Möbius Domain Wall Fermions Implementation

Andrew Pochinsky

X.XX.XX Termidor 18, XXXX

Contents

1 PHYSICS **5**
- 1.1 Gamma Matrices . . . 5
- 1.2 Preconditioning . . . 11
- 1.3 Inverting A and B . . . 12
- 1.4 Combinations of A and B . . . 14

2 ALGORITHMS **17**
- 2.1 Conjugate gradient . . . 17
- 2.2 Shifted Conjugate Gradient . . . 17

3 INTERFACE **21**
- 3.1 Magic numbers . . . 21
- 3.2 Library version . . . 22
- 3.3 Library signature . . . 22
- 3.4 Parity of even-odd preconditioning . . . 22
- 3.5 Performance monitoring . . . 22
- 3.6 Initialization . . . 23
- 3.7 Cleanup . . . 24
- 3.8 Errors . . . 24
- 3.9 OpenMP control . . . 24
- 3.10 Parameter setting . . . 25
- 3.11 Gauge . . . 27
- 3.12 Fermions . . . 29
- 3.13 Preconditioned fermions . . . 33
- 3.14 Fermion Vectors . . . 36
- 3.15 Dirac Operator . . . 38
- 3.16 Deflation . . . 40
- 3.17 Solvers . . . 43
- 3.18 Preconditioned operator functions . . . 48
- 3.19 Helper routines . . . 49
- 3.20 Debugging functions . . . 51

A CODE CHUNKS **53**

B SYMBOLS **55**

Chapter 1

PHYSICS

The Domain Wall Fermion Dirac operator is defined by

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

where

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

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

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

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

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

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 10a.

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+γ0) 6a⟩
  ⟨Project (1+γ1) 7a⟩
  ⟨Project (1+γ2) 8a⟩
  ⟨Project (1+γ3) 9a⟩
  ⟨Project (1−γ0) 6c⟩
  ⟨Project (1−γ1) 7c⟩
  ⟨Project (1−γ2) 8c⟩
  ⟨Project (1−γ3) 9d⟩
  ⟨Unproject (1+γ0) 6b⟩
  ⟨Unproject (1+γ1) 7b⟩
  ⟨Unproject (1+γ2) 8b⟩
  ⟨Unproject (1+γ3) 9b⟩
  ⟨Unproject (1−γ0) 6d⟩
  ⟨Unproject (1−γ1) 7d⟩
  ⟨Unproject (1−γ2) 8d⟩
  ⟨Unproject (1−γ3) 9e⟩))
```
This definition is continued in chunk 10a.
This code is used in chunk 10b.

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

10a ⟨*Scheme definitions* 9f⟩+≡
```
⟨Start μ sum 9c⟩
```
This code is used in chunk 10b.

Here is a module for PLT:

10b ⟨*File* `../utils/basis.ss` 10b⟩≡
```
(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

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

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

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

$$b_\alpha^{(+)}(k) = -\frac{u_\alpha(k)}{w_\alpha(k)}, \tag{1.38}$$

$$c_\alpha^{(+)}(k) = \frac{1}{w_\alpha(k)}. \tag{1.39}$$

Then algorithm 1 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
  $\eta \leftarrow \psi_0$
  $k \leftarrow 1$
  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_+.

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:

$$A^{-1} = A_{X+}^{-1}A_{L+}^{-1}P_+ + A_{X-}^{-1}A_{L-}^{-1}P_- \tag{1.48}$$

$$A^{-1} = A_{L+}^{-1}A_{Y+}^{-1}P_+ + A_{L-}^{-1}A_{Y-}^{-1}P_- \tag{1.49}$$

$$B^{-1} = B_{X+}^{-1}B_{L+}^{-1}P_+ + B_{X-}^{-1}B_{L-}^{-1}P_- \tag{1.50}$$

$$B^{-1} = B_{L+}^{-1}B_{Y+}^{-1}P_+ + B_{L-}^{-1}B_{Y-}^{-1}P_- \tag{1.51}$$

$$A_+ = A_{L+}A_{X+} \tag{1.52}$$

$$A_+ = A_{Y+}A_{L+} \tag{1.53}$$

$$A_- = A_{L-}A_{X-} \tag{1.54}$$

$$A_- = A_{Y-}A_{L-} \tag{1.55}$$

$$
\begin{aligned}
B_+ &= B_{L+}B_{X+} && (1.56)\\
B_+ &= B_{Y+}B_{L+} && (1.57)\\
B_- &= B_{L-}B_{X-} && (1.58)\\
B_- &= B_{Y-}B_{L-} && (1.59)\\
A^\dagger &= A^\dagger_+ P_+ + A^\dagger_- P_- && (1.60)\\
B^\dagger &= B^\dagger_+ P_+ + B^\dagger_- P_- && (1.61)\\
& && (1.62)
\end{aligned}
$$

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.

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: 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**
    $\omega \leftarrow M\pi$
    $\zeta \leftarrow M^\dagger \omega$
    $a \leftarrow r / \langle \omega, \omega \rangle$
    $\rho \leftarrow \rho - a\zeta$
    $g \leftarrow \langle \rho, \rho \rangle$
    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.

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** $w[i] \leftarrow v[i] \leftarrow 1$

  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** $w[i] \leftarrow 1/(1 + a * (s[i] + b_{-1} * (1 - w[i])/a_{-1}))$

    $\rho \leftarrow \rho - a * \zeta$
    $g \leftarrow \langle \rho, \rho \rangle$
    $b \leftarrow g/r$
    $r \leftarrow g$
    if $r < \epsilon$ **then**
      $\xi \leftarrow \xi + a * \pi$
      foreach i **do** $\xi[i] \leftarrow \xi[i] + (a * w[i] * v[i]) * \pi[i]$

      break
    end
    $\xi \leftarrow \xi + a * \pi$
    foreach i **do** $\xi[i] \leftarrow \xi[i] + (a * w[i] * v[i]) * \pi[i]$

    $\pi \leftarrow \rho + b * \pi$
    foreach i **do** $\pi[i] \leftarrow \rho + (b * w[i]) * \pi[i]$

    foreach i **do** $v[i] \leftarrow v[i] * w[i]$

    $b_{-1} \leftarrow b$
    $a_{-1} \leftarrow a$
    $k \leftarrow k + 1$
  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.

21a ⟨*File* ../port/qop-mdwf3.h 21a⟩≡

```
#ifndef QOP_MDWF_0bd50d0caeec4311a0d7c183032c43c2
# define QOP_MDWF_0bd50d0caeec4311a0d7c183032c43c2
  ⟨Interface macros 21b⟩
  ⟨Interface types 23a⟩
  ⟨Interface functions 22b⟩
# if defined(QOP_MDWF_DEFAULT_PRECISION) && (QOP_MDWF_DEFAULT_PRECISION == 'F')
   ⟨Single precision defaults 28b⟩
# endif
# if defined(QOP_MDWF_DEFAULT_PRECISION) && (QOP_MDWF_DEFAULT_PRECISION == 'D')
    ⟨Double precision defaults 28c⟩
# endif
#endif
```

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

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:

21b ⟨*Interface macros* 21b⟩≡

```
#define QOP_MDWF_DIM 4
```

This definition is continued in chunks 21c and 22a.
This code is used in chunk 21a.
Defines:
QOP_MDWF_DIM, never used.

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

21c ⟨*Interface macros* 21b⟩+≡

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

This code is used in chunk 21a.
Defines:
QOP_MDWF_FERMION_DIM, never used.
QOP_MDWF_PROJECTED_FERMION_DIM, never used.

We work only with $SU(3)$

22a ⟨*Interface macros* 21b⟩+≡
```
#define QOP_MDWF_COLORS 3
```
This code is used in chunk 21a.
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.

22b ⟨*Interface functions* 22b⟩≡
```
const char *QOP_MDWF_version(void);
```
This definition is continued in chunks 22–35, 37–46, and 48–51.
This code is used in chunk 21a.
Defines:
QOP_MDWF_version, never used.

3.3 Library signature

The following function returns a constant string which specifies the format of the deflator state. If the format of exported state deflator changes, a different string will be returned.

22c ⟨*Interface functions* 22b⟩+≡
```
const char *QOP_MDWF_signature(struct QOP_MDWF_State *state);
```
This code is used in chunk 21a.
Defines:
QOP_MDWF_signature, never used.
Uses QOP_MDWF_State 23a.

3.4 Parity of even-odd preconditioning

The parity of even-odd preconditioning used internally. The the returned value is zero, `QDP_even_L()` is used internally, otherwise the internals work on `QDP_odd_L()`.

22d ⟨*Interface functions* 22b⟩+≡
```
int QOP_MDWF_parity(struct QOP_MDWF_State *state);
```
This code is used in chunk 21a.
Defines:
QOP_MDWF_parity, never used.
Uses QOP_MDWF_State 23a.

3.5 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.

22e ⟨*Interface functions* 22b⟩+≡
```
int QOP_MDWF_performance(double *time_sec,
                         long long *flops,
                         long long *sent,
                         long long *receive,
                         struct QOP_MDWF_State *state);
```
This code is used in chunk 21a.
Defines:
QOP_MDWF_performance, never used.
Uses QOP_MDWF_State 23a.

3.6 Initialization

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

23a ⟨*Interface types* 23a⟩≡
```
struct QOP_MDWF_State;
```
This definition is continued in chunks 23, 27b, 29b, 34a, 36c, 40, and 43a.
This code is used in chunk 21a.
Defines:
QOP_MDWF_State, used in chunks 22–27, 29, 30b, 33–35, 37a, and 40.

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

23b ⟨*Interface types* 23a⟩+≡
```
struct QOP_MDWF_Parameters;
```
This code is used in chunk 21a.
Defines:
QOP_MDWF_Parameters, used in chunks 25, 26, 38–40, 42–46, 48a, and 51c.

Initialization needs to know lattice configuration and its layout on the machine. It expects to find the information in the following structure

23c ⟨*Interface types* 23a⟩+≡
```
struct QOP_MDWF_Config {
  int    self;                    /* this node QMP id */
  int    master_p;                /* if != 0, do I/O from this node */
  int    rank;                    /* lattice rank, must be 4 */
  int   *lat;                     /* [rank] */
  int    ls;                      /* lattice extend in the flavor dimension */
  int   *net;                     /* [rank] */
  int   *neighbor_up;             /* [rank], QMP ids of neighbors in up dirs */
  int   *neighbor_down;           /* [rank], QMP ids of neighbors in down dirs */
  void (*sublattice)(int lo[],    /*   [rank] */
                     int hi[],    /*   [rank] */
                     int node,    /*   any node QMP id */
                     void *env);  /*   lexical variables */
  void  *env;                     /* lexical variables for sublattice */
};
```
This code is used in chunk 21a.
Defines:
QOP_MDWF_Config, used in chunk 23d.

This structure is used by `QOP_MDWF_init()` only. Once the initialization is done, the elements of the structure (and data they point to) will never be referenced again.
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.

23d ⟨*Interface functions* 22b⟩+≡
```
int QOP_MDWF_init(struct QOP_MDWF_State **state_ptr,
                  struct QOP_MDWF_Config *config);
```
This code is used in chunk 21a.
Defines:
QOP_MDWF_init, never used.
Uses QOP_MDWF_Config 23c and QOP_MDWF_State 23a.

Arguments of `init()` are

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

`config` lattice layout information.

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 `*state_ptr`.
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.

3.7 Cleanup

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

24a ⟨*Interface functions* 22b⟩+≡
```
void QOP_MDWF_fini(struct QOP_MDWF_State **state_ptr);
```
This code is used in chunk 21a.
Defines:
QOP_MDWF_fini, never used.
Uses QOP_MDWF_State 23a.

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.

3.8 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:

24b ⟨*Interface functions* 22b⟩+≡
```
const char *QOP_MDWF_error(struct QOP_MDWF_State *state);
```
This code is used in chunk 21a.
Defines:
QOP_MDWF_error, never used.
Uses QOP_MDWF_State 23a.

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.

3.9 OpenMP control

The library may be condifured to use OpenMP for threading on a node. The granularity of threadable loops is controlled with

24c ⟨*Interface functions* 22b⟩+≡
```
int QOP_MDWF_set_threads(struct QOP_MDWF_State *state,
                         int threads);
```
This code is used in chunk 21a.
Defines:
QOP_MDWF_set_threads, never used.
Uses QOP_MDWF_State 23a.

If the value of `threads` is non-positive and the library was compiled with OpenMP support, then the current value of *nthreads-var* ICV is used. If the library was compiled without OpenMP, non-positive values of `threads` will be converted to `1`.

3.10 Parameter setting

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

25a ⟨*Interface functions* 22b⟩+≡

```
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);
int QOP_MDWF_set_complex(struct QOP_MDWF_Parameters **param_ptr,
                         struct QOP_MDWF_State *state,
                         const double b_5_re[],
                         const double b_5_im[],
                         const double c_5_re[],
                         const double c_5_im[],
                         double M_5,
                         double m);
```

This code is used in chunk 21a.
Defines:
`QOP_MDWF_set_complex`, never used.
`QOP_MDWF_set_generic`, never used.
Uses `QOP_MDWF_Parameters` 23b and `QOP_MDWF_State` 23a.

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

25b ⟨*Interface functions* 22b⟩+≡

```
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);
```

This code is used in chunk 21a.
Defines:
`QOP_MDWF_set_Moebius`, never used.
Uses `QOP_MDWF_Parameters` 23b and `QOP_MDWF_State` 23a.

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

25c ⟨*Interface functions* 22b⟩+≡

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

This code is used in chunk 21a.
Defines:
`QOP_MDWF_set_Shamir`, never used.
Uses `QOP_MDWF_Parameters` 23b and `QOP_MDWF_State` 23a.

For Boriçi, $b_5(s) = c_5(s) = a_5$:

26a ⟨*Interface functions* 22b⟩+≡

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

This code is used in chunk 21a.
Defines:
QOP_MDWF_set_Borichi, never used.
Uses QOP_MDWF_Parameters 23b and QOP_MDWF_State 23a.

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

26b ⟨*Interface functions* 22b⟩+≡

```
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);
```

This code is used in chunk 21a.
Defines:
QOP_MDWF_set_Chiu, never used.
Uses QOP_MDWF_Parameters 23b and QOP_MDWF_State 23a.

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

26c ⟨*Interface functions* 22b⟩+≡

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

This code is used in chunk 21a.
Defines:
QOP_MDWF_free_parameters, never used.
Uses QOP_MDWF_Parameters 23b.

3.11 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.

27a ⟨*Interface functions* 22b⟩+≡

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

This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_import_gauge, used in chunk 28c.
QOP_F3_MDWF_import_gauge, used in chunk 28b.
Uses QOP_D3_MDWF_Gauge 27b, QOP_F3_MDWF_Gauge 27b, and QOP_MDWF_State 23a.

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:

27b ⟨*Interface types* 23a⟩+≡

```
struct QOP_F3_MDWF_Gauge;
struct QOP_D3_MDWF_Gauge;
```

This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_Gauge, used in chunks 27, 28, 32a, 38, 39, 43–46, 48a, and 51c.
QOP_F3_MDWF_Gauge, used in chunks 27, 28, 32a, 38–40, 42–46, 48a, and 51c.

It is convenient to have a converter from double to single precision. It allocates space for single precision result.

27c ⟨*Interface functions* 22b⟩+≡

```
int QOP_MDWF_gauge_float_from_double(struct QOP_F3_MDWF_Gauge **result,
                                     struct QOP_D3_MDWF_Gauge *gauge_ptr);
```

This code is used in chunk 21a.
Defines:
QOP_MDWF_gauge_float_from_double, never used.
Uses QOP_D3_MDWF_Gauge 27b and QOP_F3_MDWF_Gauge 27b.

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

28a ⟨*Interface functions* 22b⟩+≡

```
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);
```

This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_free_gauge, used in chunk 28c.
QOP_F3_MDWF_free_gauge, used in chunk 28b.
Uses QOP_D3_MDWF_Gauge 27b and QOP_F3_MDWF_Gauge 27b.

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

28b ⟨*Single precision defaults* 28b⟩≡

```
#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
```

This definition is continued in chunks 32c, 36a, 37d, 39c, 47a, 48b, and 51a.
This code is used in chunk 21a.
Defines:
QOP_MDWF_free_gauge, never used.
QOP_MDWF_Gauge, never used.
QOP_MDWF_import_gauge, never used.
Uses QOP_F3_MDWF_free_gauge 28a, QOP_F3_MDWF_Gauge 27b, and QOP_F3_MDWF_import_gauge 27a.

28c ⟨*Double precision defaults* 28c⟩≡

```
#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
```

This definition is continued in chunks 33a, 36b, 38a, 39d, 47b, 48c, and 51b.
This code is used in chunk 21a.
Defines:
QOP_MDWF_free_gauge, never used.
QOP_MDWF_Gauge, never used.
QOP_MDWF_import_gauge, never used.
Uses QOP_D3_MDWF_free_gauge 28a, QOP_D3_MDWF_Gauge 27b, and QOP_D3_MDWF_import_gauge 27a.

3.12 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.

29a ⟨*Interface functions* 22b⟩+≡

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

This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_import_fermion, used in chunk 33a.
QOP_F3_MDWF_import_fermion, used in chunk 32c.
Uses QOP_D3_MDWF_Fermion 29b, QOP_F3_MDWF_Fermion 29b, and QOP_MDWF_State 23a.

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:

29b ⟨*Interface types* 23a⟩+≡

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

This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_Fermion, used in chunks 29–33, 38, 43, 44, 46b, and 49–51.
QOP_F3_MDWF_Fermion, used in chunks 29–32, 38, 43b, 44b, and 49–51.

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

29c ⟨*Interface functions* 22b⟩+≡

```
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);
```

This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_allocate_fermion, used in chunk 33a.
QOP_F3_MDWF_allocate_fermion, used in chunk 32c.
Uses QOP_D3_MDWF_Fermion 29b, QOP_F3_MDWF_Fermion 29b, and QOP_MDWF_State 23a.

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

30a ⟨*Interface functions* 22b⟩+≡

```
int QOP_F3_MDWF_export_fermion(void (*writer)(const int pos[5],
                                              int color,
                                              int dirac,
                                              double val_re,
                                              double val_im,
                                              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,
                                              double val_re,
                                              double val_im,
                                              void *env),
                               void *env,
                               const struct QOP_D3_MDWF_Fermion *fermion);
```

This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_export_fermion, used in chunk 33a.
QOP_F3_MDWF_export_fermion, used in chunk 32c.
Uses QOP_D3_MDWF_Fermion 29b and QOP_F3_MDWF_Fermion 29b.

We also provide convenience routines to import four-dimensional fermions

30b ⟨*Interface functions* 22b⟩+≡

```
int QOP_F3_MDWF_import_4d_fermion(struct QOP_F3_MDWF_Fermion **fermion_ptr,
                                  struct QOP_MDWF_State *state,
                                  void (*reader)(double *val_re,
                                                 double *val_im,
                                                 const int pos[4],
                                                 int color,
                                                 int dirac,
                                                 void *env),
                                  void *env);
int QOP_D3_MDWF_import_4d_fermion(struct QOP_D3_MDWF_Fermion **fermion_ptr,
                                  struct QOP_MDWF_State *state,
                                  void (*reader)(double *val_re,
                                                 double *val_im,
                                                 const int pos[4],
                                                 int color,
                                                 int dirac,
                                                 void *env),
                                  void *env);
```

This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_import_4d_fermion, used in chunk 33a.
QOP_F3_MDWF_import_4d_fermion, used in chunk 32c.
Uses QOP_D3_MDWF_Fermion 29b, QOP_F3_MDWF_Fermion 29b, and QOP_MDWF_State 23a.

Export routines are provided as well:

31a ⟨*Interface functions* 22b⟩+≡

```
int QOP_F3_MDWF_export_4d_fermion(void (*writer)(const int pos[4],
                                                 int color,
                                                 int dirac,
                                                 double val_re,
                                                 double val_im,
                                                 void *env),
                                  void *env,
                                  const struct QOP_F3_MDWF_Fermion *fermion);
int QOP_D3_MDWF_export_4d_fermion(void (*writer)(const int pos[4],
                                                 int color,
                                                 int dirac,
                                                 double val_re,
                                                 double val_im,
                                                 void *env),
                                  void *env,
                                  const struct QOP_D3_MDWF_Fermion *fermion);
```

This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_export_4d_fermion, used in chunk 33a.
QOP_F3_MDWF_export_4d_fermion, used in chunk 32c.
Uses QOP_D3_MDWF_Fermion 29b and QOP_F3_MDWF_Fermion 29b.

For residual mass calculations, one needs the following routines:

31b ⟨*Interface functions* 22b⟩+≡

```
int QOP_F3_MDWF_midpoint_pseudo(void (*writer)(const int pos[4],
                                               double value,
                                               void *env),
                                void *env,
                                const struct QOP_F3_MDWF_Fermion *fermion);
int QOP_D3_MDWF_midpoint_pseudo(void (*writer)(const int pos[4],
                                               double value,
                                               void *env),
                                void *env,
                                const struct QOP_D3_MDWF_Fermion *fermion);
```

This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_midpoint_pseudo, used in chunk 33a.
QOP_F3_MDWF_midpoint_pseudo, used in chunk 32c.
Uses QOP_D3_MDWF_Fermion 29b and QOP_F3_MDWF_Fermion 29b.

It is also convenient to have the conserved axial current export (it is real by construction):

32a ⟨*Interface functions* 22b⟩+≡

```
int QOP_F3_MDWF_axial_current(void (*writer)(const int pos[4],
                                             int dir,
                                             double value,
                                             void *env),
                              void *env,
                              const struct QOP_F3_MDWF_Fermion *fermion,
                              const struct QOP_F3_MDWF_Gauge *gauge);
int QOP_D3_MDWF_axial_current(void (*writer)(const int pos[4],
                                             int dir,
                                             double value,
                                             void *env),
                              void *env,
                              const struct QOP_D3_MDWF_Fermion *fermion,
                              const struct QOP_D3_MDWF_Gauge *gauge);
```

This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_axial_current, used in chunk 33a.
QOP_F3_MDWF_axial_current, used in chunk 32c.
Uses QOP_D3_MDWF_Fermion 29b, QOP_D3_MDWF_Gauge 27b, QOP_F3_MDWF_Fermion 29b, and QOP_F3_MDWF_Gauge 27b.

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

32b ⟨*Interface functions* 22b⟩+≡

```
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);
```

This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_free_fermion, used in chunk 33a.
QOP_F3_MDWF_free_fermion, used in chunk 32c.
Uses QOP_D3_MDWF_Fermion 29b and QOP_F3_MDWF_Fermion 29b.

Finally, macros for prefered precision

32c ⟨*Single precision defaults* 28b⟩+≡

```
#define QOP_MDWF_import_fermion QOP_F3_MDWF_import_fermion
#define QOP_MDWF_import_4d_fermion QOP_F3_MDWF_import_4d_fermion
#define QOP_MDWF_export_fermion QOP_F3_MDWF_export_fermion
#define QOP_MDWF_export_4d_fermion QOP_F3_MDWF_export_4d_fermion
#define QOP_MDWF_allocate_fermion QOP_F3_MDWF_allocate_fermion
#define QOP_MDWF_midpoint_pseudo QOP_F3_MDWF_midpoint_pseudo
#define QOP_MDWF_axial_current QOP_F3_MDWF_axial_current
#define QOP_MDWF_free_fermion QOP_F3_MDWF_free_fermion
#define QOP_MDWF_Fermion QOP_F3_MDWF_Fermion
```

This code is used in chunk 21a.
Defines:
QOP_MDWF_allocate_fermion, never used.
QOP_MDWF_axial_current, never used.
QOP_MDWF_export_4d_fermion, never used.
QOP_MDWF_export_fermion, never used.
QOP_MDWF_Fermion, never used.
QOP_MDWF_free_fermion, never used.
QOP_MDWF_import_4d_fermion, never used.
QOP_MDWF_import_fermion, never used.
QOP_MDWF_midpoint_pseudo, never used.
Uses QOP_F3_MDWF_allocate_fermion 29c, QOP_F3_MDWF_axial_current 32a, QOP_F3_MDWF_export_4d_fermion 31a, QOP_F3_MDWF_export_fermion 30a, QOP_F3_MDWF_Fermion 29b, QOP_F3_MDWF_free_fermion 32b, QOP_F3_MDWF_import_4d_fermion 30b, QOP_F3_MDWF_import_fermion 29a, and QOP_F3_MDWF_midpoint_pseudo 31b.

33a ⟨*Double precision defaults* 28c⟩+≡

```
#define QOP_MDWF_import_fermion QOP_D3_MDWF_import_fermion
#define QOP_MDWF_import_4d_fermion QOP_D3_MDWF_import_4d_fermion
#define QOP_MDWF_export_fermion QOP_D3_MDWF_export_fermion
#define QOP_MDWF_export_4d_fermion QOP_D3_MDWF_export_4d_fermion
#define QOP_MDWF_allocate_fermion QOP_D3_MDWF_allocate_fermion
#define QOP_MDWF_midpoint_pseudo QOP_D3_MDWF_midpoint_pseudo
#define QOP_MDWF_axial_current QOP_D3_MDWF_axial_current
#define QOP_MDWF_free_fermion QOP_D3_MDWF_free_fermion
#define QOP_MDWF_Fermion QOP_D3_MDWF_Fermion
```

This code is used in chunk 21a.
Defines:
QOP_MDWF_allocate_fermion, never used.
QOP_MDWF_axial_current, never used.
QOP_MDWF_export_4d_fermion, never used.
QOP_MDWF_export_fermion, never used.
QOP_MDWF_Fermion, never used.
QOP_MDWF_free_fermion, never used.
QOP_MDWF_import_4d_fermion, never used.
QOP_MDWF_import_fermion, never used.
QOP_MDWF_midpoint_pseudo, never used.
Uses QOP_D3_MDWF_allocate_fermion 29c, QOP_D3_MDWF_axial_current 32a, QOP_D3_MDWF_export_4d_fermion 31a, QOP_D3_MDWF_export_fermion 30a, QOP_D3_MDWF_Fermion 29b, QOP_D3_MDWF_free_fermion 32b, QOP_D3_MDWF_import_4d_fermion 30b, QOP_D3_MDWF_import_fermion 29a, and QOP_D3_MDWF_midpoint_pseudo 31b.

3.13 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.

33b ⟨*Interface functions* 22b⟩+≡

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

This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_import_half_fermion, used in chunk 36b.
QOP_F3_MDWF_import_half_fermion, used in chunk 36a.
Uses QOP_D3_MDWF_HalfFermion 34a, QOP_F3_MDWF_HalfFermion 34a, and QOP_MDWF_State 23a.

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:

34a ⟨*Interface types* 23a⟩+≡
```
struct QOP_F3_MDWF_HalfFermion;
struct QOP_D3_MDWF_HalfFermion;
```
This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_HalfFermion, used in chunks 33–37, 39, 45, 46a, and 48–50.
QOP_F3_MDWF_HalfFermion, used in chunks 33–37, 39, 42, 45, 46a, and 48–50.

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

34b ⟨*Interface functions* 22b⟩+≡
```
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);
```
This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_allocate_half_fermion, used in chunk 36b.
QOP_F3_MDWF_allocate_half_fermion, used in chunk 36a.
Uses QOP_D3_MDWF_HalfFermion 34a, QOP_F3_MDWF_HalfFermion 34a, and QOP_MDWF_State 23a.

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

34c ⟨*Interface functions* 22b⟩+≡
```
int QOP_F3_MDWF_export_half_fermion(void (*writer)(const int pos[5],
                                                   int color,
                                                   int dirac,
                                                   double val_re,
                                                   double val_im,
                                                   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,
                                                   double val_re,
                                                   double val_im,
                                                   void *env),
                                    void *env,
                                    const struct QOP_D3_MDWF_HalfFermion *hfermion);
```
This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_export_half_fermion, used in chunk 36b.
QOP_F3_MDWF_export_half_fermion, used in chunk 36a.
Uses QOP_D3_MDWF_HalfFermion 34a and QOP_F3_MDWF_HalfFermion 34a.

A fast path interface to BLAS allows one to convert between a half-fermion and an array of float (actually complex) numbers that BLAS and friends prefer. The `size` parameters must be at least as large as the value returned by `QDP_MDWF_half_fermion_size()` below. Export function write exactly `QDP_MDWF_half_fermion_size()` numbers to `data`; import functions read exactly `QDP_MDWF_half_fermion_size()` numbers from `data`.

35a ⟨*Interface functions* 22b⟩+≡

```
int QOP_F3_MDWF_blas_from_half_fermion(float *data,
                                       int size,
                                       const struct QOP_F3_MDWF_HalfFermion *hfermion);
int QOP_D3_MDWF_blas_from_half_fermion(double *data,
                                       int size,
                                       const struct QOP_D3_MDWF_HalfFermion *hfermion);
int QOP_F3_MDWF_half_fermion_from_blas(struct QOP_F3_MDWF_HalfFermion *hfermion,
                                       const float *data,
                                       int size);
int QOP_D3_MDWF_half_fermion_from_blas(struct QOP_D3_MDWF_HalfFermion *hfermion,
                                       const double *data,
                                       int size);
```

This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_blas_from_half_fermion, used in chunk 36b.
QOP_D3_MDWF_half_fermion_from_blas, used in chunk 36b.
QOP_F3_MDWF_blas_from_half_fermion, used in chunk 36a.
QOP_F3_MDWF_half_fermion_from_blas, used in chunk 36a.
Uses QOP_D3_MDWF_HalfFermion 34a and QOP_F3_MDWF_HalfFermion 34a.

We also provide a way to determine the number of floating point numbers needed to represent a half-fermion.

35b ⟨*Interface functions* 22b⟩+≡

```
int QOP_MDWF_half_fermion_size(struct QOP_MDWF_State *state_ptr);
```

This code is used in chunk 21a.
Defines:
QOP_MDWF_half_fermion_size, never used.
Uses QOP_MDWF_State 23a.

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

35c ⟨*Interface functions* 22b⟩+≡

```
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);
```

This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_free_half_fermion, used in chunk 36b.
QOP_F3_MDWF_free_half_fermion, used in chunk 36a.
Uses QOP_D3_MDWF_HalfFermion 34a and QOP_F3_MDWF_HalfFermion 34a.

Finally, macros for prefered precision

36a ⟨*Single precision defaults* 28b⟩+≡

```
#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_blas_from_half_fermion QOP_F3_MDWF_blas_from_half_fermion
#define QOP_MDWF_half_fermion_from_blas QOP_F3_MDWF_half_fermion_from_blas
#define QOP_MDWF_free_half_fermion QOP_F3_MDWF_free_half_fermion
#define QOP_MDWF_HalfFermion QOP_F3_MDWF_HalfFermion
```

This code is used in chunk 21a.
Defines:
QOP_MDWF_allocate_half_fermion, never used.
QOP_MDWF_blas_from_half_fermion, never used.
QOP_MDWF_export_half_fermion, never used.
QOP_MDWF_free_half_fermion, never used.
QOP_MDWF_half_fermion_from_blas, never used.
QOP_MDWF_HalfFermion, never used.
QOP_MDWF_import_half_fermion, never used.
Uses QOP_F3_MDWF_allocate_half_fermion 34b, QOP_F3_MDWF_blas_from_half_fermion 35a, QOP_F3_MDWF_export_half_fermion 34c, QOP_F3_MDWF_free_half_fermion 35c, QOP_F3_MDWF_half_fermion_from_blas 35a, QOP_F3_MDWF_HalfFermion 34a, and QOP_F3_MDWF_import_half_fermion 33b.

36b ⟨*Double precision defaults* 28c⟩+≡

```
#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_blas_from_half_fermion QOP_D3_MDWF_blas_from_half_fermion
#define QOP_MDWF_half_fermion_from_blas QOP_D3_MDWF_half_fermion_from_blas
#define QOP_MDWF_free_half_fermion QOP_D3_MDWF_free_half_fermion
#define QOP_MDWF_HalfFermion QOP_D3_MDWF_HalfFermion
```

This code is used in chunk 21a.
Defines:
QOP_MDWF_allocate_half_fermion, never used.
QOP_MDWF_blas_from_half_fermion, never used.
QOP_MDWF_export_half_fermion, never used.
QOP_MDWF_free_half_fermion, never used.
QOP_MDWF_half_fermion_from_blas, never used.
QOP_MDWF_HalfFermion, never used.
QOP_MDWF_import_half_fermion, never used.
Uses QOP_D3_MDWF_allocate_half_fermion 34b, QOP_D3_MDWF_blas_from_half_fermion 35a, QOP_D3_MDWF_export_half_fermion 34c, QOP_D3_MDWF_free_half_fermion 35c, QOP_D3_MDWF_half_fermion_from_blas 35a, QOP_D3_MDWF_HalfFermion 34a, and QOP_D3_MDWF_import_half_fermion 33b.

3.14 Fermion Vectors

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

36c ⟨*Interface types* 23a⟩+≡

```
struct QOP_F3_MDWF_VectorFermion;
struct QOP_D3_MDWF_VectorFermion;
```

This code is used in chunk 21a.

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

37a ⟨*Interface functions* 22b⟩+≡

```
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);
```

This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_allocate_vector_fermion, used in chunk 38a.
QOP_F3_MDWF_allocate_vector_fermion, used in chunk 37d.
Uses QOP_MDWF_State 23a.

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

37b ⟨*Interface functions* 22b⟩+≡

```
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);
```

This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_free_vector_fermion, used in chunk 38a.
QOP_F3_MDWF_free_vector_fermion, used in chunk 37d.

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

37c ⟨*Interface functions* 22b⟩+≡

```
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);
```

This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_get_vector_fermion, used in chunk 38a.
QOP_D3_MDWF_put_vector_fermion, used in chunk 38a.
QOP_F3_MDWF_get_vector_fermion, used in chunk 37d.
QOP_F3_MDWF_put_vector_fermion, used in chunk 37d.
Uses QOP_D3_MDWF_HalfFermion 34a and QOP_F3_MDWF_HalfFermion 34a.

Macros for default precision:

37d ⟨*Single precision defaults* 28b⟩+≡

```
#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
```

This code is used in chunk 21a.
Defines:
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.
QOP_MDWF_VectorFermion, never used.
Uses QOP_F3_MDWF_allocate_vector_fermion 37a, QOP_F3_MDWF_free_vector_fermion 37b, QOP_F3_MDWF_get_vector_fermion 37c, and QOP_F3_MDWF_put_vector_fermion 37c.

38a ⟨*Double precision defaults* 28c⟩+≡

```
#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
```

This code is used in chunk 21a.
Defines:
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.
QOP_MDWF_VectorFermion, never used.
Uses QOP_D3_MDWF_allocate_vector_fermion 37a, QOP_D3_MDWF_free_vector_fermion 37b, QOP_D3_MDWF_get_vector_fermion 37c, and QOP_D3_MDWF_put_vector_fermion 37c.

3.15 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.

38b ⟨*Interface functions* 22b⟩+≡

```
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);
```

This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_DDW_operator, used in chunk 39d.
QOP_F3_MDWF_DDW_operator, used in chunk 39c.
Uses QOP_D3_MDWF_Fermion 29b, QOP_D3_MDWF_Gauge 27b, QOP_F3_MDWF_Fermion 29b, QOP_F3_MDWF_Gauge 27b, and QOP_MDWF_Parameters 23b.

38c ⟨*Interface functions* 22b⟩+≡

```
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);
```

This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_DDW_operator_conjugated, used in chunk 39d.
QOP_F3_MDWF_DDW_operator_conjugated, used in chunk 39c.
Uses QOP_D3_MDWF_Fermion 29b, QOP_D3_MDWF_Gauge 27b, QOP_F3_MDWF_Fermion 29b, QOP_F3_MDWF_Gauge 27b, and QOP_MDWF_Parameters 23b.

39a ⟨*Interface functions* 22b⟩+≡

```
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);
```

This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_M_operator, used in chunk 39d.
QOP_F3_MDWF_M_operator, used in chunk 39c.
Uses QOP_D3_MDWF_Gauge 27b, QOP_D3_MDWF_HalfFermion 34a, QOP_F3_MDWF_Gauge 27b, QOP_F3_MDWF_HalfFermion 34a, and QOP_MDWF_Parameters 23b.

39b ⟨*Interface functions* 22b⟩+≡

```
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);
```

This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_M_operator_conjugated, used in chunk 39d.
QOP_F3_MDWF_M_operator_conjugated, used in chunk 39c.
Uses QOP_D3_MDWF_Gauge 27b, QOP_D3_MDWF_HalfFermion 34a, QOP_F3_MDWF_Gauge 27b, QOP_F3_MDWF_HalfFermion 34a, and QOP_MDWF_Parameters 23b.

Also the default precision macros

39c ⟨*Single precision defaults* 28b⟩+≡

```
#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
```

This code is used in chunk 21a.
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 38b, QOP_F3_MDWF_DDW_operator_conjugated 38c, QOP_F3_MDWF_M_operator 39a, and QOP_F3_MDWF_M_operator_conjugated 39b.

39d ⟨*Double precision defaults* 28c⟩+≡

```
#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
```

This code is used in chunk 21a.
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 38b, QOP_D3_MDWF_DDW_operator_conjugated 38c, QOP_D3_MDWF_M_operator 39a, and QOP_D3_MDWF_M_operator_conjugated 39b.

3.16 Deflation

For delfation solvers, the following interface is provided. Deflators are used by the deflation solvers to keep low eigenvalue vector space. Low eigenmode space can be either accumulated while solving Dirac equations (EigCG) or loaded into a deflator. There are two way to load vectors: one by one or as a blas matrix. In the latter case, a BLAS matrix must be pre-allocated as `(...)_HalfFermionMat` and filled following the site/spin/color conventions compatible with `(...)_blas_from_half_fermion`. In practice, such a matrix is allocated and used in Lanczos iterations, and later supplied to create a deflator object; in-place conversion helps to avoid copying vectors and allocating the associated memory twice. Vectors may be exported from the deflator one by one. Currently, only single precision deflator is built by default.
The state of the deflator is packaged into an opaque structure:

40a ⟨*Interface types* 23a⟩+≡
```
struct QOP_F3_MDWF_Deflator;
```
This code is used in chunk 21a.
Defines:
QOP_F3_MDWF_Deflator, used in chunks 40–42 and 46b.

The deflators are created with given capacity, size and precision:

40b ⟨*Interface functions* 22b⟩+≡
```
int QOP_F3_MDWF_create_deflator(struct QOP_F3_MDWF_Deflator **defl_ptr,
                                struct QOP_MDWF_State       *state,
                                int nev, int vsize, double eps, int umax);
```
This code is used in chunk 21a.
Defines:
QOP_F3_MDWF_create_deflator, never used.
Uses QOP_F3_MDWF_Deflator 40a and QOP_MDWF_State 23a.

Alternatively, one can use a user-filled BLAS matrix (sequence of BLAS vectors), from which the first `hfm_nev` vectors are used for deflation, and the remaining space in the allocated matrix may be filled with additional vectors from EigCG (but not to exceed `eigcg_umax`). Workspace for EigCG is allocated if and only if `eigcg_vmax`, `eigcg_nev` > 0:

40c ⟨*Interface functions* 22b⟩+≡
```
int QOP_F3_MDWF_create_deflator_inplace(struct QOP_F3_MDWF_Deflator       **defl_ptr,
                                        const struct QOP_MDWF_Parameters  *params,
                                        const struct QOP_F3_MDWF_Gauge    *gauge,
                                        struct QOP_F3_MDWF_HalfFermionMat **hfm_ptr,
                                        int                               hfm_nev,
                                        int                               eigcg_vmax,
                                        int                               eigcg_nev,
                                        double                            eigcg_eps,
                                        int                               eigcg_umax);
```
This code is used in chunk 21a.
Defines:
QOP_F3_MDWF_create_deflator_inplace, never used.
Uses QOP_F3_MDWF_Deflator 40a, QOP_F3_MDWF_Gauge 27b, QOP_F3_MDWF_HalfFermionMat 40d, and QOP_MDWF_Parameters 23b.

The BLAS-like space handler `hfm_ptr` has an opaque type

40d ⟨*Interface types* 23a⟩+≡
```
struct QOP_F3_MDWF_HalfFermionMat;
```
This code is used in chunk 21a.
Defines:
QOP_F3_MDWF_HalfFermionMat, used in chunks 40 and 41.

and must be preallocated by the function

40e ⟨*Interface functions* 22b⟩+≡
```
int QOP_F3_MDWF_alloc_half_fermion_matrix(struct QOP_F3_MDWF_HalfFermionMat **hfm_ptr,
                                          struct QOP_MDWF_State *state,
                                          int ncol);
```
This code is used in chunk 21a.
Defines:
QOP_F3_MDWF_alloc_half_fermion_matrix, never used.
Uses QOP_F3_MDWF_HalfFermionMat 40d and QOP_MDWF_State 23a.

A `HalfFermionMat` object is essentially a wrapper for a BLAS-style matrix; to use it as such, the following function provides a raw pointer to the data, together with the number of rows and columns and the "leading dimension" of the array:

41a ⟨*Interface functions* 22b⟩+≡

```
int QOP_F3_MDWF_blas_view_half_fermion_matrix(struct QOP_F3_MDWF_HalfFermionMat *hfm_ptr,
                                              int        *nrow_loc,
                                              int        *ncol,
                                              float      **blas_ptr,
                                              int        *blas_ld);
```

This code is used in chunk 21a.
Defines:
`QOP_F3_MDWF_blas_view_half_fermion_matrix`, never used.
Uses `QOP_F3_MDWF_HalfFermionMat` 40d.

The element order in the matrix conforms to `FORTRAN` conventions, that is

$$\begin{aligned} \mathrm{Re}M_{ij} &= (*\texttt{blas\_ptr})[2*i+2*j*\texttt{blas\_ld}]\,, \\ \mathrm{Im}M_{ij} &= (*\texttt{blas\_ptr})[2*i+2*j*\texttt{blas\_ld}+1]\,, \\ & \quad 0 \le i < \texttt{nrow\_loc}, \quad 0 \le j < \texttt{ncol}\,, \end{aligned}$$

and `nrow_loc` is the local vector size (complex numbers per process). The j-th column represents the j-th vector. Once a `HalfFermionMat` object is used to create a deflator, it cannot be reused since its allocated memory belongs to the new deflator. For this reason, `create_deflator_inplace` actually deallocates the associated storage structures and sets `*hfm_ptr` to `NULL`. The following function may be called to manually deallocate a `HalfFermionMat` object (`NULL` pointer-safe):

41b ⟨*Interface functions* 22b⟩+≡

```
void QOP_F3_MDWF_free_half_fermion_matrix(struct QOP_F3_MDWF_HalfFermionMat **hfm_ptr);
```

This code is used in chunk 21a.
Uses `QOP_F3_MDWF_HalfFermionMat` 40d.

Once done, the deflators must be freed with the following functions:

41c ⟨*Interface functions* 22b⟩+≡

```
void QOP_F3_MDWF_free_deflator(struct QOP_F3_MDWF_Deflator **defl_ptr);
```

This code is used in chunk 21a.
Defines:
`QOP_F3_MDWF_free_deflator`, never used.
Uses `QOP_F3_MDWF_Deflator` 40a.

Functions to control the state of the deflator. If one needs to keep the handle but purge the stored or EigCG-accumulated data, there is a `eigcg_reset` operation:

41d ⟨*Interface functions* 22b⟩+≡

```
int QOP_F3_MDWF_deflator_eigcg_reset(struct QOP_F3_MDWF_Deflator *defl_ptr);
```

This code is used in chunk 21a.
Defines:
`QOP_F3_MDWF_deflator_eigcg_reset`, never used.
Uses `QOP_F3_MDWF_Deflator` 40a.

To control collection of the Krylov space, use `eigcg_stop` and `eigcg_resume`. While the EigCG is stopped, it can still be used in the deflated solver and queried about accumulated eigenvalues.

41e ⟨*Interface functions* 22b⟩+≡

```
int QOP_F3_MDWF_deflator_eigcg_stop(struct QOP_F3_MDWF_Deflator *defl_ptr);
int QOP_F3_MDWF_deflator_eigcg_resume(struct QOP_F3_MDWF_Deflator *defl_ptr);
```

This code is used in chunk 21a.
Defines:
`QOP_F3_MDWF_deflator_eigcg_resume`, never used.
`QOP_F3_MDWF_deflator_eigcg_stop`, never used.
Uses `QOP_F3_MDWF_Deflator` 40a.

The following fuction may be used to inquire the state of EigCG:

42a ⟨*Interface functions* 22b⟩+≡
```
int QOP_F3_MDWF_deflator_eigcg_is_stopped(struct QOP_F3_MDWF_Deflator *defl_ptr);
```
This code is used in chunk 21a.
Defines:
QOP_F3_MDWF_deflator_eigcg_is_stopped, never used.
Uses QOP_F3_MDWF_Deflator 40a.

The current dimension of low space may be requested from the deflator:

42b ⟨*Interface functions* 22b⟩+≡
```
int QOP_F3_MDWF_deflator_current_dim(struct QOP_F3_MDWF_Deflator *defl_ptr);
```
This code is used in chunk 21a.
Defines:
QOP_F3_MDWF_deflator_current_dim, never used.
Uses QOP_F3_MDWF_Deflator 40a.

Extract a vector from the deflator. If the vector index is out of range, nothing is written into memory and a non-zero value is returned.

42c ⟨*Interface functions* 22b⟩+≡
```
int QOP_F3_MDWF_deflator_extract_vector(struct QOP_F3_MDWF_HalfFermion *hfermion_ptr,
                                        const struct QOP_F3_MDWF_Deflator *defl_ptr,
                                        int idx);
```
This code is used in chunk 21a.
Defines:
QOP_F3_MDWF_deflator_extract_vector, never used.
Uses QOP_F3_MDWF_Deflator 40a and QOP_F3_MDWF_HalfFermion 34a.

Before adding vectors to the deflator, it must be marked to prevent accidental changes to the deflator state

42d ⟨*Interface functions* 22b⟩+≡
```
int QOP_F3_MDWF_deflator_start_load(struct QOP_F3_MDWF_Deflator *defl_ptr);
```
This code is used in chunk 21a.
Defines:
QOP_F3_MDWF_deflator_start_load, never used.
Uses QOP_F3_MDWF_Deflator 40a.

Manually add a vector to the deflator. Zero value is returned if the insersion was a success.

42e ⟨*Interface functions* 22b⟩+≡
```
int QOP_F3_MDWF_deflator_add_vector(const struct QOP_MDWF_Parameters *params,
                                    const struct QOP_F3_MDWF_Gauge *gauge,
                                    struct QOP_F3_MDWF_Deflator *deflator,
                                    const struct QOP_F3_MDWF_HalfFermion *hfermion);
```
This code is used in chunk 21a.
Defines:
QOP_F3_MDWF_deflator_add_vector, never used.
Uses QOP_F3_MDWF_Deflator 40a, QOP_F3_MDWF_Gauge 27b, QOP_F3_MDWF_HalfFermion 34a, and QOP_MDWF_Parameters 23b.

After a set of vector is loaded, the deflator should be released before using it in an inverter

42f ⟨*Interface functions* 22b⟩+≡
```
int QOP_F3_MDWF_deflator_stop_load(struct QOP_F3_MDWF_Deflator *defl_ptr);
```
This code is used in chunk 21a.
Defines:
QOP_F3_MDWF_deflator_stop_load, never used.
Uses QOP_F3_MDWF_Deflator 40a.

The deflator can return its current estimate of the eigenvalues if it has it. There are either `nev` eigenmode estimates, or none. If the deflator does not have the estimates, `eigen` returns `1`, otherwise it returns `0`.

42g ⟨*Interface functions* 22b⟩+≡
```
int QOP_F3_MDWF_deflator_eigen(int n, double *eigen_values,
                               struct QOP_F3_MDWF_Deflator *defl_ptr);
```
This code is used in chunk 21a.
Defines:
QOP_F3_MDWF_deflator_eigen, never used.
Uses QOP_F3_MDWF_Deflator 40a.

3.17 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.

43a ⟨*Interface types* 23a⟩+≡

```
enum {
   QOP_MDWF_LOG_NONE              = 0x00,
   QOP_MDWF_LOG_CG_RESIDUAL       = 0x01,
   QOP_MDWF_LOG_TRUE_RESIDUAL     = 0x02,
   QOP_MDWF_FINAL_CG_RESIDUAL     = 0x04,
   QOP_MDWF_FINAL_DIRAC_RESIDUAL  = 0x08,
   QOP_MDWF_LOG_EIG_UPDATE1       = 0x10,
   QOP_MDWF_LOG_EIG_POSTAMBLE     = 0x20,
   QOP_MDWF_LOG_EVERYTHING        = 0x3f
};
```

This code is used in chunk 21a.
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_EIG_POSTAMBLE, never used.
QOP_MDWF_LOG_EIG_UPDATE1, never used.
QOP_MDWF_LOG_EVERYTHING, never used.
QOP_MDWF_LOG_NONE, never used.
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.

43b ⟨*Interface functions* 22b⟩+≡

```
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);
```

This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_DDW_CG, used in chunk 47b.
QOP_F3_MDWF_DDW_CG, used in chunk 47a.
Uses QOP_D3_MDWF_Fermion 29b, QOP_D3_MDWF_Gauge 27b, QOP_F3_MDWF_Fermion 29b, QOP_F3_MDWF_Gauge 27b, and QOP_MDWF_Parameters 23b.

The mixed precision solver uses the trick communicated by Stefan Krieg: run the CG in single precision while accumulating the result in double. It has only a double precision version and takes an extra parameter `f_iterations` which controls the number of float CG iterations between double updates.

44a ⟨*Interface functions* 22b⟩+≡

```
int QOP_MDWF_mixed_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 f_iterations,
                          double f_epsilon,
                          int max_iteration,
                          double epsilon,
                          unsigned int options);
```

This code is used in chunk 21a.
Defines:
QOP_MDWF_mixed_DDW_CG, never used.
Uses QOP_D3_MDWF_Fermion 29b, QOP_D3_MDWF_Gauge 27b, and QOP_MDWF_Parameters 23b.

A solver for $D^\dagger D\psi = \eta$ is also provided:

44b ⟨*Interface functions* 22b⟩+≡

```
int QOP_F3_MDWF_DxD_CG(struct QOP_F3_MDWF_Fermion *psi,
                       int *out_iterations,
                       double *out_epsilon,
                       const struct QOP_MDWF_Parameters *params,
                       const struct QOP_F3_MDWF_Fermion *psi_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_DxD_CG(struct QOP_D3_MDWF_Fermion *psi,
                       int *out_iterations,
                       double *out_epsilon,
                       const struct QOP_MDWF_Parameters *params,
                       const struct QOP_D3_MDWF_Fermion *psi_0,
                       const struct QOP_D3_MDWF_Gauge *gauge,
                       const struct QOP_D3_MDWF_Fermion *rhs,
                       int max_iteration,
                       double epsilon,
                       unsigned int options);
```

This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_DxD_CG, used in chunk 47b.
QOP_F3_MDWF_DxD_CG, used in chunk 47a.
Uses QOP_D3_MDWF_Fermion 29b, QOP_D3_MDWF_Gauge 27b, QOP_F3_MDWF_Fermion 29b, QOP_F3_MDWF_Gauge 27b, and QOP_MDWF_Parameters 23b.

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.

45 ⟨*Interface functions* 22b⟩+≡

```
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);
```

This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_MxM_CG, used in chunk 47b.
QOP_F3_MDWF_MxM_CG, used in chunk 47a.
Uses QOP_D3_MDWF_Gauge 27b, QOP_D3_MDWF_HalfFermion 34a, QOP_F3_MDWF_Gauge 27b, QOP_F3_MDWF_HalfFermion 34a, and QOP_MDWF_Parameters 23b.

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

46a ⟨*Interface functions* 22b⟩+≡

```
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);
```

This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_MxM_SCG, used in chunk 47b.
QOP_F3_MDWF_MxM_SCG, used in chunk 47a.
Uses QOP_D3_MDWF_Gauge 27b, QOP_D3_MDWF_HalfFermion 34a, QOP_F3_MDWF_Gauge 27b, QOP_F3_MDWF_HalfFermion 34a, and QOP_MDWF_Parameters 23b.

A mixed precision deflated solver is provided. The interface is basically the same as the plain mixed solver above with addition of an extra parameter for the deflator state.

46b ⟨*Interface functions* 22b⟩+≡

```
int QOP_MDWF_deflated_mixed_D_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 *chi_0,
                                 const struct QOP_D3_MDWF_Gauge *gauge,
                                 const struct QOP_D3_MDWF_Fermion *rhs,
                                 struct QOP_F3_MDWF_Deflator *deflator,
                                 int f_iterations,
                                 double f_epsilon,
                                 int max_iteration,
                                 double epsilon,
                                 unsigned int options);
```

This code is used in chunk 21a.
Defines:
QOP_MDWF_deflated_mixed_D_CG, never used.
Uses QOP_D3_MDWF_Fermion 29b, QOP_D3_MDWF_Gauge 27b, QOP_F3_MDWF_Deflator 40a, and QOP_MDWF_Parameters 23b.

Again, macros

47a ⟨*Single precision defaults* 28b⟩+≡

```
#define QOP_MDWF_DDW_CG QOP_F3_MDWF_DDW_CG
#define QDP_MDWF_DxD_CG QOP_F3_MDWF_DxD_CG
#define QOP_MDWF_MxM_CG QOP_F3_MDWF_MxM_CG
#define QOP_MDWF_MxM_SCG QOP_F3_MDWF_MxM_SCG
```

This code is used in chunk 21a.
Defines:
QDP_MDWF_DxD_CG, never used.
QOP_MDWF_DDW_CG, never used.
QOP_MDWF_MxM_CG, never used.
QOP_MDWF_MxM_SCG, never used.
Uses QOP_F3_MDWF_DDW_CG 43b, QOP_F3_MDWF_DxD_CG 44b, QOP_F3_MDWF_MxM_CG 45, and QOP_F3_MDWF_MxM_SCG 46a.

47b ⟨*Double precision defaults* 28c⟩+≡

```
#define QOP_MDWF_DDW_CG QOP_D3_MDWF_DDW_CG
#define QDP_MDWF_DxD_CG QOP_D3_MDWF_DxD_CG
#define QOP_MDWF_MxM_CG QOP_D3_MDWF_MxM_CG
#define QOP_MDWF_MxM_SCG QOP_D3_MDWF_MxM_SCG
```

This code is used in chunk 21a.
Defines:
QDP_MDWF_DxD_CG, never used.
QOP_MDWF_DDW_CG, never used.
QOP_MDWF_MxM_CG, never used.
QOP_MDWF_MxM_SCG, never used.
Uses QOP_D3_MDWF_DDW_CG 43b, QOP_D3_MDWF_DxD_CG 44b, QOP_D3_MDWF_MxM_CG 45, and QOP_D3_MDWF_MxM_SCG 46a.

3.18 Preconditioned operator functions

Apply an $n \geq 1$ degree polynomial of $M^\dagger M$ to vector ψ. The polynomial is defined using the general three-term recurrence relation

$$\begin{aligned} P_0(M^\dagger M)\psi &= c_0\psi \\ P_1(M^\dagger M)\psi &= a_0\psi + b_0 M^\dagger M\psi \\ P_n(M^\dagger M)\psi &= (a_{n-1} + b_{n-1}M^\dagger M)P_{n-1}(M^\dagger M)\psi + c_{n-1}P_{n-2}(M^\dagger M)\psi, \quad n > 1 \end{aligned}$$

The value of $P_n(M^\dagger M)\psi$ is returned in `result`, and if `result_prev` is not `NULL`, it shall contain the value of $P_{n-1}(M^\dagger M)\psi$ on return. Functions below compute operator polynomials for positive `n`. Arrays `a`, `b`, and `c` contain recursion coefficients and must be at least of length `n`.

48a ⟨*Interface functions* 22b⟩+≡

```
int QOP_F3_MDWF_MxM_poly(struct QOP_F3_MDWF_HalfFermion *result,
                         struct QOP_F3_MDWF_HalfFermion *result_prev,
                         const struct QOP_MDWF_Parameters *params,
                         const struct QOP_F3_MDWF_Gauge *gauge,
                         const struct QOP_F3_MDWF_HalfFermion *psi,
                         int n,
                         const double a[/* n */],
                         const double b[/* n */],
                         const double c[/* n */]);
int QOP_D3_MDWF_MxM_poly(struct QOP_D3_MDWF_HalfFermion *result,
                         struct QOP_D3_MDWF_HalfFermion *result_prev,
                         const struct QOP_MDWF_Parameters *params,
                         const struct QOP_D3_MDWF_Gauge *gauge,
                         const struct QOP_D3_MDWF_HalfFermion *psi,
                         int n,
                         const double a[/* n */],
                         const double b[/* n */],
                         const double c[/* n */]);
```

This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_MxM_poly, used in chunk 48c.
QOP_F3_MDWF_MxM_poly, used in chunk 48b.
Uses QOP_D3_MDWF_Gauge 27b, QOP_D3_MDWF_HalfFermion 34a, QOP_F3_MDWF_Gauge 27b, QOP_F3_MDWF_HalfFermion 34a, and QOP_MDWF_Parameters 23b.

The macros for default precision:

48b ⟨*Single precision defaults* 28b⟩+≡

```
#define QOP_MDWF_MxM_poly QOP_F3_MDWF_MxM_poly
```

This code is used in chunk 21a.
Defines:
QOP_MDWF_MxM_poly, never used.
Uses QOP_F3_MDWF_MxM_poly 48a.

48c ⟨*Double precision defaults* 28c⟩+≡

```
#define QOP_MDWF_MxM_poly QOP_D3_MDWF_MxM_poly
```

This code is used in chunk 21a.
Defines:
QOP_MDWF_MxM_poly, never used.
Uses QOP_D3_MDWF_MxM_poly 48a.

Since polynomials may have values that can cause overflow of floating point operations, it may be necessary to rescale these polynomials to some finite value. The following function rescales the coefficients `a`, `b`, `c` so that all the polynomials are equal to 1 at some point x_0, unless they have absolute values smaller than `tol`:

49a ⟨*Interface functions* 22b⟩+≡

```
int QOP_MDWF_poly_normalize(int n,
                            double a[/* n */],
                            double b[/* n */],
                            double c[/* n */],
                            double x0,
                            double tol);
```

This code is used in chunk 21a.
Defines:
`QOP_MDWF_poly_normalize`, never used.

3.19 Helper routines

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

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

49b ⟨*Interface functions* 22b⟩+≡

```
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);
```

This code is used in chunk 21a.
Defines:
`QOP_D3_MDWF_madd_fermion`, used in chunk 51b.
`QOP_D3_MDWF_madd_half_fermion`, used in chunk 51b.
`QOP_F3_MDWF_madd_fermion`, used in chunk 51a.
`QOP_F3_MDWF_madd_half_fermion`, used in chunk 51a.
Uses `QOP_D3_MDWF_Fermion` 29b, `QOP_D3_MDWF_HalfFermion` 34a, `QOP_F3_MDWF_Fermion` 29b, and `QOP_F3_MDWF_HalfFermion` 34a.

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

50a ⟨*Interface functions* 22b⟩+≡

```
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);
```

This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_dot_fermion, used in chunk 51b.
QOP_D3_MDWF_dot_half_fermion, used in chunk 51b.
QOP_F3_MDWF_dot_fermion, used in chunk 51a.
QOP_F3_MDWF_dot_half_fermion, used in chunk 51a.
Uses QOP_D3_MDWF_Fermion 29b, QOP_D3_MDWF_HalfFermion 34a, QOP_F3_MDWF_Fermion 29b, and QOP_F3_MDWF_HalfFermion 34a.

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

50b ⟨*Interface functions* 22b⟩+≡

```
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);
```

This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_norm2_fermion, used in chunk 51b.
QOP_D3_MDWF_norm2_half_fermion, used in chunk 51b.
QOP_F3_MDWF_norm2_fermion, used in chunk 51a.
QOP_F3_MDWF_norm2_half_fermion, used in chunk 51a.
Uses QOP_D3_MDWF_Fermion 29b, QOP_D3_MDWF_HalfFermion 34a, QOP_F3_MDWF_Fermion 29b, and QOP_F3_MDWF_HalfFermion 34a.

Also, the macros

51a ⟨*Single precision defaults* 28b⟩+≡

```
#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
```

This code is used in chunk 21a.
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 50a, QOP_F3_MDWF_dot_half_fermion 50a, QOP_F3_MDWF_madd_fermion 49b, QOP_F3_MDWF_madd_half_fermion 49b, QOP_F3_MDWF_norm2_fermion 50b, and QOP_F3_MDWF_norm2_half_fermion 50b.

51b ⟨*Double precision defaults* 28c⟩+≡

```
#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
```

This code is used in chunk 21a.
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 50a, QOP_D3_MDWF_dot_half_fermion 50a, QOP_D3_MDWF_madd_fermion 49b, QOP_D3_MDWF_madd_half_fermion 49b, QOP_D3_MDWF_norm2_fermion 50b, and QOP_D3_MDWF_norm2_half_fermion 50b.

3.20 Debugging functions

Debugging functions give direct access to internal plumbing of the library. Every operator[1] that is used by preconditioned solvers can be accessed by the following function:

51c ⟨*Interface functions* 22b⟩+≡

```
int QOP_F3_MDWF_debugmesilly(struct QOP_F3_MDWF_Fermion *y,
                             const struct QOP_MDWF_Parameters *params,
                             const struct QOP_F3_MDWF_Gauge *gauge,
                             const char *op_name,
                             const struct QOP_F3_MDWF_Fermion *x);
int QOP_D3_MDWF_debugmesilly(struct QOP_D3_MDWF_Fermion *y,
                             const struct QOP_MDWF_Parameters *params,
                             const struct QOP_D3_MDWF_Gauge *gauge,
                             const char *op_name,
                             const struct QOP_D3_MDWF_Fermion *x);
```

This code is used in chunk 21a.
Defines:
QOP_D3_MDWF_debugmesilly, never used.
QOP_F3_MDWF_debugmesilly, never used.
Uses QOP_D3_MDWF_Fermion 29b, QOP_D3_MDWF_Gauge 27b, QOP_F3_MDWF_Fermion 29b, QOP_F3_MDWF_Gauge 27b, and QOP_MDWF_Parameters 23b.

[1] That is, if the developers ever suspected that any of these functions was incorrectly implemented. Additional hook-ups are rather straightforward to implement when necessary.

This function takes a Dirac fermion `x`, applies the operator `op_name` to it, and returns a Dirac fermion in `y`. If the operator can act only on the even part, the odd part of the argument `x` will be ignored and that of the result `y` will be zero upon return. If versions of the operator exist for both even and odd arguments, the result `y` will contain appropriate values for both. Note that this function is extremely experimental and appropriate values for `op_name` may change without notice; the only way to know which operators are supported is to examine the code.

Appendix A

CODE CHUNKS

⟨*Double precision defaults* 28c⟩ 21a, 28c, 33a, 36b, 38a, 39d, 47b, 48c, 51b
⟨*File* `../port/qop-mdwf3.h` 21a⟩ 21a
⟨*File* `../utils/basis.ss` 10b⟩ 10b
⟨*Interface functions* 22b⟩ 21a, 22b, 22c, 22d, 22e, 23d, 24a, 24b, 24c, 25a, 25b, 25c, 26a, 26b, 26c, 27a, 27c, 28a, 29a, 29c, 30a, 30b, 31a, 31b, 32a, 32b, 33b, 34b, 34c, 35a, 35b, 35c, 37a, 37b, 37c, 38b, 38c, 39a, 39b, 40b, 40c, 40e, 41a, 41b, 41c, 41d, 41e, 42a, 42b, 42c, 42d, 42e, 42f, 42g, 43b, 44a, 44b, 45, 46a, 46b, 48a, 49a, 49b, 50a, 50b, 51c
⟨*Interface macros* 21b⟩ 21a, 21b, 21c, 22a
⟨*Interface types* 23a⟩ 21a, 23a, 23b, 23c, 27b, 29b, 34a, 36c, 40a, 40d, 43a
⟨*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, 10a, 10b
⟨*Single precision defaults* 28b⟩ 21a, 28b, 32c, 36a, 37d, 39c, 47a, 48b, 51a
⟨*Start* μ *sum* 9c⟩ 9c, 10a
⟨*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

QDP_MDWF_DxD_CG: 47a, 47b
QOP_D3_MDWF_allocate_fermion: 29c, 33a
QOP_D3_MDWF_allocate_half_fermion: 34b, 36b
QOP_D3_MDWF_allocate_vector_fermion: 37a, 38a
QOP_D3_MDWF_axial_current: 32a, 33a
QOP_D3_MDWF_blas_from_half_fermion: 35a, 36b
QOP_D3_MDWF_DDW_CG: 43b, 47b
QOP_D3_MDWF_DDW_operator: 38b, 39d
QOP_D3_MDWF_DDW_operator_conjugated: 38c, 39d
QOP_D3_MDWF_debugmesilly: 51c
QOP_D3_MDWF_dot_fermion: 50a, 51b
QOP_D3_MDWF_dot_half_fermion: 50a, 51b
QOP_D3_MDWF_DxD_CG: 44b, 47b
QOP_D3_MDWF_export_4d_fermion: 31a, 33a
QOP_D3_MDWF_export_fermion: 30a, 33a
QOP_D3_MDWF_export_half_fermion: 34c, 36b
QOP_D3_MDWF_Fermion: 29a, 29b, 29c, 30a, 30b, 31a, 31b, 32a, 32b, 33a, 38b, 38c, 43b, 44a, 44b, 46b, 49b, 50a, 50b, 51c
QOP_D3_MDWF_free_fermion: 32b, 33a
QOP_D3_MDWF_free_gauge: 28a, 28c
QOP_D3_MDWF_free_half_fermion: 35c, 36b
QOP_D3_MDWF_free_vector_fermion: 37b, 38a
QOP_D3_MDWF_Gauge: 27a, 27b, 27c, 28a, 28c, 32a, 38b, 38c, 39a, 39b, 43b, 44a, 44b, 45, 46a, 46b, 48a, 51c
QOP_D3_MDWF_get_vector_fermion: 37c, 38a
QOP_D3_MDWF_HalfFermion: 33b, 34a, 34b, 34c, 35a, 35c, 36b, 37c, 39a, 39b, 45, 46a, 48a, 49b, 50a, 50b
QOP_D3_MDWF_half_fermion_from_blas: 35a, 36b
QOP_D3_MDWF_import_4d_fermion: 30b, 33a
QOP_D3_MDWF_import_fermion: 29a, 33a
QOP_D3_MDWF_import_gauge: 27a, 28c
QOP_D3_MDWF_import_half_fermion: 33b, 36b
QOP_D3_MDWF_madd_fermion: 49b, 51b
QOP_D3_MDWF_madd_half_fermion: 49b, 51b
QOP_D3_MDWF_midpoint_pseudo: 31b, 33a
QOP_D3_MDWF_M_operator: 39a, 39d
QOP_D3_MDWF_M_operator_conjugated: 39b, 39d
QOP_D3_MDWF_MxM_CG: 45, 47b
QOP_D3_MDWF_MxM_poly: 48a, 48c
QOP_D3_MDWF_MxM_SCG: 46a, 47b
QOP_D3_MDWF_norm2_fermion: 50b, 51b
QOP_D3_MDWF_norm2_half_fermion: 50b, 51b
QOP_D3_MDWF_put_vector_fermion: 37c, 38a
QOP_F3_MDWF_allocate_fermion: 29c, 32c

QOP_F3_MDWF_allocate_half_fermion: 34b, 36a
QOP_F3_MDWF_allocate_vector_fermion: 37a, 37d
QOP_F3_MDWF_alloc_half_fermion_matrix: 40e
QOP_F3_MDWF_axial_current: 32a, 32c
QOP_F3_MDWF_blas_from_half_fermion: 35a, 36a
QOP_F3_MDWF_blas_view_half_fermion_matrix: 41a
QOP_F3_MDWF_create_deflator: 40b
QOP_F3_MDWF_create_deflator_inplace: 40c
QOP_F3_MDWF_DDW_CG: 43b, 47a
QOP_F3_MDWF_DDW_operator: 38b, 39c
QOP_F3_MDWF_DDW_operator_conjugated: 38c, 39c
QOP_F3_MDWF_debugmesilly: 51c
QOP_F3_MDWF_Deflator: 40a, 40b, 40c, 41c, 41d, 41e, 42a, 42b, 42c, 42d, 42e, 42f, 42g, 46b
QOP_F3_MDWF_deflator_add_vector: 42e
QOP_F3_MDWF_deflator_current_dim: 42b
QOP_F3_MDWF_deflator_eigcg_is_stopped: 42a
QOP_F3_MDWF_deflator_eigcg_reset: 41d
QOP_F3_MDWF_deflator_eigcg_resume: 41e
QOP_F3_MDWF_deflator_eigcg_stop: 41e
QOP_F3_MDWF_deflator_eigen: 42g
QOP_F3_MDWF_deflator_extract_vector: 42c
QOP_F3_MDWF_deflator_start_load: 42d
QOP_F3_MDWF_deflator_stop_load: 42f
QOP_F3_MDWF_dot_fermion: 50a, 51a
QOP_F3_MDWF_dot_half_fermion: 50a, 51a
QOP_F3_MDWF_DxD_CG: 44b, 47a
QOP_F3_MDWF_export_4d_fermion: 31a, 32c
QOP_F3_MDWF_export_fermion: 30a, 32c
QOP_F3_MDWF_export_half_fermion: 34c, 36a
QOP_F3_MDWF_Fermion: 29a, 29b, 29c, 30a, 30b, 31a, 31b, 32a, 32b, 32c, 38b, 38c, 43b, 44b, 49b, 50a, 50b, 51c
QOP_F3_MDWF_free_deflator: 41c
QOP_F3_MDWF_free_fermion: 32b, 32c
QOP_F3_MDWF_free_gauge: 28a, 28b
QOP_F3_MDWF_free_half_fermion: 35c, 36a
QOP_F3_MDWF_free_vector_fermion: 37b, 37d
QOP_F3_MDWF_Gauge: 27a, 27b, 27c, 28a, 28b, 32a, 38b, 38c, 39a, 39b, 40c, 42e, 43b, 44b, 45, 46a, 48a, 51c
QOP_F3_MDWF_get_vector_fermion: 37c, 37d
QOP_F3_MDWF_HalfFermion: 33b, 34a, 34b, 34c, 35a, 35c, 36a, 37c, 39a, 39b, 42c, 42e, 45, 46a, 48a, 49b, 50a, 50b
QOP_F3_MDWF_half_fermion_from_blas: 35a, 36a
QOP_F3_MDWF_HalfFermionMat: 40c, 40d, 40e, 41a, 41b
QOP_F3_MDWF_import_4d_fermion: 30b, 32c
QOP_F3_MDWF_import_fermion: 29a, 32c
QOP_F3_MDWF_import_gauge: 27a, 28b
QOP_F3_MDWF_import_half_fermion: 33b, 36a
QOP_F3_MDWF_madd_fermion: 49b, 51a
QOP_F3_MDWF_madd_half_fermion: 49b, 51a
QOP_F3_MDWF_midpoint_pseudo: 31b, 32c
QOP_F3_MDWF_M_operator: 39a, 39c
QOP_F3_MDWF_M_operator_conjugated: 39b, 39c
QOP_F3_MDWF_MxM_CG: 45, 47a
QOP_F3_MDWF_MxM_poly: 48a, 48b
QOP_F3_MDWF_MxM_SCG: 46a, 47a
QOP_F3_MDWF_norm2_fermion: 50b, 51a
QOP_F3_MDWF_norm2_half_fermion: 50b, 51a
QOP_F3_MDWF_put_vector_fermion: 37c, 37d
QOP_MDWF_allocate_fermion: 32c, 33a

QOP_MDWF_allocate_half_fermion: 36a, 36b
QOP_MDWF_allocate_vector_fermion: 37d, 38a
QOP_MDWF_axial_current: 32c, 33a
QOP_MDWF_0bd50d0caeec4311a0d7c183032c43c2: 21a
QOP_MDWF_blas_from_half_fermion: 36a, 36b
QOP_MDWF_COLORS: 22a
QOP_MDWF_Config: 23c, 23d
QOP_MDWF_DDW_CG: 47a, 47b
QOP_MDWF_DDW_operator: 39c, 39d
QOP_MDWF_DDW_operator_conjugated: 39c, 39d
QOP_MDWF_deflated_mixed_D_CG: 46b
QOP_MDWF_DIM: 21b
QOP_MDWF_dot_fermion: 51a, 51b
QOP_MDWF_dot_half_fermion: 51a, 51b
QOP_MDWF_error: 24b
QOP_MDWF_export_4d_fermion: 32c, 33a
QOP_MDWF_export_fermion: 32c, 33a
QOP_MDWF_export_half_fermion: 36a, 36b
QOP_MDWF_Fermion: 32c, 33a
QOP_MDWF_FERMION_DIM: 21c
QOP_MDWF_FINAL_CG_RESIDUAL: 43a
QOP_MDWF_FINAL_DIRAC_RESIDUAL: 43a
QOP_MDWF_fini: 24a
QOP_MDWF_free_fermion: 32c, 33a
QOP_MDWF_free_gauge: 28b, 28c
QOP_MDWF_free_half_fermion: 36a, 36b
QOP_MDWF_free_parameters: 26c
QOP_MDWF_free_vector_fermion: 37d, 38a
QOP_MDWF_Gauge: 28b, 28c
QOP_MDWF_gauge_float_from_double: 27c
QOP_MDWF_get_vector_fermion: 37d, 38a
QOP_MDWF_HalfFermion: 36a, 36b
QOP_MDWF_half_fermion_from_blas: 36a, 36b
QOP_MDWF_half_fermion_size: 35b
QOP_MDWF_import_4d_fermion: 32c, 33a
QOP_MDWF_import_fermion: 32c, 33a
QOP_MDWF_import_gauge: 28b, 28c
QOP_MDWF_import_half_fermion: 36a, 36b
QOP_MDWF_init: 23d
QOP_MDWF_LOG_CG_RESIDUAL: 43a
QOP_MDWF_LOG_EIG_POSTAMBLE: 43a
QOP_MDWF_LOG_EIG_UPDATE1: 43a
QOP_MDWF_LOG_EVERYTHING: 43a
QOP_MDWF_LOG_NONE: 43a
QOP_MDWF_LOG_TRUE_RESIDUAL: 43a
QOP_MDWF_madd_fermion: 51a, 51b
QOP_MDWF_madd_half_fermion: 51a, 51b
QOP_MDWF_midpoint_pseudo: 32c, 33a
QOP_MDWF_mixed_DDW_CG: 44a
QOP_MDWF_M_operator: 39c, 39d
QOP_MDWF_M_operator_conjugated: 39c, 39d
QOP_MDWF_MxM_CG: 47a, 47b
QOP_MDWF_MxM_poly: 48b, 48c
QOP_MDWF_MxM_SCG: 47a, 47b
QOP_MDWF_norm2_fermion: 51a, 51b
QOP_MDWF_norm2_half_fermion: 51a, 51b

QOP_MDWF_Parameters: 23b, 25a, 25b, 25c, 26a, 26b, 26c, 38b, 38c, 39a, 39b, 40c, 42e, 43b, 44a, 44b, 45, 46a, 46b, 48a, 51c
QOP_MDWF_parity: 22d
QOP_MDWF_performance: 22e
QOP_MDWF_poly_normalize: 49a
QOP_MDWF_PROJECTED_FERMION_DIM: 21c
QOP_MDWF_put_vector_fermion: 37d, 38a
QOP_MDWF_set_Borichi: 26a
QOP_MDWF_set_Chiu: 26b
QOP_MDWF_set_complex: 25a
QOP_MDWF_set_generic: 25a
QOP_MDWF_set_Moebius: 25b
QOP_MDWF_set_Shamir: 25c
QOP_MDWF_set_threads: 24c
QOP_MDWF_signature: 22c
QOP_MDWF_State: 22c, 22d, 22e, 23a, 23d, 24a, 24b, 24c, 25a, 25b, 25c, 26a, 26b, 27a, 29a, 29c, 30b, 33b, 34b, 35b, 37a, 40b, 40e
QOP_MDWF_VectorFermion: 37d, 38a
QOP_MDWF_version: 22b