Möbius Domain Wall Fermions Implementation

Andrew Pochinsky

$Id: mdwf.nw 607 2008-05-18 01:46:41Z avp $
Release 1.1.4

Contents

Chapter 1

PHYSICS

The Domain Wall Fermion Dirac operator is defined by

$$\langle \bar{\psi} | D_{DW} | \psi \rangle = \sum_{x,x'} \bar{\psi}(x) D_{DW}(x,x') \psi(x'), \tag{1.1}$$

where

$$\begin{aligned} D_{DW}(x,x') &= D_+^{(s)}(x,x')\delta_{s,s'} \\ &\quad + D_-^{(s)}(x,x')P_+\delta_{s,s'+1} - m D_-^{(s)}(x,x')P_+\delta_{s,0}\delta_{s',L_s-1} \\ &\quad + D_-^{(s)}(x,x')P_-\delta_{s,s'-1} - m D_-^{(s)}(x,x')P_-\delta_{s,L_s-1}\delta_{s',0}, \end{aligned} \tag{1.2}$$

$$P_+ = \frac{1+\gamma_5}{2}, \tag{1.3}$$

$$P_- = \frac{1-\gamma_5}{2}, \tag{1.4}$$

$$D_+^{(s)}(x,x') = b_5(s) D_W(x,x') + 1, \tag{1.5}$$

$$D_-^{(s)}(x,x') = c_5(s) D_W(x,x') - 1, \tag{1.6}$$

and

$$D_W(x,x') = (4+M_5)\delta_{x,x'} - \frac{1}{2}\sum_{\mu=0}^{3}\left[(1-\gamma_\mu)U_\mu(x)\delta_{x,x'-\hat{\mu}} + (1+\gamma_\mu)U_\mu^\dagger(x-\hat{\mu})\delta_{x,x'+\hat{\mu}}\right] \tag{1.7}$$

is the standard Wilson action.

1.1 Gamma Matrices

We use the same γ-matrix basis as Chroma to simplify conversion between two codes. The choice below could be changed with a few modifications to the rest of the code, if γ_5 is kept diagonal, and one of other γ-matrices has all nonzero entries equal to $+1$.
In the γ-basis defined below one has

$$\gamma_5 = \gamma_0\gamma_1\gamma_2\gamma_3 = \sigma_3 \otimes 1 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \tag{1.8}$$

Fragments for γ_μ below are not the most fool-proof, but they should do for now.

$$\gamma_0 = -\sigma_2 \otimes \sigma_1 = \begin{pmatrix} 0 & i\sigma_1 \\ -i\sigma_1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & i \\ 0 & 0 & i & 0 \\ 0 & -i & 0 & 0 \\ -i & 0 & 0 & 0 \end{pmatrix} \tag{1.9}$$

6a ⟨*Project* $(1+\gamma_0)$ 6a⟩≡

```
((project 0 plus) . ((plus-one 0 plus-i 3)
                     (plus-one 1 plus-i 2)))
```

This code is used in chunk 9f.

6b ⟨*Unproject* $(1+\gamma_0)$ 6b⟩≡

```
((unproject 0 plus) . ((plus-one 0)
                       (plus-one 1)
                       (minus-i 1)
                       (minus-i 0)))
```

This code is used in chunk 9f.

6c ⟨*Project* $(1-\gamma_0)$ 6c⟩≡

```
((project 0 minus) . ((plus-one 0 minus-i 3)
                      (plus-one 1 minus-i 2)))
```

This code is used in chunk 9f.

6d ⟨*Unproject* $(1-\gamma_0)$ 6d⟩≡

```
((unproject 0 minus) . ((plus-one 0)
                        (plus-one 1)
                        (plus-i 1)
                        (plus-i 0)))
```

This code is used in chunk 9f.

$$\gamma_1 = \sigma_2 \otimes \sigma_2 = \begin{pmatrix} 0 & -i\sigma_2 \\ i\sigma_2 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \end{pmatrix} \tag{1.10}$$

7a ⟨*Project* $(1+\gamma_1)$ 7a⟩≡

```
((project 1 plus) . ((plus-one 0 minus-one 3)
                     (plus-one 1 plus-one 2)))
```

This code is used in chunk 9f.

7b ⟨*Unproject* $(1+\gamma_1)$ 7b⟩≡

```
((unproject 1 plus) . ((plus-one 0)
                       (plus-one 1)
                       (plus-one 1)
                       (minus-one 0)))
```

This code is used in chunk 9f.

7c ⟨*Project* $(1-\gamma_1)$ 7c⟩≡

```
((project 1 minus) . ((plus-one 0 plus-one 3)
                      (plus-one 1 minus-one 2)))
```

This code is used in chunk 9f.

7d ⟨*Unproject* $(1-\gamma_1)$ 7d⟩≡

```
((unproject 1 minus) . ((plus-one 0)
                        (plus-one 1)
                        (minus-one 1)
                        (plus-one 0)))
```

This code is used in chunk 9f.

$$\gamma_2 = -\sigma_2 \otimes \sigma_3 = \begin{pmatrix} 0 & i\sigma_3 \\ -i\sigma_3 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & i & 0 \\ 0 & 0 & 0 & -i \\ -i & 0 & 0 & 0 \\ 0 & i & 0 & 0 \end{pmatrix} \tag{1.11}$$

8a ⟨*Project* $(1+\gamma_2)$ 8a⟩≡

```
((project 2 plus) . ((plus-one 0 plus-i 2)
                     (plus-one 1 minus-i 3)))
```

This code is used in chunk 9f.

8b ⟨*Unproject* $(1+\gamma_2)$ 8b⟩≡

```
((unproject 2 plus) . ((plus-one 0)
                       (plus-one 1)
                       (minus-i 0)
                       (plus-i 1)))
```

This code is used in chunk 9f.

8c ⟨*Project* $(1-\gamma_2)$ 8c⟩≡

```
((project 2 minus) . ((plus-one 0 minus-i 2)
                      (plus-one 1 plus-i 3)))
```

This code is used in chunk 9f.

8d ⟨*Unproject* $(1-\gamma_2)$ 8d⟩≡

```
((unproject 2 minus) . ((plus-one 0)
                        (plus-one 1)
                        (plus-i 0)
                        (minus-i 1)))
```

This code is used in chunk 9f.

$$\gamma_3 = \sigma_1 \otimes 1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \tag{1.12}$$

9a ⟨*Project* $(1+\gamma_3)$ 9a⟩≡
```
((project 3 plus) . ((plus-one 0 plus-one 2)
                     (plus-one 1 plus-one 3)))
```
This code is used in chunk 9f.

9b ⟨*Unproject* $(1+\gamma_3)$ 9b⟩≡
```
((unproject 3 plus) . ((plus-one 0)
                       (plus-one 1)
                       (plus-one 0)
                       (plus-one 1)))
```
This code is used in chunk 9f.

This is our starting point in sums over directions

9c ⟨*Start* μ *sum* 9c⟩≡
```
(define mdwf-start-sum-dimension 3)
(define mdwf-start-sum-direction 'plus)
```
This code is used in chunk 9g.

9d ⟨*Project* $(1-\gamma_3)$ 9d⟩≡
```
((project 3 minus) . ((plus-one 0 minus-one 2)
                      (plus-one 1 minus-one 3)))
```
This code is used in chunk 9f.

9e ⟨*Unproject* $(1-\gamma_3)$ 9e⟩≡
```
((unproject 3 minus) . ((plus-one 0)
                        (plus-one 1)
                        (minus-one 0)
                        (minus-one 1)))
```
This code is used in chunk 9f.

Now let us collect the γ-matrix projections and reconstructions. We put them all together into `mdwf-basis` as an a list of pairs with keys of the form `(<op> <dir> <sign>)`.

9f ⟨*Scheme definitions* 9f⟩≡
```
(define mdwf-basis '(
```
⟨*Project* $(1+\gamma_0)$ 6a⟩
⟨*Project* $(1+\gamma_1)$ 7a⟩
⟨*Project* $(1+\gamma_2)$ 8a⟩
⟨*Project* $(1+\gamma_3)$ 9a⟩
⟨*Project* $(1-\gamma_0)$ 6c⟩
⟨*Project* $(1-\gamma_1)$ 7c⟩
⟨*Project* $(1-\gamma_2)$ 8c⟩
⟨*Project* $(1-\gamma_3)$ 9d⟩
⟨*Unproject* $(1+\gamma_0)$ 6b⟩
⟨*Unproject* $(1+\gamma_1)$ 7b⟩
⟨*Unproject* $(1+\gamma_2)$ 8b⟩
⟨*Unproject* $(1+\gamma_3)$ 9b⟩
⟨*Unproject* $(1-\gamma_0)$ 6d⟩
⟨*Unproject* $(1-\gamma_1)$ 7d⟩
⟨*Unproject* $(1-\gamma_2)$ 8d⟩
⟨*Unproject* $(1-\gamma_3)$ 9e⟩`))`
This definition is continued in chunk 9g.
This code is used in chunk 10.

We also define a starting link in a sum over links:

9g ⟨*Scheme definitions* 9f⟩+≡
⟨*Start* μ *sum* 9c⟩

Here is a module for PLT:

10 ⟨*File* `../utils/basis.ss` 10⟩≡

```
(module basis
        mzscheme
   (provide mdwf-basis
            mdwf-start-sum-dimension
            mdwf-start-sum-direction)
   ⟨Scheme definitions 9f⟩
)
```

Root chunk (not used in this document).

1.2 Preconditioning

We use four dimensional preconditioner to improve convergence of the CG. Following Kostas Orginos, let us color the lattice sites according to the parity of $x_0 + x_1 + x_2 + x_3$. Then we can rewrite D_{DW} from Eq. (1.2) as follows:

$$D_{DW} = \begin{pmatrix} A_{oo} & F_{oe}B_{ee} \\ F_{eo}B_{oo} & A_{ee} \end{pmatrix}, \tag{1.13}$$

where

$$\begin{aligned} A_{oo}(x,x') &= \{(c_5(s)(M_5+4)-1)\left[P_+\delta_{s,s'+1} - mP_+\delta_{s,0}\delta_{s',L_s-1} + P_-\delta_{s,s'-1} - mP_-\delta_{s,L_s-1}\delta_{s',0}\right] \\ &\quad +(b_5(s)(M_5+4)+1)\delta_{s,s'}\}\,\delta_{x,x'}, \qquad (1.14) \\ B_{oo}(x,x') &= \{c_5(s)\left[P_+\delta_{s,s'+1} - mP_+\delta_{s,0}\delta_{s',L_s-1} + P_-\delta_{s,s'-1} - mP_-\delta_{s,L_s-1}\delta_{s',0}\right] + b_5(s)\delta_{s,s'}\}\,\delta_{x,x'}, \qquad (1.15) \\ F_{oe}(x,x') &= -\frac{\delta_{s,s'}}{2}\sum_{\mu=0}^{3}\left[(1-\gamma_\mu)U_\mu(x)\delta_{x,x'-\hat{\mu}} + (1+\gamma_\mu)U_\mu^\dagger(x-\hat{\mu})\delta_{x,x'+\hat{\mu}}\right], \qquad (1.16) \end{aligned}$$

and similary for other parity components. (On the LHS x and x' are 5-d indices, hereafter spinor and color indices are suppressed but presumed.)
Let us rewrite Eq. (1.13) as follows:

$$D_{DW} = \begin{pmatrix} I_{oo} & 0 \\ F_{eo}B_{oo}A_{oo}^{-1} & A_{ee}B_{ee}^{-1} \end{pmatrix} \begin{pmatrix} A_{oo} & F_{oe} \\ 0 & I_{ee} - B_{ee}A_{ee}^{-1}F_{eo}B_{oo}A_{oo}^{-1}F_{oe} \end{pmatrix} \begin{pmatrix} I_{oo} & 0 \\ 0 & B_{ee} \end{pmatrix}. \tag{1.17}$$

To solve the equation

$$D_{DW}\psi = \begin{pmatrix} A_{oo} & F_{oe}B_{ee} \\ F_{eo}B_{oo} & A_{ee} \end{pmatrix} \begin{pmatrix} \psi_e \\ \psi_o \end{pmatrix} = \begin{pmatrix} \eta_e \\ \eta_o \end{pmatrix},$$

one performs the following steps:

1. Use $M = I_{ee} - B_{ee}A_{ee}^{-1}F_{eo}B_{oo}A_{oo}^{-1}F_{oe}$ in the following.
2. Use $M^\dagger = I_{ee} - (F_{oe})^\dagger \left(A_{oo}^{-1}\right)^\dagger (B_{oo})^\dagger (F_{eo})^\dagger \left(A_{ee}^{-1}\right)^\dagger (B_{ee})^\dagger$ in the following.
3. Compute
$$\chi_e = M^\dagger B_{ee}A_{ee}^{-1}\left(\eta_e - F_{eo}B_{oo}A_{oo}^{-1}\eta_o\right).$$
4. Solve
$$M^\dagger M\xi_e = \chi_e$$
for ξ_e using Alg. 3.
5. Compute
$$\begin{aligned} \psi_o &= A_{oo}^{-1}\left(\eta_o - F_{oe}\xi_e\right), \qquad (1.18) \\ \psi_e &= B_{ee}^{-1}\xi_e. \qquad (1.19) \end{aligned}$$

1.3 Inverting A and B

Note that A and B have the following form:

$$A(x,x') = [A_+(s,s')P_+ + A_-(s,s')P_-]\,\delta_{x,x'} \tag{1.20}$$
$$B(x,x') = [B_+(s,s')P_+ + B_-(s,s')P_-]\,\delta_{x,x'} \tag{1.21}$$

where

$$A_+(s,s') = u_\alpha(s)\delta_{s,s'+1} + v_\alpha(s)\delta_{s,0}\delta_{s',L_s-1} + w_\alpha(s)\delta_{s,s'}, \tag{1.22}$$
$$A_-(s,s') = u_\alpha(s)\delta_{s,s'-1} + v_\alpha(s)\delta_{s,L_s-1}\delta_{s',0} + w_\alpha(s)\delta_{s,s'}, \tag{1.23}$$
$$B_+(s,s') = u_\beta(s)\delta_{s,s'+1} + v_\beta(s)\delta_{s,0}\delta_{s',L_s-1} + w_\beta(s)\delta_{s,s'}, \tag{1.24}$$
$$B_-(s,s') = u_\beta(s)\delta_{s,s'-1} + v_\beta(s)\delta_{s,L_s-1}\delta_{s',0} + w_\beta(s)\delta_{s,s'}; \tag{1.25}$$

and

$$u_\alpha(s) = c_5(s)(M_5+4) - 1, \tag{1.26}$$
$$v_\alpha(s) = -mu_\alpha(s), \tag{1.27}$$
$$w_\alpha(s) = b_5(s)(M_5+4) + 1, \tag{1.28}$$
$$u_\beta(s) = c_5(s), \tag{1.29}$$
$$v_\beta(s) = -mu_\beta(s), \tag{1.30}$$
$$w_\beta(s) = b_5(s). \tag{1.31}$$

This allows us to invert A and B as follows.
For A_+ one has (formulae for B_+ are obtained by replacing α with β, see Eqs. (1.22) and (1.24)):

$$A_+ = A_{Y+}A_{L+} = \begin{pmatrix} w_\alpha(0) & 0 & 0 & \cdots & 0 & v_\alpha(0) \\ u_\alpha(1) & w_\alpha(1) & 0 & \cdots & 0 & 0 \\ 0 & u_\alpha(2) & w_\alpha(2) & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & w_\alpha(L_s-2) & 0 \\ 0 & 0 & 0 & \cdots & u_\alpha(L_s-1) & w_\alpha(L_s-1) \end{pmatrix}, \tag{1.32}$$

$$A_{L+} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ u_\alpha(1) & w_\alpha(1) & 0 & \cdots & 0 & 0 \\ 0 & u_\alpha(2) & w_\alpha(2) & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & w_\alpha(L_s-2) & 0 \\ 0 & 0 & 0 & \cdots & u_\alpha(L_s-1) & w_\alpha(L_s-1) \end{pmatrix}, \tag{1.33}$$

$$A_{Y+} = \begin{pmatrix} 1/z_\alpha^{(+)} & a_\alpha^{(+)}(1) & \cdots & a_\alpha^{(+)}(Ls-2) & a_\alpha^{(+)}(Ls-1) \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix}. \tag{1.34}$$

For the following it is convenient to define

$$a_\alpha^{(+)}(Ls-1) = -\frac{v_\alpha(0)}{w_\alpha(L_s-1)}, \tag{1.35}$$
$$a_\alpha^{(+)}(k) = -\frac{a_\alpha^{(+)}(k+1)u_\alpha(k)}{w_\alpha(k)}, \tag{1.36}$$
$$z_\alpha^{(+)} = \frac{1}{w_\alpha(0)(1-a_\alpha^{(+)}(0))}, \tag{1.37}$$
$$b_\alpha^{(+)}(k) = -\frac{u_\alpha(k)}{w_\alpha(k)}, \tag{1.38}$$
$$c_\alpha^{(+)}(k) = \frac{1}{w_\alpha(k)}. \tag{1.39}$$

Input: z, precomputed part of A_{Y+}
Input: a, precomputed part of A_{Y+}
Input: b, precomputed part of A_{L+}
Input: c, precomputed part of A_{L+}
Input: ψ, the right hand side
Input: L_s, flavor dimension
Output: ϕ, the result
begin
  $\eta \leftarrow \psi_0$
  $k \leftarrow 1$
  for $k < L_s$ **do**
    $\eta \leftarrow \eta + a_k \psi_k$
    $k \leftarrow k + 1$
  end
  $\phi_0 \leftarrow \eta \leftarrow z\eta$
  $k \leftarrow 1$
  for $k < L_s$ **do**
    $\phi_k \leftarrow \eta \leftarrow b_k \eta + c_k \psi_k$
    $k \leftarrow k + 1$
  end
  return ϕ.
end

Algorithm 1: Computing the inverse of A_+.

Then algorithm 1 could be used to compute $\phi \leftarrow A_+^{-1}\psi$.
For A_- we have the following (once again, B_- is similar.)

$$A_- = A_{Y-}A_{L-} = \begin{pmatrix} w_\alpha(0) & u_\alpha(0) & 0 & \cdots & 0 & 0 \\ 0 & w_\alpha(1) & u_\alpha(1) & \cdots & 0 & 0 \\ 0 & 0 & w_\alpha(2) & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & w_\alpha(L_s-2) & u_\alpha(L_s-2) \\ v_\alpha(L_s-1) & 0 & 0 & \cdots & 0 & w_\alpha(L_s-1) \end{pmatrix}, \tag{1.40}$$

$$A_{L-} = \begin{pmatrix} w_\alpha(0) & u_\alpha(0) & 0 & \cdots & 0 & 0 \\ 0 & w_\alpha(1) & u_\alpha(1) & \cdots & 0 & 0 \\ 0 & 0 & w_\alpha(2) & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & w_\alpha(L_s-2) & u_\alpha(L_s-2) \\ 0 & 0 & 0 & \cdots & 0 & 1 \end{pmatrix}, \tag{1.41}$$

$$A_{Y-} = \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \\ a_\alpha^{(-)}(0) & a_\alpha^{(-)}(1) & \cdots a_\alpha^{(-)}(L_s-2) & 1/z_\alpha^{(-)} & \end{pmatrix}. \tag{1.42}$$

Once again, it is convenient to define

$$a_\alpha^{(-)}(0) = -\frac{v_\alpha(L_s-1)}{w_\alpha(0)}, \tag{1.43}$$

$$a_\alpha^{(-)}(k) = -\frac{a_\alpha^{(-)}(k-1)u_\alpha(k-1)}{w_\alpha(k)}, \tag{1.44}$$

$$z_\alpha^{(-)} = \frac{1}{w_\alpha(L_s-1)(1-a_\alpha^{(-)}(L_s-1))}, \tag{1.45}$$

$$b_\alpha^{(-)}(k) = -\frac{u_\alpha(k)}{w_\alpha(k)}, \tag{1.46}$$

$$c_\alpha^{(-)}(k) = \frac{1}{w_\alpha(k)}. \tag{1.47}$$

Then algorithm 2 could be used to compute $\phi \leftarrow A_-^{-1}\psi$.

Input: z, precomputed part of A_{Y-}
Input: a, precomputed part of A_{Y-}
Input: b, precomputed part of A_{L-}
Input: c, precomputed part of A_{L-}
Input: ψ, the right hand side
Input: L_s, flavor dimension
Output: ϕ, the result
begin
  $\eta \leftarrow \psi_{L_s-1}$
  $k \leftarrow 0$
  for $k < L_s - 1$ **do**
    $\eta \leftarrow \eta + a_k\psi_k$
    $k \leftarrow k + 1$
  end
  $\phi_{L_s-1} \leftarrow \eta \leftarrow z\eta$
  $k \leftarrow L_s - 2$
  for $k \geq 0$ **do**
    $\phi_k \leftarrow \eta \leftarrow b_k\eta + c_k\psi_k$
    $k \leftarrow k - 1$
  end
  return ϕ.
end

Algorithm 2: Computing the inverse of A_-.

1.4 Combinations of A and B

With the notations above one can write other s-pieces we need:

$$A^{-1} = A_{X+}^{-1}A_{L+}^{-1}P_+ + A_{X-}^{-1}A_{L-}^{-1}P_- \tag{1.48}$$
$$A^{-1} = A_{L+}^{-1}A_{Y+}^{-1}P_+ + A_{L-}^{-1}A_{Y-}^{-1}P_- \tag{1.49}$$
$$B^{-1} = B_{X+}^{-1}B_{L+}^{-1}P_+ + B_{X-}^{-1}B_{L-}^{-1}P_- \tag{1.50}$$
$$B^{-1} = B_{L+}^{-1}B_{Y+}^{-1}P_+ + B_{L-}^{-1}B_{Y-}^{-1}P_- \tag{1.51}$$
$$A_+ = A_{L+}A_{X+} \tag{1.52}$$
$$A_+ = A_{Y+}A_{L+} \tag{1.53}$$
$$A_- = A_{L-}A_{X-} \tag{1.54}$$
$$A_- = A_{Y-}A_{L-} \tag{1.55}$$
$$B_+ = B_{L+}B_{X+} \tag{1.56}$$
$$B_+ = B_{Y+}B_{L+} \tag{1.57}$$
$$B_- = B_{L-}B_{X-} \tag{1.58}$$
$$B_- = B_{Y-}B_{L-} \tag{1.59}$$
$$A^\dagger = A_+^\dagger P_+ + A_-^\dagger P_- \tag{1.60}$$
$$B^\dagger = B_+^\dagger P_+ + B_-^\dagger P_- \tag{1.61}$$
$$\tag{1.62}$$

Chapter 2

ALGORITHMS

2.1 Conjugate gradient

The equation

$$M^\dagger M\xi = \chi$$

can be solve by the conjugate gradient method if the condition number of $M^\dagger M$ is small enough.

Input: M, the matrix
Input: χ, the right hand side of the linear equation
Input: ξ_0, an initial guess
Input: n, the maximum number of iterations
Input: ϵ, required precision
Output: ξ, approximate solution
Output: r, final residue
Output: k, number of iterations used
begin
  $\xi \leftarrow \xi_0$
  $\rho \leftarrow \chi - M^\dagger M\xi$
  $\pi \leftarrow \rho$
  $r \leftarrow \langle\rho, \rho\rangle$
  $k \leftarrow 0$
  while $r > \epsilon$ *or* $k < n$ **do**
    $\omega \leftarrow Mp$
    $\zeta \leftarrow M^\dagger\omega$
    $a \leftarrow r/\langle\omega, \omega\rangle$
    $\rho \leftarrow \rho - a\zeta$
    $g \leftarrow \langle\rho, \rho\rangle$
    if $g < \epsilon$ **then**
      $\xi \leftarrow \xi + a\pi$
      $r \leftarrow g$
      break
    end
    $b \leftarrow g/r$
    $r \leftarrow g$
    $\xi \leftarrow \xi + a\pi$
    $\pi \leftarrow \rho + b\pi$
    $k \leftarrow k + 1$
  end
  return ξ, r, k.
end

Algorithm 3: Conjugate Gradient Solver.

2.2 Shifted Conjugate Gradient

We also need the ability to solve equations $(A+s_nI)\xi_n = \chi$, $A = M^\dagger M$ for several s_n and the same RHS χ. It is possible to do this with little extra work because Krylov's spaces of A and $A + s_nI$ are the same. We assume that the solution of $A\xi = \chi$ is also needed and that $s_n > 0$ for all n. could be used. For the details of the algorithm see van der Eshof and Sleijpen, 2003. Notice that SCG always starts with $\xi_0 = 0$.

Input: M, the matrix
Input: $s[m]$, the vector of shifts
Input: χ, the right hand side
Input: n, the maximal number of iterations
Input: ϵ, required precision for $\sigma = 0$
Output: $\xi[m]$, vector of approximate solutions
Output: ξ, approximate solution for $\sigma = 0$
Output: r, final residue for $s = 0$
Output: k, number of iterations used
begin
  $k \leftarrow 0$
  $\xi \leftarrow 0$
  $\rho \leftarrow \pi \leftarrow \chi$
  $r \leftarrow \langle \rho, \rho \rangle$
  $a_{-1} \leftarrow b_{-1} \leftarrow 1$
  foreach i **do** $\xi[i] \leftarrow 0$
  foreach i **do** $\pi[i] \leftarrow \rho$
  foreach i **do** $d_{-1}[i] \leftarrow d_0[i] \leftarrow 1$
  while $k < n$ **do**
    $\omega \leftarrow M\pi$
    $z \leftarrow \langle \omega, \omega \rangle$
    $\zeta \leftarrow M^\dagger \omega$
    $a \leftarrow r/z$
    foreach i **do** $d_1[i] \leftarrow d_0[i] * (1 + a * s[i]) + a * b_{-1} * (d_0[i] - d_{-1}[i])/a_{-1}$
    $\rho \leftarrow \rho - a * \zeta$
    $g \leftarrow \langle \rho, \rho \rangle$
    $b \leftarrow g/r$
    $r \leftarrow g$
    if $r < \epsilon$ **then**
      $\xi \leftarrow \xi + a * \pi$
      foreach i **do** $\xi[i] \leftarrow \xi[i] + (a/d_1[i]) * \pi[i]$
      break
    end
    $\xi \leftarrow \xi + a * \pi$
    foreach i **do** $\xi[i] \leftarrow \xi[i] + (a/d_1[i]) * \pi[i]$
    $\pi \leftarrow \rho + b * \pi$
    foreach i **do** $\pi[i] \leftarrow \rho + (b * d_0[i]/d_1[i]) * \pi[i]$
    foreach i **do** $d_{-1}[i] \leftarrow d_0[i]$
    foreach i **do** $d_0[i] \leftarrow d_1[i]$
    $b_{-1} \leftarrow b$
    $a_{-1} \leftarrow a$
    $k \leftarrow k + 1$
  end
  return $\xi[m]$, ξ, r, k.
end

Algorithm 4: Shifted Conjugate Gradient Solver.

Chapter 3

INTERFACE

The MDWF interface is fully functional to isolate users of the code from implementation details. Several types defined in the interface provide help with typechecking.

17a ⟨*File* ../port/qop-mdwf3.h 17a⟩≡

```
#ifndef QOP_MDWF_0bd50d0caeec4311a0d7c183032c43c2
# define QOP_MDWF_0bd50d0caeec4311a0d7c183032c43c2
  ⟨Interface macros 17b⟩
  ⟨Interface types 18c⟩
  ⟨Interface functions 18a⟩
# if defined(QOP_MDWF_DEFAULT_PRECISION) && (QOP_MDWF_DEFAULT_PRECISION == 'F')
   ⟨Single precision defaults 22c⟩
# endif
# if defined(QOP_MDWF_DEFAULT_PRECISION) && (QOP_MDWF_DEFAULT_PRECISION == 'D')
    ⟨Double precision defaults 22d⟩
# endif
#endif
```

Defines:
QOP_MDWF_0bd50d0caeec4311a0d7c183032c43c2, never used.
Root chunk (not used in this document).

3.1 Magic numbers

The numbers below are provided as names for magic numbers in the code. They can not be safely changed.
First, the dimension is always four:

17b ⟨*Interface macros* 17b⟩≡

```
#define QOP_MDWF_DIM 4
```

Defines:
QOP_MDWF_DIM, used in chunk 18e.
This definition is continued in chunk 17.
This code is used in chunk 17a.

Next, the number of components in the Dirac fermion and the projected fermion

17c ⟨*Interface macros* 17b⟩+≡

```
#define QOP_MDWF_FERMION_DIM 4
#define QOP_MDWF_PROJECTED_FERMION_DIM 2
```

Defines:
QOP_MDWF_FERMION_DIM, never used.
QOP_MDWF_PROJECTED_FERMION_DIM, never used.

We work only with $SU(3)$

17d ⟨*Interface macros* 17b⟩+≡

```
#define QOP_MDWF_COLORS 3
```

Defines:
QOP_MDWF_COLORS, never used.

3.2 Library version

The following function returns a version of the library. The goal is to provide enough information to uniquely identify library's version. Since there are many features packed into the library, a human-readable string is returned.

18a ⟨*Interface functions* 18a⟩≡
```
const char *QOP_MDWF_version(void);
```
Defines:
`QOP_MDWF_version`, never used.
This definition is continued in chunks 18–33.
This code is used in chunk 17a.

3.3 Performance monitoring

Each interface function records its exectution time and number of floating point operations in the `State` structure. These numbers are accessed via the following function. Only data on the current node is recorded. The function returns `0` if performance counters are updated.

18b ⟨*Interface functions* 18a⟩+≡
```
int QOP_MDWF_performance(double *time_sec,
                         long long *flops,
                         long long *sent,
                         long long *receive,
                         struct QOP_MDWF_State *state);
```
Defines:
`QOP_MDWF_performance`, never used.
Uses `QOP_MDWF_State` 18c.

3.4 Initialization

All library state is incapsulated into an opaque structure. We do not need to expose any components of the structure to the user.

18c ⟨*Interface types* 18c⟩≡
```
struct QOP_MDWF_State;
```
Defines:
`QOP_MDWF_State`, used in chunks 18–21, 23, 25, and 27b.
This definition is continued in chunks 18d, 22a, 23b, 25b, 27a, and 30a.
This code is used in chunk 17a.

We also need an opaque type for parameters of the domain wall action.

18d ⟨*Interface types* 18c⟩+≡
```
struct QOP_MDWF_Parameters;
```
Defines:
`QOP_MDWF_Parameters`, used in chunks 18d, 20, 21a, 18d, 21b, and 28–31.

The library initialization routine creates the state structure and fills it with necessary information. It returns `0` if successful and a non-zero value otherwise. In any case `state_ptr` is set to some value suitable for other library functions.

18e ⟨*Interface functions* 18a⟩+≡
```
int QOP_MDWF_init(struct QOP_MDWF_State **state_ptr,
                  const int lattice[QOP_MDWF_DIM + 1],
                  const int network[QOP_MDWF_DIM],
                  const int node[QOP_MDWF_DIM],
                  int master_p,
                  void (*sublattice)(int lo[],
                                     int hi[],
                                     const int node[],
                                     void *env),
                  void *env);
```
Defines:
`QOP_MDWF_init`, used in chunk 18.
Uses `QOP_MDWF_DIM` 17b and `QOP_MDWF_State` 18c.

Arguments of `init()` are

`state_ptr` points to the `State` to be set.

`lattice` 5–d lattice geometry. L_s is `lattice[4]`.

`network` 4–d network geometry. Each element should be positive.

`node` 4–d address of this node.

`master_p` is non-zero on the master node, zero otherwise.

`sublattice` Function giving sublattice dimensions on an arbitrary node.

`env` Common environment for `sublattice()`.

When `QOP_MDWF_init()` returns, `*state_ptr` will point to a valid state of the library (if any error occurs during initialization, the error will be stored in `*ptr`. The `lattice[]` provides the total lattice size, and `sublattice` is a pointer to a function that given a 4-d node address `node` returns in `lo` and `hi` the sublattice coordinates local to the `node`, in effect saying that the local sublattice starts at `x[i]=lo[i]` and ends at `x[i]=hi[i]-1`.
It is possible to call `QOP_MDWF_init()` multiple times with different arguments. The library does not require that the lattice size and layout agree in different calls.

3.5 Cleanup

When the state is no longer needed it should be closed by the following function

19a ⟨*Interface functions* 18a⟩+≡
```
void QOP_MDWF_fini(struct QOP_MDWF_State **state_ptr);
```
Defines:
`QOP_MDWF_fini`, never used.
Uses `QOP_MDWF_State` 18c.

It is an error to use `*state_ptr` after it was closed. To help in error detection this function sets `*state_ptr` to `NULL`. All library functions check if the state they are passed is `NULL` and abort if it is.

3.6 Errors

When something goes wrong in the library, a library function will return some non-zero value and store the error code in the library state. The error codes are accessible as human-readable strings via the following function:

19b ⟨*Interface functions* 18a⟩+≡
```
const char *QOP_MDWF_error(struct QOP_MDWF_State *state);
```
Defines:
`QOP_MDWF_error`, used in chunk 19.
Uses `QOP_MDWF_State` 18c.

Note that the first error will be latched until `QOP_MDWF_error()` is called. This is a design choice made to help in pinpointing the origin of the problem when something goes wrong intead of reporting spiritious errors if multiple calls to the library are made before an error is checked for. If there is no error, `QOP_MDWF_error()` returns `NULL`. The function could be called multiple times, it does not reset the error code, instead it marks in the state that the error was reported thus allowing latching another error.

3.7 Parameter setting

Following functions set parameters of the MDWF into the state. These functions allocate `QOP_MDWF_Parameters` structure.

20a ⟨*Interface functions* 18a⟩+≡

```
int QOP_MDWF_set_generic(struct QOP_MDWF_Parameters **param_ptr,
                         struct QOP_MDWF_State *state,
                         const double b_5[],
                         const double c_5[],
                         double M_5,
                         double m);
```

Defines:
`QOP_MDWF_set_generic`, never used.
Uses `QOP_MDWF_Parameters` 18d and `QOP_MDWF_State` 18c.

As a convenience, specialized setups are provided as well. For Möbius fermions, $b_5(s) + c_5(s) = \kappa$:

20b ⟨*Interface functions* 18a⟩+≡

```
int QOP_MDWF_set_Moebius(struct QOP_MDWF_Parameters **param_ptr,
                         struct QOP_MDWF_State *state,
                         const double b_5[],
                         double kappa,
                         double M_5,
                         double m);
```

Defines:
`QOP_MDWF_set_Moebius`, never used.
Uses `QOP_MDWF_Parameters` 18d and `QOP_MDWF_State` 18c.

For Shamir fermions, $b_5(s) = a_5$, $c_5 = 0$:

20c ⟨*Interface functions* 18a⟩+≡

```
int QOP_MDWF_set_Shamir(struct QOP_MDWF_Parameters **param_ptr,
                        struct QOP_MDWF_State *state,
                        double a_5,
                        double M_5,
                        double m);
```

Defines:
`QOP_MDWF_set_Shamir`, never used.
Uses `QOP_MDWF_Parameters` 18d and `QOP_MDWF_State` 18c.

For Boriçi, $b_5(s) = c_5(s) = a_5$:

20d ⟨*Interface functions* 18a⟩+≡

```
int QOP_MDWF_set_Borichi(struct QOP_MDWF_Parameters **param_ptr,
                         struct QOP_MDWF_State *state,
                         double a_5,
                         double M_5,
                         double m);
```

Defines:
`QOP_MDWF_set_Borichi`, never used.
Uses `QOP_MDWF_Parameters` 18d and `QOP_MDWF_State` 18c.

For Chiu, $b_5(s) = c_5(s) = a_5(s)$:

21a ⟨*Interface functions* 18a⟩+≡

```
int QOP_MDWF_set_Chiu(struct QOP_MDWF_Parameters **param_ptr,
                      struct QOP_MDWF_State *state,
                      const double a_5[],
                      double M_5,
                      double m);
```

Defines:
  QOP_MDWF_set_Chiu, never used.
Uses QOP_MDWF_Parameters 18d and QOP_MDWF_State 18c.

We also provide a corresponding destructor for QOP_MDWF_Parameters. This function will write NULL back to the pointer to help in bug detection.

21b ⟨*Interface functions* 18a⟩+≡

```
void QOP_MDWF_free_parameters(struct QOP_MDWF_Parameters **param_ptr);
```

Defines:
  QOP_MDWF_free_parameters, never used.
Uses QOP_MDWF_Parameters 18d.

3.8 Gauge

Any gauge field should be imported into the library format before it could be used. To keep the interface as general as possible, we use a query function approach for inport. There are two versions of QOP_MDWF_import_gauge, one for double precision, another for single precision.

21c ⟨*Interface functions* 18a⟩+≡

```
int QOP_F3_MDWF_import_gauge(struct QOP_F3_MDWF_Gauge **gauge_ptr,
                             struct QOP_MDWF_State *state,
                             double (*reader)(int dir,
                                              const int pos[4],
                                              int a,
                                              int b,
                                              int re_im,
                                              void *env),
                             void *env);
int QOP_D3_MDWF_import_gauge(struct QOP_D3_MDWF_Gauge **gauge_ptr,
                             struct QOP_MDWF_State *state,
                             double (*reader)(int dir,
                                              const int pos[4],
                                              int a,
                                              int b,
                                              int re_im,
                                              void *env),
                             void *env);
```

Defines:
  QOP_D3_MDWF_import_gauge, used in chunk 22d.
  QOP_F3_MDWF_import_gauge, used in chunk 22c.
Uses QOP_D3_MDWF_Gauge 22a, QOP_F3_MDWF_Gauge 22a, and QOP_MDWF_State 18c.

The `reader()` points to a function that provides a value of the gauge field at a given point on the lattice, e.g., it returns the value of `U[dir][pos][a][b].re` for `re_im==0` and `U[dir][pos][a][b].im` for `re_im==1`. If will be called only for `pos` in a local sublattice. The `reader()` function is passed the `env` parameter that may be used to access the guage field from the outer space. The `env` parameter is not used by `import_gauge()` functions for any other purpose.
If the function succeeds then the `*gauge_ptr` will be initialized to a value that may be passed to other library functions. If something goes wrong, `*gauge_ptr` will be set to `NULL`.
Here are corresponding opaque types:

22a ⟨*Interface types* 18c⟩+≡
```
struct QOP_F3_MDWF_Gauge;
struct QOP_D3_MDWF_Gauge;
```
Defines:
`QOP_D3_MDWF_Gauge`, used in chunks 21, 22, and 28–31.
`QOP_F3_MDWF_Gauge`, used in chunks 21, 22, and 28–31.

We also need a couple of destructors for gauge fields. For convenience, they will accept `NULL` intead of a valid gauge field.

22b ⟨*Interface functions* 18a⟩+≡
```
void QOP_F3_MDWF_free_gauge(struct QOP_F3_MDWF_Gauge **gauge_ptr);
void QOP_D3_MDWF_free_gauge(struct QOP_D3_MDWF_Gauge **gauge_ptr);
```
Defines:
`QOP_D3_MDWF_free_gauge`, used in chunk 22d.
`QOP_F3_MDWF_free_gauge`, used in chunk 22c.
Uses `QOP_D3_MDWF_Gauge` 22a and `QOP_F3_MDWF_Gauge` 22a.

Here are macros defining default values for gauge field types and functions:

22c ⟨*Single precision defaults* 22c⟩≡
```
#define QOP_MDWF_import_gauge QOP_F3_MDWF_import_gauge
#define QOP_MDWF_free_gauge QOP_F3_MDWF_free_gauge
#define QOP_MDWF_Gauge QOP_F3_MDWF_Gauge
```
Defines:
`QOP_MDWF_Gauge`, never used.
`QOP_MDWF_free_gauge`, never used.
`QOP_MDWF_import_gauge`, used in chunk 22c.
Uses `QOP_F3_MDWF_Gauge` 22a, `QOP_F3_MDWF_free_gauge` 22b, and `QOP_F3_MDWF_import_gauge` 21c.
This definition is continued in chunks 24c, 26c, 28a, 29c, 32a, and 33c.
This code is used in chunk 17a.

22d ⟨*Double precision defaults* 22d⟩≡
```
#define QOP_MDWF_import_gauge QOP_D3_MDWF_import_gauge
#define QOP_MDWF_free_gauge QOP_D3_MDWF_free_gauge
#define QOP_MDWF_Gauge QOP_D3_MDWF_Gauge
```
Defines:
`QOP_MDWF_Gauge`, never used.
`QOP_MDWF_free_gauge`, never used.
`QOP_MDWF_import_gauge`, used in chunk 22c.
Uses `QOP_D3_MDWF_Gauge` 22a, `QOP_D3_MDWF_free_gauge` 22b, and `QOP_D3_MDWF_import_gauge` 21c.
This definition is continued in chunks 24d, 26d, 28b, 29d, 32b, and 34.
This code is used in chunk 17a.

3.9 Fermions

Unlike the gauge field, fermions are provided with a richer set of functions. In addition to import and destruction, they could be created empty and exported.

23a ⟨*Interface functions* 18a⟩+≡

```
int QOP_F3_MDWF_import_fermion(struct QOP_F3_MDWF_Fermion **fermion_ptr,
                               struct QOP_MDWF_State *state,
                               double (*reader)(const int pos[5],
                                                int color,
                                                int dirac,
                                                int re_im,
                                                void *env),
                               void *env);
int QOP_D3_MDWF_import_fermion(struct QOP_D3_MDWF_Fermion **fermion_ptr,
                               struct QOP_MDWF_State *state,
                               double (*reader)(const int pos[5],
                                                int color,
                                                int dirac,
                                                int re_im,
                                                void *env),
                               void *env);
```

Defines:
`QOP_D3_MDWF_import_fermion`, used in chunk 24d.
`QOP_F3_MDWF_import_fermion`, used in chunk 24c.
Uses `QOP_D3_MDWF_Fermion` 23b, `QOP_F3_MDWF_Fermion` 23b, and `QOP_MDWF_State` 18c.

The `reader()` points to a function that provides a value of the fermion field at a given point on the lattice, e.g., it returns the value of `F[pos][color][dirac].re` for `re_im==0` and `F[pos][color][dirac].im` for `re_im==1`. If will be called only for `pos` in a local sublattice. The `env` parameter is passed blindly to the `reader()` without any interpretation whatsoever. It could be used to access the fermion field from the outer space or for any other purpose.
If the function succeeds then the `*fermion_ptr` will be initialized to a value that may be passed to other library functions. If something goes wrong, `*fermion_ptr` will be set to `NULL`.
Here are corresponding opaque types:

23b ⟨*Interface types* 18c⟩+≡

```
struct QOP_F3_MDWF_Fermion;
struct QOP_D3_MDWF_Fermion;
```

Defines:
`QOP_D3_MDWF_Fermion`, used in chunks 23, 24, 28, 30b, 32, and 33.
`QOP_F3_MDWF_Fermion`, used in chunks 23, 24, 28, 30b, 32, and 33.

One may also need to create a fermion field without any useful initial value. For convenience, we provide functions to do that

23c ⟨*Interface functions* 18a⟩+≡

```
int QOP_F3_MDWF_allocate_fermion(struct QOP_F3_MDWF_Fermion **fermion_ptr,
                                 struct QOP_MDWF_State *state);
int QOP_D3_MDWF_allocate_fermion(struct QOP_D3_MDWF_Fermion **fermion_ptr,
                                 struct QOP_MDWF_State *state);
```

Defines:
`QOP_D3_MDWF_allocate_fermion`, used in chunk 24d.
`QOP_F3_MDWF_allocate_fermion`, used in chunk 24c.
Uses `QOP_D3_MDWF_Fermion` 23b, `QOP_F3_MDWF_Fermion` 23b, and `QOP_MDWF_State` 18c.

Unlike the gauge fields, fermions need a way to be exported back to the user. We also use a functional interface to provide indexing.

24a ⟨*Interface functions* 18a⟩+≡

```
int QOP_F3_MDWF_export_fermion(void (*writer)(const int pos[5],
                                              int color,
                                              int dirac,
                                              int re_im,
                                              double value,
                                              void *env),
                               void *env,
                               const struct QOP_F3_MDWF_Fermion *fermion);
int QOP_D3_MDWF_export_fermion(void (*writer)(const int pos[5],
                                              int color,
                                              int dirac,
                                              int re_im,
                                              double value,
                                              void *env),
                               void *env,
                               const struct QOP_D3_MDWF_Fermion *fermion);
```

Defines:
QOP_D3_MDWF_export_fermion, used in chunk 24d.
QOP_F3_MDWF_export_fermion, used in chunk 24c.
Uses QOP_D3_MDWF_Fermion 23b and QOP_F3_MDWF_Fermion 23b.

We also need a couple of destructors for fermion fields. For convenience, they will accept NULL intead of a valid fermion field.

24b ⟨*Interface functions* 18a⟩+≡

```
void QOP_F3_MDWF_free_fermion(struct QOP_F3_MDWF_Fermion **fermion_ptr);
void QOP_D3_MDWF_free_fermion(struct QOP_D3_MDWF_Fermion **fermion_ptr);
```

Defines:
QOP_D3_MDWF_free_fermion, used in chunk 24d.
QOP_F3_MDWF_free_fermion, used in chunk 24c.
Uses QOP_D3_MDWF_Fermion 23b and QOP_F3_MDWF_Fermion 23b.

Finally, macros for prefered precision

24c ⟨*Single precision defaults* 22c⟩+≡

```
#define QOP_MDWF_import_fermion QOP_F3_MDWF_import_fermion
#define QOP_MDWF_export_fermion QOP_F3_MDWF_export_fermion
#define QOP_MDWF_allocate_fermion QOP_F3_MDWF_allocate_fermion
#define QOP_MDWF_free_fermion QOP_F3_MDWF_free_fermion
#define QOP_MDWF_Fermion QOP_F3_MDWF_Fermion
```

Defines:
QOP_MDWF_Fermion, never used.
QOP_MDWF_allocate_fermion, never used.
QOP_MDWF_export_fermion, never used.
QOP_MDWF_free_fermion, never used.
QOP_MDWF_import_fermion, never used.
Uses QOP_F3_MDWF_Fermion 23b, QOP_F3_MDWF_allocate_fermion 23c, QOP_F3_MDWF_export_fermion 24a, QOP_F3_MDWF_free_fermion 24b, and QOP_F3_MDWF_import_fermion 23a.

24d ⟨*Double precision defaults* 22d⟩+≡

```
#define QOP_MDWF_import_fermion QOP_D3_MDWF_import_fermion
#define QOP_MDWF_export_fermion QOP_D3_MDWF_export_fermion
#define QOP_MDWF_allocate_fermion QOP_D3_MDWF_allocate_fermion
#define QOP_MDWF_free_fermion QOP_D3_MDWF_free_fermion
#define QOP_MDWF_Fermion QOP_D3_MDWF_Fermion
```

Defines:
QOP_MDWF_Fermion, never used.
QOP_MDWF_allocate_fermion, never used.
QOP_MDWF_export_fermion, never used.
QOP_MDWF_free_fermion, never used.
QOP_MDWF_import_fermion, never used.
Uses QOP_D3_MDWF_Fermion 23b, QOP_D3_MDWF_allocate_fermion 23c, QOP_D3_MDWF_export_fermion 24a, QOP_D3_MDWF_free_fermion 24b, and QOP_D3_MDWF_import_fermion 23a.

3.10 Preconditioned fermions

We also need preconditioned fermions. They exist in parallel to full fermions but the exact relation is not specified. Unlike the gauge field, fermions are provided with a richer set of functions. In addition to import and destruction, they could be created empty and exported.

25a ⟨*Interface functions* 18a⟩+≡

```
int QOP_F3_MDWF_import_half_fermion(struct QOP_F3_MDWF_HalfFermion **hfermion_ptr,
                                    struct QOP_MDWF_State *state,
                                    double (*reader)(const int pos[5],
                                                     int color,
                                                     int dirac,
                                                     int re_im,
                                                     void *env),
                                    void *env);
int QOP_D3_MDWF_import_half_fermion(struct QOP_D3_MDWF_HalfFermion **hfermion_ptr,
                                    struct QOP_MDWF_State *state,
                                    double (*reader)(const int pos[5],
                                                     int color,
                                                     int dirac,
                                                     int re_im,
                                                     void *env),
                                    void *env);
```

Defines:
`QOP_D3_MDWF_import_half_fermion`, used in chunk 26d.
`QOP_F3_MDWF_import_half_fermion`, used in chunk 26c.
Uses `QOP_D3_MDWF_HalfFermion` 25b, `QOP_F3_MDWF_HalfFermion` 25b, and `QOP_MDWF_State` 18c.

The `reader()` points to a function that provides a value of the preconditioned fermion field at a given point on the lattice, e.g., it returns the value of `F[pos][color][dirac].re` for `re_im==0` and `F[pos][color][dirac].im` for `re_im==1`. If will be called only for `pos` in a local sublattice. The `env` parameter is passed blindly to the `reader()` without any interpretation whatsoever. It could be used to access the half fermion in the calling layer.
If the function succeeds then the `*hfermion_ptr` will be initialized to a value that may be passed to other library functions. If something goes wrong, `*hfermion_ptr` will be set to `NULL`.
Here are corresponding opaque types:

25b ⟨*Interface types* 18c⟩+≡

```
struct QOP_F3_MDWF_HalfFermion;
struct QOP_D3_MDWF_HalfFermion;
```

Defines:
`QOP_D3_MDWF_HalfFermion`, used in chunks 25–27, 29, and 31–33.
`QOP_F3_MDWF_HalfFermion`, used in chunks 25–27, 29, and 31–33.

One may also need to create a fermion field without any useful initial value. For convenience, we provide functions to do that

25c ⟨*Interface functions* 18a⟩+≡

```
int QOP_F3_MDWF_allocate_half_fermion(struct QOP_F3_MDWF_HalfFermion **hfermion_ptr,
                                      struct QOP_MDWF_State *state);
int QOP_D3_MDWF_allocate_half_fermion(struct QOP_D3_MDWF_HalfFermion **hfermion_ptr,
                                      struct QOP_MDWF_State *state);
```

Defines:
`QOP_D3_MDWF_allocate_half_fermion`, used in chunk 26d.
`QOP_F3_MDWF_allocate_half_fermion`, used in chunk 26c.
Uses `QOP_D3_MDWF_HalfFermion` 25b, `QOP_F3_MDWF_HalfFermion` 25b, and `QOP_MDWF_State` 18c.

Preconditioned fermions may be exported back to the user. We also use a functional interface to provide indexing.

26a ⟨*Interface functions* 18a⟩+≡

```
int QOP_F3_MDWF_export_half_fermion(void (*writer)(const int pos[5],
                                                   int color,
                                                   int dirac,
                                                   int re_im,
                                                   double value,
                                                   void *env),
                                    void *env,
                                    const struct QOP_F3_MDWF_HalfFermion *hfermion);
int QOP_D3_MDWF_export_half_fermion(void (*writer)(const int pos[5],
                                                   int color,
                                                   int dirac,
                                                   int re_im,
                                                   double value,
                                                   void *env),
                                    void *env,
                                    const struct QOP_D3_MDWF_HalfFermion *hfermion);
```

Defines:
QOP_D3_MDWF_export_half_fermion, used in chunk 26d.
QOP_F3_MDWF_export_half_fermion, used in chunk 26c.
Uses QOP_D3_MDWF_HalfFermion 25b and QOP_F3_MDWF_HalfFermion 25b.

We also need a couple of destructors for fermion fields. For convenience, they will accept NULL intead of a valid fermion field.

26b ⟨*Interface functions* 18a⟩+≡

```
void QOP_F3_MDWF_free_half_fermion(struct QOP_F3_MDWF_HalfFermion **hfermion_ptr);
void QOP_D3_MDWF_free_half_fermion(struct QOP_D3_MDWF_HalfFermion **hfermion_ptr);
```

Defines:
QOP_D3_MDWF_free_half_fermion, used in chunk 26d.
QOP_F3_MDWF_free_half_fermion, used in chunk 26c.
Uses QOP_D3_MDWF_HalfFermion 25b and QOP_F3_MDWF_HalfFermion 25b.

Finally, macros for prefered precision

26c ⟨*Single precision defaults* 22c⟩+≡

```
#define QOP_MDWF_import_half_fermion QOP_F3_MDWF_import_half_fermion
#define QOP_MDWF_export_half_fermion QOP_F3_MDWF_export_half_fermion
#define QOP_MDWF_allocate_half_fermion QOP_F3_MDWF_allocate_half_fermion
#define QOP_MDWF_free_half_fermion QOP_F3_MDWF_free_half_fermion
#define QOP_MDWF_HalfFermion QOP_F3_MDWF_HalfFermion
```

Defines:
QOP_MDWF_HalfFermion, never used.
QOP_MDWF_allocate_half_fermion, never used.
QOP_MDWF_export_half_fermion, never used.
QOP_MDWF_free_half_fermion, never used.
QOP_MDWF_import_half_fermion, never used.
Uses QOP_F3_MDWF_HalfFermion 25b, QOP_F3_MDWF_allocate_half_fermion 25c, QOP_F3_MDWF_export_half_fermion 26a, QOP_F3_MDWF_free_half_fermion 26b, and QOP_F3_MDWF_import_half_fermion 25a.

26d ⟨*Double precision defaults* 22d⟩+≡

```
#define QOP_MDWF_import_half_fermion QOP_D3_MDWF_import_half_fermion
#define QOP_MDWF_export_half_fermion QOP_D3_MDWF_export_half_fermion
#define QOP_MDWF_allocate_half_fermion QOP_D3_MDWF_allocate_half_fermion
#define QOP_MDWF_free_half_fermion QOP_D3_MDWF_free_half_fermion
#define QOP_MDWF_HalfFermion QOP_D3_MDWF_HalfFermion
```

Defines:
QOP_MDWF_HalfFermion, never used.
QOP_MDWF_allocate_half_fermion, never used.
QOP_MDWF_export_half_fermion, never used.
QOP_MDWF_free_half_fermion, never used.
QOP_MDWF_import_half_fermion, never used.
Uses QOP_D3_MDWF_HalfFermion 25b, QOP_D3_MDWF_allocate_half_fermion 25c, QOP_D3_MDWF_export_half_fermion 26a, QOP_D3_MDWF_free_half_fermion 26b, and QOP_D3_MDWF_import_half_fermion 25a.

3.11 Fermion Vectors

For shifted conjugate gradient solver we need vectors of preconditioned fermions. To the user only an opaque data type is presented.

27a ⟨*Interface types* 18c⟩+≡
```
struct QOP_F3_MDWF_VectorFermion;
struct QOP_D3_MDWF_VectorFermion;
```

First, the allocator. It takes a number of elements in the vector and returns an opaque vector fermion object.

27b ⟨*Interface functions* 18a⟩+≡
```
int QOP_F3_MDWF_allocate_vector_fermion(struct QOP_F3_MDWF_VectorFermion **vf_ptr,
                                        struct QOP_MDWF_State *state,
                                        int n);
int QOP_D3_MDWF_allocate_vector_fermion(struct QOP_D3_MDWF_VectorFermion **vf_ptr,
                                       struct QOP_MDWF_State *state,
                                       int n);
```

Defines:
QOP_D3_MDWF_allocate_vector_fermion, used in chunk 28b.
QOP_F3_MDWF_allocate_vector_fermion, used in chunk 28a.
Uses QOP_MDWF_State 18c.

Destructors of the vector fermions accept NULL in addition to a valid vector fermion pointer.

27c ⟨*Interface functions* 18a⟩+≡
```
void QOP_F3_MDWF_free_vector_fermion(struct QOP_F3_MDWF_VectorFermion **vf_ptr);
void QOP_D3_MDWF_free_vector_fermion(struct QOP_D3_MDWF_VectorFermion **vf_ptr);
```

Defines:
QOP_D3_MDWF_free_vector_fermion, used in chunk 28b.
QOP_F3_MDWF_free_vector_fermion, used in chunk 28a.

We also provide primitive access procedures (mnemonic convention is “get from vector” and “put into vector”):

27d ⟨*Interface functions* 18a⟩+≡
```
int QOP_F3_MDWF_get_vector_fermion(struct QOP_F3_MDWF_HalfFermion *hf,
                                   const struct QOP_F3_MDWF_VectorFermion *vf,
                                   int index);
int QOP_D3_MDWF_get_vector_fermion(struct QOP_D3_MDWF_HalfFermion *hf,
                                   const struct QOP_D3_MDWF_VectorFermion *vf,
                                   int index);
int QOP_F3_MDWF_put_vector_fermion(struct QOP_F3_MDWF_VectorFermion *vf,
                                   int index,
                                   const struct QOP_F3_MDWF_HalfFermion *hf);
int QOP_D3_MDWF_put_vector_fermion(struct QOP_D3_MDWF_VectorFermion *vf,
                                   int index,
                                   const struct QOP_D3_MDWF_HalfFermion *hf);
```

Defines:
QOP_D3_MDWF_get_vector_fermion, used in chunk 28b.
QOP_D3_MDWF_put_vector_fermion, used in chunk 28b.
QOP_F3_MDWF_get_vector_fermion, used in chunk 28a.
QOP_F3_MDWF_put_vector_fermion, used in chunk 28a.
Uses QOP_D3_MDWF_HalfFermion 25b and QOP_F3_MDWF_HalfFermion 25b.

Macros for default precision:

28a ⟨*Single precision defaults* 22c⟩+≡

```
#define QOP_MDWF_allocate_vector_fermion QOP_F3_MDWF_allocate_vector_fermion
#define QOP_MDWF_free_vector_fermion QOP_F3_MDWF_free_vector_fermion
#define QOP_MDWF_get_vector_fermion QOP_F3_MDWF_get_vector_fermion
#define QOP_MDWF_put_vector_fermion QOP_F3_MDWF_put_vector_fermion
#define QOP_MDWF_VectorFermion QOP_F3_MDWF_VectorFermion
```

Defines:
QOP_MDWF_VectorFermion, never used.
QOP_MDWF_allocate_vector_fermion, never used.
QOP_MDWF_free_vector_fermion, never used.
QOP_MDWF_get_vector_fermion, never used.
QOP_MDWF_put_vector_fermion, never used.
Uses QOP_F3_MDWF_allocate_vector_fermion 27b, QOP_F3_MDWF_free_vector_fermion 27c, QOP_F3_MDWF_get_vector_fermion 27d, and QOP_F3_MDWF_put_vector_fermion 27d.

28b ⟨*Double precision defaults* 22d⟩+≡

```
#define QOP_MDWF_allocate_vector_fermion QOP_D3_MDWF_allocate_vector_fermion
#define QOP_MDWF_free_vector_fermion QOP_D3_MDWF_free_vector_fermion
#define QOP_MDWF_get_vector_fermion QOP_D3_MDWF_get_vector_fermion
#define QOP_MDWF_put_vector_fermion QOP_D3_MDWF_put_vector_fermion
#define QOP_MDWF_VectorFermion QOP_D3_MDWF_VectorFermion
```

Defines:
QOP_MDWF_VectorFermion, never used.
QOP_MDWF_allocate_vector_fermion, never used.
QOP_MDWF_free_vector_fermion, never used.
QOP_MDWF_get_vector_fermion, never used.
QOP_MDWF_put_vector_fermion, never used.
Uses QOP_D3_MDWF_allocate_vector_fermion 27b, QOP_D3_MDWF_free_vector_fermion 27c, QOP_D3_MDWF_get_vector_fermion 27d, and QOP_D3_MDWF_put_vector_fermion 27d.

3.12 Dirac Operator

We provide both normal and conjugated Dirac Operator for the full fermion as well as the precondition operator and its conjugate both in single and double precision.

28c ⟨*Interface functions* 18a⟩+≡

```
int QOP_F3_MDWF_DDW_operator(struct QOP_F3_MDWF_Fermion *result,
                             const struct QOP_MDWF_Parameters *params,
                             const struct QOP_F3_MDWF_Gauge *gauge,
                             const struct QOP_F3_MDWF_Fermion *fermion);
int QOP_D3_MDWF_DDW_operator(struct QOP_D3_MDWF_Fermion *result,
                             const struct QOP_MDWF_Parameters *params,
                             const struct QOP_D3_MDWF_Gauge *gauge,
                             const struct QOP_D3_MDWF_Fermion *fermion);
```

Defines:
QOP_D3_MDWF_DDW_operator, used in chunk 29d.
QOP_F3_MDWF_DDW_operator, used in chunk 29c.
Uses QOP_D3_MDWF_Fermion 23b, QOP_D3_MDWF_Gauge 22a, QOP_F3_MDWF_Fermion 23b, QOP_F3_MDWF_Gauge 22a, and QOP_MDWF_Parameters 18d.

28d ⟨*Interface functions* 18a⟩+≡

```
int QOP_F3_MDWF_DDW_operator_conjugated(struct QOP_F3_MDWF_Fermion *result,
                                        const struct QOP_MDWF_Parameters *params,
                                        const struct QOP_F3_MDWF_Gauge *gauge,
                                        const struct QOP_F3_MDWF_Fermion *fermion);
int QOP_D3_MDWF_DDW_operator_conjugated(struct QOP_D3_MDWF_Fermion *result,
                                        const struct QOP_MDWF_Parameters *params,
                                        const struct QOP_D3_MDWF_Gauge *gauge,
                                        const struct QOP_D3_MDWF_Fermion *fermion);
```

Defines:
QOP_D3_MDWF_DDW_operator_conjugated, used in chunk 29d.
QOP_F3_MDWF_DDW_operator_conjugated, used in chunk 29c.
Uses QOP_D3_MDWF_Fermion 23b, QOP_D3_MDWF_Gauge 22a, QOP_F3_MDWF_Fermion 23b, QOP_F3_MDWF_Gauge 22a, and QOP_MDWF_Parameters 18d.

29a ⟨*Interface functions* 18a⟩+≡

```
int QOP_F3_MDWF_M_operator(struct QOP_F3_MDWF_HalfFermion *result,
                           const struct QOP_MDWF_Parameters *params,
                           const struct QOP_F3_MDWF_Gauge *gauge,
                           const struct QOP_F3_MDWF_HalfFermion *fermion);
int QOP_D3_MDWF_M_operator(struct QOP_D3_MDWF_HalfFermion *result,
                           const struct QOP_MDWF_Parameters *params,
                           const struct QOP_D3_MDWF_Gauge *gauge,
                           const struct QOP_D3_MDWF_HalfFermion *fermion);
```

Defines:
QOP_D3_MDWF_M_operator, used in chunk 29d.
QOP_F3_MDWF_M_operator, used in chunk 29c.
Uses QOP_D3_MDWF_Gauge 22a, QOP_D3_MDWF_HalfFermion 25b, QOP_F3_MDWF_Gauge 22a, QOP_F3_MDWF_HalfFermion 25b, and QOP_MDWF_Parameters 18d.

29b ⟨*Interface functions* 18a⟩+≡

```
int QOP_F3_MDWF_M_operator_conjugated(struct QOP_F3_MDWF_HalfFermion *result,
                                      const struct QOP_MDWF_Parameters *params,
                                      const struct QOP_F3_MDWF_Gauge *gauge,
                                      const struct QOP_F3_MDWF_HalfFermion *fermion);
int QOP_D3_MDWF_M_operator_conjugated(struct QOP_D3_MDWF_HalfFermion *result,
                                      const struct QOP_MDWF_Parameters *params,
                                      const struct QOP_D3_MDWF_Gauge *gauge,
                                      const struct QOP_D3_MDWF_HalfFermion *fermion);
```

Defines:
QOP_D3_MDWF_M_operator_conjugated, used in chunk 29d.
QOP_F3_MDWF_M_operator_conjugated, used in chunk 29c.
Uses QOP_D3_MDWF_Gauge 22a, QOP_D3_MDWF_HalfFermion 25b, QOP_F3_MDWF_Gauge 22a, QOP_F3_MDWF_HalfFermion 25b, and QOP_MDWF_Parameters 18d.

Also the default precision macros

29c ⟨*Single precision defaults* 22c⟩+≡

```
#define QOP_MDWF_DDW_operator QOP_F3_MDWF_DDW_operator
#define QOP_MDWF_DDW_operator_conjugated QOP_F3_MDWF_DDW_operator_conjugated
#define QOP_MDWF_M_operator QOP_F3_MDWF_M_operator
#define QOP_MDWF_M_operator_conjugated QOP_F3_MDWF_M_operator_conjugated
```

Defines:
QOP_MDWF_DDW_operator, never used.
QOP_MDWF_DDW_operator_conjugated, never used.
QOP_MDWF_M_operator, never used.
QOP_MDWF_M_operator_conjugated, never used.
Uses QOP_F3_MDWF_DDW_operator 28c, QOP_F3_MDWF_DDW_operator_conjugated 28d, QOP_F3_MDWF_M_operator 29a, and QOP_F3_MDWF_M_operator_conjugated 29b.

29d ⟨*Double precision defaults* 22d⟩+≡

```
#define QOP_MDWF_DDW_operator QOP_D3_MDWF_DDW_operator
#define QOP_MDWF_DDW_operator_conjugated QOP_D3_MDWF_DDW_operator_conjugated
#define QOP_MDWF_M_operator QOP_D3_MDWF_M_operator
#define QOP_MDWF_M_operator_conjugated QOP_D3_MDWF_M_operator_conjugated
```

Defines:
QOP_MDWF_DDW_operator, never used.
QOP_MDWF_DDW_operator_conjugated, never used.
QOP_MDWF_M_operator, never used.
QOP_MDWF_M_operator_conjugated, never used.
Uses QOP_D3_MDWF_DDW_operator 28c, QOP_D3_MDWF_DDW_operator_conjugated 28d, QOP_D3_MDWF_M_operator 29a, and QOP_D3_MDWF_M_operator_conjugated 29b.

3.13 Solvers

We provide three solvers for the Dirac operator. All solvers can optionally compute true CG and Dirac residuals on each iteration. Constants below can be bitwise combined to select which residual are computed and printed. The behavior of the solvers do not change when residuals are selected. Applications should not assume particular values of the constants except that `QOP_MDWF_LOG_NONE` is zero.

30a ⟨*Interface types* 18c⟩+≡

```
enum {
   QOP_MDWF_LOG_NONE = 0,
   QOP_MDWF_LOG_CG_RESIDUAL = 1,
   QOP_MDWF_LOG_TRUE_RESIDUAL = 2,
   QOP_MDWF_LOG_DIRAC_RESIDUAL = 4,
   QOP_MDWF_FINAL_CG_RESIDUAL = 8,
   QOP_MDWF_FINAL_DIRAC_RESIDUAL = 16,
   QOP_MDWF_LOG_EVERYTHING = 31
};
```

Defines:
`QOP_MDWF_FINAL_CG_RESIDUAL`, never used.
`QOP_MDWF_FINAL_DIRAC_RESIDUAL`, never used.
`QOP_MDWF_LOG_CG_RESIDUAL`, never used.
`QOP_MDWF_LOG_DIRAC_RESIDUAL`, never used.
`QOP_MDWF_LOG_EVERYTHING`, never used.
`QOP_MDWF_LOG_NONE`, used in chunk 30a.
`QOP_MDWF_LOG_TRUE_RESIDUAL`, never used.

First, the convenience routine to solve $D_{DW}\psi = \eta$. At most `max_iterations` are performed, the CG stops when the iterative precontioned residue becomes ϵ or less. If conjugate gradient converges, zero is returned. In this case `out_iterations` contains the number of iterations and `out_epsilon` contains normalized iterative CG residual.

30b ⟨*Interface functions* 18a⟩+≡

```
int QOP_F3_MDWF_DDW_CG(struct QOP_F3_MDWF_Fermion *result,
                       int *out_iterations,
                       double *out_epsilon,
                       const struct QOP_MDWF_Parameters *params,
                       const struct QOP_F3_MDWF_Fermion *chi_0,
                       const struct QOP_F3_MDWF_Gauge *gauge,
                       const struct QOP_F3_MDWF_Fermion *rhs,
                       int max_iteration,
                       double epsilon,
                       unsigned int options);
int QOP_D3_MDWF_DDW_CG(struct QOP_D3_MDWF_Fermion *result,
                       int *out_iterations,
                       double *out_epsilon,
                       const struct QOP_MDWF_Parameters *params,
                       const struct QOP_D3_MDWF_Fermion *x_0,
                       const struct QOP_D3_MDWF_Gauge *gauge,
                       const struct QOP_D3_MDWF_Fermion *rhs,
                       int max_iteration,
                       double epsilon,
                       unsigned int options);
```

Defines:
`QOP_D3_MDWF_DDW_CG`, used in chunk 32b.
`QOP_F3_MDWF_DDW_CG`, used in chunk 32a.
Uses `QOP_D3_MDWF_Fermion` 23b, `QOP_D3_MDWF_Gauge` 22a, `QOP_F3_MDWF_Fermion` 23b, `QOP_F3_MDWF_Gauge` 22a, and `QOP_MDWF_Parameters` 18d.

We also expose the preconditioned hermitian solver for $M^\dagger M \psi_e = \phi_e$. In this case the CG starts from ψ_e. If conjugate gradient converges, zero is returned. In this case `out_iterations` contains the number of iterations and `out_epsilon` contains normalized iterative CG residual.

31a ⟨*Interface functions* 18a⟩+≡

```
int QOP_F3_MDWF_MxM_CG(struct QOP_F3_MDWF_HalfFermion *result,
                       int *out_iterations,
                       double *out_epsilon,
                       const struct QOP_MDWF_Parameters *params,
                       const struct QOP_F3_MDWF_Gauge *gauge,
                       const struct QOP_F3_MDWF_HalfFermion *rhs,
                       int max_iteration,
                       double epsilon,
                       unsigned int options);
int QOP_D3_MDWF_MxM_CG(struct QOP_D3_MDWF_HalfFermion *result,
                       int *out_iterations,
                       double *out_epsilon,
                       const struct QOP_MDWF_Parameters *params,
                       const struct QOP_D3_MDWF_Gauge *gauge,
                       const struct QOP_D3_MDWF_HalfFermion *rhs,
                       int max_iteration,
                       double epsilon,
                       unsigned int options);
```

Defines:
`QOP_D3_MDWF_MxM_CG`, used in chunk 32b.
`QOP_F3_MDWF_MxM_CG`, used in chunk 32a.
Uses `QOP_D3_MDWF_Gauge` 22a, `QOP_D3_MDWF_HalfFermion` 25b, `QOP_F3_MDWF_Gauge` 22a, `QOP_F3_MDWF_HalfFermion` 25b, and `QOP_MDWF_Parameters` 18d.

A collection of preconditioned equations with different positive shifts can be solved with very little extra cost using Algorithm 4.

31b ⟨*Interface functions* 18a⟩+≡

```
int QOP_F3_MDWF_MxM_SCG(struct QOP_F3_MDWF_VectorFermion *vector_result,
                        struct QOP_F3_MDWF_HalfFermion *scalar_result,
                        int *out_iterations,
                        double *out_epsilon,
                        const struct QOP_MDWF_Parameters *params,
                        const double shift[],
                        const struct QOP_F3_MDWF_Gauge *gauge,
                        const struct QOP_F3_MDWF_HalfFermion *rhs,
                        int max_iterations,
                        double min_epsilon,
                        unsigned int options);
int QOP_D3_MDWF_MxM_SCG(struct QOP_D3_MDWF_VectorFermion *vector_result,
                        struct QOP_D3_MDWF_HalfFermion *scalar_result,
                        int *out_iterations,
                        double *out_epsilon,
                        const struct QOP_MDWF_Parameters *params,
                        const double shift[],
                        const struct QOP_D3_MDWF_Gauge *gauge,
                        const struct QOP_D3_MDWF_HalfFermion *rhs,
                        int max_iterations,
                        double min_epsilon,
                        unsigned int options);
```

Defines:
`QOP_D3_MDWF_MxM_SCG`, used in chunk 32b.
`QOP_F3_MDWF_MxM_SCG`, used in chunk 32a.
Uses `QOP_D3_MDWF_Gauge` 22a, `QOP_D3_MDWF_HalfFermion` 25b, `QOP_F3_MDWF_Gauge` 22a, `QOP_F3_MDWF_HalfFermion` 25b, and `QOP_MDWF_Parameters` 18d.

Again, macros

32a ⟨*Single precision defaults* 22c⟩+≡

```
#define QOP_MDWF_DDW_CG QOP_F3_MDWF_DDW_CG
#define QOP_MDWF_MxM_CG QOP_F3_MDWF_MxM_CG
#define QOP_MDWF_MxM_SCG QOP_F3_MDWF_MxM_SCG
```

Defines:
QOP_MDWF_DDW_CG, never used.
QOP_MDWF_MxM_CG, never used.
QOP_MDWF_MxM_SCG, never used.
Uses QOP_F3_MDWF_DDW_CG 30b, QOP_F3_MDWF_MxM_CG 31a, and QOP_F3_MDWF_MxM_SCG 31b.

32b ⟨*Double precision defaults* 22d⟩+≡

```
#define QOP_MDWF_DDW_CG QOP_D3_MDWF_DDW_CG
#define QOP_MDWF_MxM_CG QOP_D3_MDWF_MxM_CG
#define QOP_MDWF_MxM_SCG QOP_D3_MDWF_MxM_SCG
```

Defines:
QOP_MDWF_DDW_CG, never used.
QOP_MDWF_MxM_CG, never used.
QOP_MDWF_MxM_SCG, never used.
Uses QOP_D3_MDWF_DDW_CG 30b, QOP_D3_MDWF_MxM_CG 31a, and QOP_D3_MDWF_MxM_SCG 31b.

3.14 Helper routines

To avoid excessive export and import calls, we provide the following linear algebra on full and preconditioned fermions.

$$r \leftarrow a + \alpha b$$

32c ⟨*Interface functions* 18a⟩+≡

```
int QOP_F3_MDWF_madd_fermion(struct QOP_F3_MDWF_Fermion *r,
                             const struct QOP_F3_MDWF_Fermion *a,
                             double alpha,
                             const struct QOP_F3_MDWF_Fermion *b);
int QOP_D3_MDWF_madd_fermion(struct QOP_D3_MDWF_Fermion *r,
                             const struct QOP_D3_MDWF_Fermion *a,
                             double alpha,
                             const struct QOP_D3_MDWF_Fermion *b);
int QOP_F3_MDWF_madd_half_fermion(struct QOP_F3_MDWF_HalfFermion *r,
                                  const struct QOP_F3_MDWF_HalfFermion *a,
                                  double alpha,
                                  const struct QOP_F3_MDWF_HalfFermion *b);
int QOP_D3_MDWF_madd_half_fermion(struct QOP_D3_MDWF_HalfFermion *r,
                                  const struct QOP_D3_MDWF_HalfFermion *a,
                                  double alpha,
                                  const struct QOP_D3_MDWF_HalfFermion *b);
```

Defines:
QOP_D3_MDWF_madd_fermion, used in chunk 34.
QOP_D3_MDWF_madd_half_fermion, used in chunk 34.
QOP_F3_MDWF_madd_fermion, used in chunk 33c.
QOP_F3_MDWF_madd_half_fermion, used in chunk 33c.
Uses QOP_D3_MDWF_Fermion 23b, QOP_D3_MDWF_HalfFermion 25b, QOP_F3_MDWF_Fermion 23b, and QOP_F3_MDWF_HalfFermion 25b.

$$\alpha \leftarrow \langle a, b \rangle$$

33a ⟨*Interface functions* 18a⟩+≡

```
int QOP_F3_MDWF_dot_fermion(double *r_re,
                            double *r_im,
                            const struct QOP_F3_MDWF_Fermion *a,
                            const struct QOP_F3_MDWF_Fermion *b);
int QOP_D3_MDWF_dot_fermion(double *r_re,
                            double *r_im,
                            const struct QOP_D3_MDWF_Fermion *a,
                            const struct QOP_D3_MDWF_Fermion *b);
int QOP_F3_MDWF_dot_half_fermion(double *r_re,
                                 double *r_im,
                                 const struct QOP_F3_MDWF_HalfFermion *a,
                                 const struct QOP_F3_MDWF_HalfFermion *b);
int QOP_D3_MDWF_dot_half_fermion(double *r_re,
                                 double *r_im,
                                 const struct QOP_D3_MDWF_HalfFermion *a,
                                 const struct QOP_D3_MDWF_HalfFermion *b);
```

Defines:
QOP_D3_MDWF_dot_fermion, used in chunk 34.
QOP_D3_MDWF_dot_half_fermion, used in chunk 34.
QOP_F3_MDWF_dot_fermion, used in chunk 33c.
QOP_F3_MDWF_dot_half_fermion, used in chunk 33c.
Uses QOP_D3_MDWF_Fermion 23b, QOP_D3_MDWF_HalfFermion 25b, QOP_F3_MDWF_Fermion 23b, and QOP_F3_MDWF_HalfFermion 25b.

$$\alpha \leftarrow \langle a, a \rangle$$

33b ⟨*Interface functions* 18a⟩+≡

```
int QOP_F3_MDWF_norm2_fermion(double *r,
                              const struct QOP_F3_MDWF_Fermion *a);
int QOP_D3_MDWF_norm2_fermion(double *r_re,
                              const struct QOP_D3_MDWF_Fermion *a);
int QOP_F3_MDWF_norm2_half_fermion(double *r_re,
                                   const struct QOP_F3_MDWF_HalfFermion *a);
int QOP_D3_MDWF_norm2_half_fermion(double *r_re,
                                   const struct QOP_D3_MDWF_HalfFermion *a);
```

Defines:
QOP_D3_MDWF_norm2_fermion, used in chunk 34.
QOP_D3_MDWF_norm2_half_fermion, used in chunk 34.
QOP_F3_MDWF_norm2_fermion, used in chunk 33c.
QOP_F3_MDWF_norm2_half_fermion, used in chunk 33c.
Uses QOP_D3_MDWF_Fermion 23b, QOP_D3_MDWF_HalfFermion 25b, QOP_F3_MDWF_Fermion 23b, and QOP_F3_MDWF_HalfFermion 25b.

Also, the macros

33c ⟨*Single precision defaults* 22c⟩+≡

```
#define QOP_MDWF_madd_fermion QOP_F3_MDWF_madd_fermion
#define QOP_MDWF_madd_half_fermion QOP_F3_MDWF_madd_half_fermion
#define QOP_MDWF_dot_fermion QOP_F3_MDWF_dot_fermion
#define QOP_MDWF_dot_half_fermion QOP_F3_MDWF_dot_half_fermion
#define QOP_MDWF_norm2_fermion QOP_F3_MDWF_norm2_fermion
#define QOP_MDWF_norm2_half_fermion QOP_F3_MDWF_norm2_half_fermion
```

Defines:
QOP_MDWF_dot_fermion, never used.
QOP_MDWF_dot_half_fermion, never used.
QOP_MDWF_madd_fermion, never used.
QOP_MDWF_madd_half_fermion, never used.
QOP_MDWF_norm2_fermion, never used.
QOP_MDWF_norm2_half_fermion, never used.
Uses QOP_F3_MDWF_dot_fermion 33a, QOP_F3_MDWF_dot_half_fermion 33a, QOP_F3_MDWF_madd_fermion 32c, QOP_F3_MDWF_madd_half_fermion 32c, QOP_F3_MDWF_norm2_fermion 33b, and QOP_F3_MDWF_norm2_half_fermion 33b.

34 ⟨*Double precision defaults* 22d⟩+≡

```
#define QOP_MDWF_madd_fermion QOP_D3_MDWF_madd_fermion
#define QOP_MDWF_madd_half_fermion QOP_D3_MDWF_madd_half_fermion
#define QOP_MDWF_dot_fermion QOP_D3_MDWF_dot_fermion
#define QOP_MDWF_dot_half_fermion QOP_D3_MDWF_dot_half_fermion
#define QOP_MDWF_norm2_fermion QOP_D3_MDWF_norm2_fermion
#define QOP_MDWF_norm2_half_fermion QOP_D3_MDWF_norm2_half_fermion
```

Defines:
QOP_MDWF_dot_fermion, never used.
QOP_MDWF_dot_half_fermion, never used.
QOP_MDWF_madd_fermion, never used.
QOP_MDWF_madd_half_fermion, never used.
QOP_MDWF_norm2_fermion, never used.
QOP_MDWF_norm2_half_fermion, never used.
Uses QOP_D3_MDWF_dot_fermion 33a, QOP_D3_MDWF_dot_half_fermion 33a, QOP_D3_MDWF_madd_fermion 32c, QOP_D3_MDWF_madd_half_fermion 32c, QOP_D3_MDWF_norm2_fermion 33b, and QOP_D3_MDWF_norm2_half_fermion 33b.

Chapter 4

CHANGES

May 17, 2008	avp	Bug in error handling fixed.
May 15, 2008	avp	Version 1.1.3. Bug in the docs fixed (master node definition).
May 14, 2008	avp	Version 1.1.2. Solvers set error in the state if they do not converge. Samples print a helpful marker.
May 8, 2008	avp	Version 1.1.1. A nasty bug in `samples` fixed.
May 7, 2008	avp	Version 1.1.0. Shifted solver added, samples installed.
May 5, 2008	avp	Version 1.0.1. Mixed precision bug fixed.
May 2, 2008	avp	Version 1.0.0.
October 9, 2007	avp	Initial version.

Appendix A

CODE CHUNKS

⟨*Double precision defaults* 22d⟩ 17a, 22d, 24d, 26d, 28b, 29d, 32b, 34
⟨*File* `../port/qop-mdwf3.h` 17a⟩ 17a
⟨*File* `../utils/basis.ss` 10⟩ 10
⟨*Interface functions* 18a⟩ 17a, 18a, 18b, 18e, 19a, 19b, 20a, 20b, 20c, 20d, 21a, 21b, 21c, 22b, 23a, 23c, 24a, 24b, 25a, 25c, 26a, 26b, 27b, 27c, 27d, 28c, 28d, 29a, 29b, 30b, 31a, 31b, 32c, 33a, 33b
⟨*Interface macros* 17b⟩ 17a, 17b, 17c, 17d
⟨*Interface types* 18c⟩ 17a, 18c, 18d, 22a, 23b, 25b, 27a, 30a
⟨*Project* $(1+\gamma_0)$ 6a⟩ 6a, 9f
⟨*Project* $(1+\gamma_1)$ 7a⟩ 7a, 9f
⟨*Project* $(1+\gamma_2)$ 8a⟩ 8a, 9f
⟨*Project* $(1+\gamma_3)$ 9a⟩ 9a, 9f
⟨*Project* $(1-\gamma_0)$ 6c⟩ 6c, 9f
⟨*Project* $(1-\gamma_1)$ 7c⟩ 7c, 9f
⟨*Project* $(1-\gamma_2)$ 8c⟩ 8c, 9f
⟨*Project* $(1-\gamma_3)$ 9d⟩ 9d, 9f
⟨*Scheme definitions* 9f⟩ 9f, 9g, 10
⟨*Single precision defaults* 22c⟩ 17a, 22c, 24c, 26c, 28a, 29c, 32a, 33c
⟨*Start* μ *sum* 9c⟩ 9c, 9g
⟨*Unproject* $(1+\gamma_0)$ 6b⟩ 6b, 9f
⟨*Unproject* $(1+\gamma_1)$ 7b⟩ 7b, 9f
⟨*Unproject* $(1+\gamma_2)$ 8b⟩ 8b, 9f
⟨*Unproject* $(1+\gamma_3)$ 9b⟩ 9b, 9f
⟨*Unproject* $(1-\gamma_0)$ 6d⟩ 6d, 9f
⟨*Unproject* $(1-\gamma_1)$ 7d⟩ 7d, 9f
⟨*Unproject* $(1-\gamma_2)$ 8d⟩ 8d, 9f
⟨*Unproject* $(1-\gamma_3)$ 9e⟩ 9e, 9f

Appendix B

SYMBOLS