P_L$ since $D$ is applied $N_b N_s$ times, but $P_R$ is applied only once whereas $P_L$ is applied many times.

• Extra Memory : In addition to the memory for the standard procedure, one needs to store the $2 N_s$ vectors $|\psi_k^b\rangle$, and $D | \psi_k \rangle$, and the $N_s N_b \times N_s N_b$ matrix (little Dirac operator), together with some peanuts.

• Extra computation : Applying the deflated Dirac operator instead of the Dirac operator implies $N_s N_b$ extra scalar product. The little Dirac operator has to be inverted, and $N_b$ extra scalar product must computed.

$2N$ being the number of sites of the lattice
4.4 Deflation: theory

In this paragraph we give the details of the proofs and explain why Eq. (4.4) and Eq. (4.5) are good choices. Note $P_L$ and $P_R$ are the $P$ and $Q$ of the introduction, and as usual $P_L = 1 - P_R$ and $P_R = 1 - P_L$.

4.4.1 Some properties of $P_R$ and $P_R^2$

- Let us show that $P_L$ and $P_R$, or equivalently $P_L^2$ and $P_R^2$, are projectors

$$P_L^2 |\psi\rangle = \sum_{k,l} (A^{-1})_{kl} \left( \psi_l \left( \sum_{q,m} (A^{-1})_{qm} \langle \psi_m | D | \psi_q \rangle \right) \right) |\psi_k\rangle$$

$$= \sum_{k,l,q,m} (A^{-1})_{kl} (A^{-1})_{qm} \langle \psi_m | \langle \psi_l | D | \psi_q \rangle \rangle |\psi_k\rangle$$

$$= \sum_{k,l,q,m} (A^{-1})_{kl} (A^{-1})_{qm} A_{lq} \langle \psi_m | \langle \psi_l | D | \psi_q \rangle \rangle |\psi_k\rangle$$

$$= \sum_{k,l,m} (A^{-1})_{kl} \delta_{l,m} \langle \psi_m | \langle \psi_l | D | \psi_q \rangle \rangle |\psi_k\rangle$$

$$= P_L |\psi\rangle$$

where we have used the definition of the matrix $A$.

- $P_R$ is also a projector

$$P_R^2 |\psi\rangle = \sum_{k,l} (A^{-1})_{kl} \left( \psi_l \left( \sum_{q,m} (A^{-1})_{qm} \langle \psi_m | D | \psi_q \rangle \right) \right) |\psi_k\rangle$$

$$= \sum_{k,l,q,m} (A^{-1})_{kl} (A^{-1})_{qm} \langle \psi_m | \langle \psi_l | D | \psi_q \rangle \rangle |\psi_k\rangle$$

$$= \sum_{k,l,q,m} (A^{-1})_{kl} (A^{-1})_{qm} A_{lq} \langle \psi_m | \langle \psi_l | D | \psi_q \rangle \rangle |\psi_k\rangle$$

$$= \sum_{k,l,m} (A^{-1})_{kl} \delta_{l,m} \langle \psi_m | \langle \psi_l | D | \psi_q \rangle \rangle |\psi_k\rangle$$

$$= P_R |\psi\rangle$$

18
• Let’s now show that $P_L D = D P_R$ or equivalently $\overline{P_L} D = D \overline{P_R}$

\[
D \overline{P_R} |\psi\rangle = D \left( \sum_{k,l=1} (A^{-1})_{kl} \langle \psi_l | D \psi_k \rangle |\psi_k\rangle \right)
\]
\[
= \sum_{k,l=1} (A^{-1})_{kl} \langle \psi_l | D \psi_k \rangle |\psi_k\rangle
\]
\[
= \overline{P_L} D |\psi\rangle
\]

• Let’s show $PP_L = 0$

\[
PP_L |\psi\rangle = P \left( |\psi\rangle - \sum_{q,l} (A^{-1})_{k,l} \langle \psi_l | D \psi_q \rangle |\psi_q\rangle \right)
\]
\[
= \sum_q |\psi_q\rangle \langle \psi_q | |\psi\rangle - \sum_{q,k,l} |\psi_q\rangle \langle \psi_q | (A^{-1})_{k,l} \langle \psi_l | D \psi_q \rangle |\psi_q\rangle
\]
\[
= \sum_q |\psi_q\rangle \langle \psi_q | |\psi\rangle - \sum_{q,l} \left( \sum_k A_{q,k} (A^{-1})_{k,l} \right) \langle \psi_l | D \psi_q \rangle |\psi_q\rangle
\]
\[
= \sum_q |\psi_q\rangle \langle \psi_q | |\psi\rangle - \sum_{q,l} \delta_{q,l} \langle \psi_l | D \psi_q \rangle |\psi_q\rangle
\]
\[
= 0
\]

4.4.2 The Little Dirac Operator

This is the system

\[
\overline{P_L} D |\psi\rangle = \overline{P_L} |\eta\rangle
\]

Consider the vector

\[
|S\rangle = D^{-1} \overline{P_L} |\eta\rangle
\]

it is easy to see that it is a solution of Eq. (4.6) ($\overline{P_L} DD^{-1} \overline{P_L} |\eta\rangle = \overline{P_L} |\eta\rangle$). Then as explained in section one has to apply the operator $\overline{P_R}$ to $|S\rangle$. Note that $\overline{P_R} D^{-1} \overline{P_L} = D^{-1} \overline{P_L} = D^{-1}$ is very easy to compute as

\[
D^{-1} \overline{P_L} |\eta\rangle = \sum_{k,l} (A^{-1})_{k,l} \langle \psi_l | \eta \rangle |\psi_k\rangle
\]
Chapter 5

Mathematic tools

Definition of Gamma matrices

\[ \gamma_4 = \gamma_0 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \] (5.1)

\[ \gamma_2 = \gamma_1 = \begin{pmatrix} 0 & 0 & 0 & i \\ 0 & 0 & i & 0 \\ 0 & -i & 0 & 0 \\ -i & 0 & 0 & 0 \end{pmatrix} \] (5.2)

\[ \gamma_9 = \gamma_2 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \] (5.3)

\[ \gamma_9 = \gamma_3 = \begin{pmatrix} 0 & 0 & i & 0 \\ 0 & 0 & 0 & -i \\ -i & 0 & 0 & 0 \\ 0 & i & 0 & 0 \end{pmatrix} \] (5.4)

\[ \gamma_5 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \] (5.5)
There are other representations of the gamma matrices which have the same properties. We will keep these ones.

**Algebraic properties of the gamma matrices**

For \( \mu, \nu = 0, 3 \) (or identically \( \mu, \nu = 1, 4 \)) and using the shorthand notation

\[ 1 = I_{4 \times 4} \]

\[ \gamma_\mu \gamma_\nu + \gamma_\nu \gamma_\mu = \delta_{\mu \nu} \quad \gamma_\mu \gamma_5 + \gamma_5 \gamma_\mu = 0 \quad \gamma_5^2 = 1 \]

whence

\[ \frac{1 \pm \gamma_5}{2} = 0 \quad \frac{1 \pm \gamma_\mu}{2} = \frac{1 \pm \gamma_\mu}{2} \gamma_\mu \]

where \( \delta_{\mu \nu} \) is the Kronecker symbol. Notice that \( (1 \pm \gamma_5)/2 \) are complementary projectors in spin space. These are denoted “chirality projectors” with \( (1 + \gamma_5)/2 \) (respectively \( (1 - \gamma_5)/2 \)) is the projector on positive (respectively negative) chirality.

**Definition of SU(3)**

The hermitian conjugate of a \( 3 \times 3 \) complex matrix

\[ U \equiv \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \\ u_{31} & u_{32} & u_{33} \end{pmatrix} \]  

is defined by

\[ U^\dagger \equiv \begin{pmatrix} u_{11}^* & u_{21}^* & u_{31}^* \\ u_{12}^* & u_{22}^* & u_{32}^* \\ u_{13}^* & u_{23}^* & u_{33}^* \end{pmatrix} \]  

where for any complex number \( z, z^* \) is the complex conjugate (changed the sign of the imaginary part). A matrix \( U \) is told to be unitary if

\[ U^\dagger \ast U = I_{3 \times 3} \]  

where \( \ast \) denotes the matrix product and \( I_{3 \times 3} \) is the identity \( 3 \times 3 \) matrix. "Special unitary" means that furthermore the determinant of \( U \) equals 1.

These matrices can be stored in a reduced array, for example keeping only two lines allows to recompute the third line using the \( SU(3) \) properties. This is equivalent to store 12 floating point numbers. An even better choice is given in http://www.ccs.tsukuba.ac.jp/workshop/EP09/, using only 8 floating point numbers. Of course the gain in storage has to be balanced against the cost in additional online computation.
Chapter 6

Samples from ETMC codes

```c
void mul_one_pm_imu_sub_mul_gamma5(spinor * const l, spinor * const k,
    spinor * const j, const double _sign){
    complex z,w;
    int ix;
    double sign=1.;
    spinor *r, *s, *t;
    static su3_vector phi1, phi2, phi3, phi4;

    if(_sign < 0.){
        sign = -1.;
    }

    z.re = 1.;
    z.im = sign * g_mu;
    w.re = 1.;
    w.im = -sign * g_mu;
#if (defined BGL3 && defined XLC)
    __alignx(16,l);
    __alignx(16,k);
    __alignx(16,j);
#endif
    /************ loop over all lattice sites ************/
    for(ix = 0; ix < (VOLUME/2); ix++){
        r = k+ix;
        s = j+ix;
        t = l+ix;
        /* Multiply the spinorfield with 1+imu\gamma_5 */
        #if (defined SSE22 || defined SSE32)
```
```c
//Prefetch spinor
(prefetch_spinor((r+predist));
(prefetch_spinor((s+predist));
(sse_load_up((*r).s0);
(sse_vector_cmplx_mul(z);
(sse_load((*s).s0);
(sse_vector_sub_up();
(sse_store_nt_up((*t).s0);
(sse_load_up((*r).s1);
(sse_vector_cmplx_mul_two();
(sse_load((*s).s1);
(sse_vector_sub_up();
(sse_store_nt_up((*t).s1);
(sse_load_up((*r).s2);
(sse_vector_cmplx_mul(w);
(sse_load((*s).s2);
(sse_vector_sub();
(sse_store_nt_up((*t).s2);
(sse_load_up((*r).s3);
(sse_vector_cmplx_mul_two();
(sse_load((*s).s3);
(sse_vector_sub();
(sse_store_nt_up((*t).s3);

#else
(complex_times_vector(phi1, z, (*r).s0);
(complex_times_vector(phi2, z, (*r).s1);
(complex_times_vector(phi3, w, (*r).s2);
(complex_times_vector(phi4, w, (*r).s3);
/* Subtract s and store the result in t */
/* multiply with gamma5 included by */
/* reversed order of s and phi3|4 */
(vector_sub((*t).s0, phi1, (*s).s0);
(vector_sub((*t).s1, phi2, (*s).s1);
(vector_sub((*t).s2, (*s).s2, phi3);
(vector_sub((*t).s3, (*s).s3, phi4);
#endif
}
}

return to Wilson-twisted Dirac operator

/* $Id: Hopping_Matrix.c,v 1.43 2007/08/29 08:39:29 urbach Exp $ */

/ ******************************************
* Hopping_Matrix is the conventional Wilson
* hopping matrix
```
\[ \kappa \sum_{\pm \mu} (r + \gamma_{\mu}) U_{x, \mu} \]

for ieo = 0 this is \( M_{\text{eo}} \), for ieo = 1 it is \( M_{\text{oe}} \)

** is the number of the output field

* is the number of the input field

*******************************

static su3_vector psi1, psi2, psi, chi, phi1, phi3;

/* l output , k input*/
/* for ieo=0, k resides on odd sites and l on even sites */
void Hopping_Matrix(int ieo, spinor * const l, spinor * const k){
    int ix, iy;
    int ioff, ioff2, icx, icy;
    su3 * restrict up, * restrict um;
    spinor * restrict r, * restrict sp, * restrict sm;
    spinor temp;

    /* for parallelization */
    # if (defined MPI && !(defined _NO_COMM))
        xchange_field(k, ieo);
    # endif

    if(k == l){
        printf("Error in H_psi (simple.c):\n  Arguments k and l must be different\n");
        printf("Program aborted\n");
        exit(1);
    }
    if(ieo == 0){
        ioff = 0;
    }
    else{
        ioff = (VOLUME+RAND)/2;
    }
    ioff2 = (VOLUME+RAND)/2-ioff;

    **************** loop over all lattice sites *****************

    for (icx = ioff; icx < (VOLUME/2 + ioff); icx++){
        ix=g_eo2lexic[icx];
r = l + (icx - ioff);

/*********************** direction +0 ************************/
 iy = g_iup[ix][0]; icy = g_lexic2eosub[iy];

sp = k + icy;
 # if ((defined _GAUGE_COPY))
 up = &g_gauge_field_copy[icx][0];
 # else
 up = &g_gauge_field[ix][0];
 # endif
 _vector_add(psi, (*sp).s0, (*sp).s2);
 _su3_multiply(chi, (*up), psi);
 _complex_times_vector(psi, ka0, chi);
 _vector_assign(temp.s0, psi);
 _vector_assign(temp.s2, psi);

/sp = k + icy;
 # if ((defined _GAUGE_COPY))
 um = up + 1;
 # else
 um = &g_gauge_field[iy][0];
 # endif
 _vector_sub(psi, (*sm).s0, (*sm).s2);
 _su3_inverse_multiply(chi, (*um), psi);
 _complex_cjg_times_vector(psi, ka0, chi);

iy = g_idn[ix][0]; icy = g_lexic2eosub[iy];

# if ((defined _GAUGE_COPY))
 um = up + 1;
 # else
 um = &g_gauge_field[iy][0];
 # endif
 _vector_sub(psi, (*sm).s0, (*sm).s2);
 _su3_inverse_multiply(chi, (*um), psi);
 _complex_cjg_times_vector(psi, ka0, chi);
VECTOR_ADD_ASSIGN(temp.s0, psi);
VECTOR_SUB_ASSIGN(temp.s2, psi);
VECTOR_SUB(psi, (*sm).s1, (*sm).s3);
su3_inverse_multiply(chi, (*um), psi);
complex_cjg_times_vector(psi, ka0, chi);
VECTOR_ADD_ASSIGN(temp.s1, psi);
VECTOR_SUB_ASSIGN(temp.s3, psi);

/******************** direction +1 *********************/

iy = g_iup[ix][1]; icy = g_lexic2eosub[iy];

sp = k + icy;

# if ((defined _GAUGE_COPY))
up = um + 1;
# else
up *= 1;
# endif

VECTOR_I_ADD(psi, (*sp).s0, (*sp).s3);
su3_multiply(chi, (*up), psi);
complex_times_vector(psi, ka1, chi);
VECTOR_ADD_ASSIGN(temp.s0, psi);
VECTOR_I_SUB_ASSIGN(temp.s3, psi);
VECTOR_I_ADD(psi, (*sp).s1, (*sp).s2);
su3_multiply(chi, (*up), psi);
complex_times_vector(psi, ka1, chi);
VECTOR_ADD_ASSIGN(temp.s1, psi);
VECTOR_I_SUB_ASSIGN(temp.s2, psi);

/******************** direction -1 *********************/

iy = g_idn[ix][1]; icy = g_lexic2eosub[iy];
sm=k+icy;
# ifndef _GAUGE_COPY
  um=&g_gauge_field[iy][1];
# else
  um=up+1;
# endif

_vector_i_sub(psi,(*sm).s0,(*sm).s3);

_su3_inverse_multiply(chi,(*um),psi);
_complexcjcjg_times_vector(psi,ka1,chi);

_vector_add_assign(temp.s0,psi);
_vector_i_add_assign(temp.s3,psi);

_vector_i_sub(psi,(*sm).s1,(*sm).s2);

_su3_inverse_multiply(chi,(*um),psi);
_complexcjcjg_times_vector(psi,ka1,chi);

_vector_add_assign(temp.s1,psi);
_vector_i_add_assign(temp.s2,psi);

return to Wilson-twisted Dirac operator

/*********************** direction +2 **********************/

iy=g_iup[ix][2]; icy=g_lexic2eosub[iy];

sp=k+icy;
# if ((defined _GAUGE_COPY))
  up=um+1;
# else
  up+=1;
# endif

_vector_add(psi,(*sp).s0,(*sp).s3);

_su3_multiply(chi,(*up),psi);
_complex_times_vector(psi,ka2,chi);

_vector_add_assign(temp.s0,psi);
_vector_add_assign(temp.s3,psi);

_vector_sub(psi,(*sp).s1,(*sp).s2);
_su3_multiply(chi,(*up),psi);
_complex_times_vector(psi,ka2,chi);

_vector_add_assign(temp.s1,psi);
_vector_sub_assign(temp.s2,psi);

/*********************** direction -2 ***********************/
iy=g_idn[ix][2]; icy=g_lexic2eosub[iy];

sm=k+icy;
# ifndef _GAUGE_COPY
um = &g_gauge_field[iy][2];
# else
um = up +1;
# endif

_vector_sub(psi,(sm).s0,(sm).s3);

_su3_inverse_multiply(chi,(um),psi);
_complexcjh_times_vector(psi,ka2,chi);

_vector_add_assign(temp.s0,psi);
_vector_sub_assign(temp.s3,psi);

_vector_add(psi,(sm).s1,(sm).s2);

_su3_inverse_multiply(chi,(um),psi);
_complexcjh_times_vector(psi,ka2,chi);

_vector_add_assign(temp.s1,psi);
_vector_add_assign(temp.s2,psi);

/*********************** direction +3 ***********************/
iy=g_iup[ix][3]; icy=g_lexic2eosub[iy];

sp=k+icy;
# if (defined _GAUGE_COPY)
up=um+1;
# else
up=up+1;
# endif

_vector_i_add(psi,(sp).s0,(sp).s2);
_su3_multiply(chi,(*up),psi);
_complex_times_vector(psi,ka3,chi);

_vector_add_assign(temp.s0,psi);
_vector_i_sub_assign(temp.s2,psi);

_vector_i_sub(psi,(*sp).s1,(*sp).s3);

/su3_multiply(chi,(*up),psi);
/complex_times_vector(psi,ka3,chi);
/vector_add_assign(temp.s1,psi);
/vector_i_add_assign(temp.s3,psi);

/************************ direction -3 ***********************/
iy=g_idn[ix][3]; icy=g_lexic2eosub[iy];

sm=k+icy;
#ifdef _GAUGE_COPY
  um = &g_gauge_field[iy][3];
#else
  um = up+1;
#endif

/vector_i_sub(psi,(*sm).s0,(*sm).s2);

/su3_inverse_multiply(chi,(*um),psi);
/complexcjg_times_vector(psi,ka3,chi);
/vector_add((*r).s0, temp.s0, psi);
/vector_i_add((*r).s2, temp.s2, psi);
/vector_i_add(psi,(*sm).s1,(*sm).s3);

/su3_inverse_multiply(chi,(*um),psi);
/complexcjg_times_vector(psi,ka3,chi);
/vector_add((*r).s1, temp.s1, psi);
/vector_i_sub((*r).s3, temp.s3, psi);

/************************ end of loop **********************/
}
}
/* end of If defined SSE2 */
#endif
static void random_fields(const int Ns) {

    int i, j, ix;
    float r, s[24];
    double *t;

    r = (float)(1.0/sqrt(24.0*(double)(VOLUME)));

    for (i=0; i<Ns; i++) {
        t = (double*)(dfl_fields[i]);
        for (ix = 0; ix < VOLUME; ix++){
            ranlxs(s, 24);
            for (j = 0; j < 24; j++) {
                (*t) = (double)(r*(s[j]-0.5f));
                t += 1;
            }
        }
    }

    return;
}

return to deflation
Bibliography

[1] Local coherence and deflation of the low quark modes in QCD
   M. Luscher JHEP07 081 (2007)