Cuda Lattice Gauge Document

Ji-Chong Yang

目录

1 Data

1.1 Index of lattice

1.1.1 UINT Index of lattice

Generally, in CLG, we have three kinds of indexes:

- **site index**
- **link index**
- **fat index**

Let the lattice have $V = L_x \times L_y \times L_z \times L_t$ sites.

Note: for $D = 3$, we assume $L_x = 1$, $L_{y,z,t} > 1$; for $D = 2$, we assume $L_x = L_y = 1$, $L_{z,t} > 1$.

For a site at (x, y, z, t)

$$siteIndex = x \times L_y \times L_z \times L_t + y \times L_z \times L_t + z \times L_t + t \tag{1}$$

For a link at direction dir, link with site at (x, y, z, t), and on a lattice with number of directions of links is $dirCount$,

$$linkIndex = siteIndex \times dirCount + dir \tag{2}$$

Note: we do NOT assume dimension equal number of links. For example for $D = 2$ triangle lattice, number of directions of links is 6, for $D = 2$ hexagon number of directions of links is 3. Only for square lattice, number of links equal dimension.

For a link at direction dir, link with site at (x, y, z, t), and on a lattice with number of directions of links is $dirCount$,

$$fatIndex = \begin{cases} siteIndex \times (dirCount + 1); & for\ site. \\ siteIndex \times (dirCount + 1) + (dir + 1); & for\ link \end{cases} \tag{3}$$

1.1.2 SIndex of lattice

1.1.3 Index and boundary condition, a int2 or a uint2 structure

In CLGLib, sometimes, the index function return a uint2 structure.

1.1.4 Index walking

1.2 CParemeters

2 Update scheme

2.1 HMC

HMC is abbreviation for hybrid Monte Carlo.

2.1.1 The Fermion action

Cooperating with HMC, the fermion is usually the 'Pseudofermions'.

We begin with Eq. (1.85) and Eq. (1.86) of Ref. [1].

$$Z = \int \mathcal{D}[U] \prod_{f=1}^{N_f} \mathcal{D}[\bar{\psi}_f]\mathcal{D}[\psi_f] \exp\left(-S_G[U] - \sum_{f=1}^{N_f} \bar{\psi}_f \left(\hat{D}_f\right) \psi_f\right) \tag{4}$$

where $\hat{D}_f = D + m_f$. (Note that, there seems a typo in Eq. (1.85), we have $S_F = +\bar{\psi} D \psi$, see also Eq. (5.39) of Ref. [2] and Eq. (7.6) of Ref. [3], Eq. (3.75) of Ref. [1], etc.)

It can be evaluated as Eq. (1.86) of Ref. [1] (or Eq. (4.19) of Ref. [4]) (Note, there is another minus sign in Eq. (5.28) of Ref. [2])

$$\begin{aligned} \int \mathcal{D}\bar{\psi}\psi \exp\left(-\bar{\psi} A \psi\right) &= \det\left(A\right), \\ Z = \prod_{f=1}^{N_f} \det\left(\hat{D}_f\right) &\int \mathcal{D}[U] \exp\left(-S_G[U]\right). \end{aligned} \tag{5}$$

On the other hand, with the help of Gaussian integral of complex vectors Eq. (3.17) of Ref. [4]

$$\int d\mathbf{v}^\dagger d\mathbf{v} \exp(-\mathbf{v}^\dagger \mathbf{A} \mathbf{v}) = \pi^N \left(\det \mathbf{A}\right)^{-1} \tag{6}$$

which is (3.31) of Ref. [1]

$$\frac{1}{\det(\mathbf{A})} = \int \mathcal{D}[\eta] \exp(-\eta^\dagger \mathbf{A} \eta) \tag{7}$$

where η now is a complex Bosonic field, and the normalization

$$\mathcal{D}[\eta] = \prod \frac{d\mathrm{Re}(\eta_i) d\mathrm{Im}(\eta_i)}{\pi}, \quad 1 = \int \mathcal{D}[\eta] \exp(-\eta^\dagger \eta) \tag{8}$$

is assumed. With the condition such that

$$\lambda(\mathbf{A} + \mathbf{A}^\dagger) > 0. \tag{9}$$

where $\lambda(\mathbf{M})$ denoted as eigen-values of $\mathbf{M}$.

We now, concentrate on two degenerate fermion flavours. i.e. considering

$$S_F = \bar{\psi}_u \hat{D} \psi_u + \bar{\psi}_d \hat{D} \psi_d. \tag{10}$$

Using $\det(DD^\dagger) = \det(D)\det(D^\dagger)$ and $\det(M^{-1}) = (\det(M))^{-1}$ and $\det(D) = \det(D^\dagger)$ (**Only for Wilson Fermions or γ_5-hermiticity fermions, $\hat{D}^\dagger = \gamma_5 D \gamma_5 + m = \gamma_5(D + m)\gamma_5 = \gamma_5 \hat{D} \gamma_5$, and $\det(\hat{D}^\dagger) = \det(\gamma_5)\det(\hat{D})\det(\gamma_5) = \det(\hat{D})$.** See also Ref. [5].), one can show Eq. (8.9) of Ref. [2] (Eq. (2.77) of Ref. [6])

$$\int \mathcal{D}[\bar{\psi}]\mathcal{D}[\psi] \exp\left(-\bar{\psi}_u \hat{D} \psi_u - \bar{\psi}_d \hat{D} \psi_d\right) = \det(\hat{D}\hat{D}^\dagger) = \int \mathcal{D}[\phi] \exp(-\phi^\dagger \left(\hat{D}\hat{D}^\dagger\right)^{-1} \phi) \tag{11}$$

where ϕ now is a complex Bosnic field. (Note that, there is a sign typo in Eq. (8.31) of Ref. [2], see also Eqs. (8.38) and (8.39) of Ref. [2])

So, generally, we are using HMC to evaluate the action with 'Pseudofermions', or in other words, we are working with an action including only gauge and bosons.

$$S = S_G + S_{pf} = S_G + \phi^\dagger \left(\hat{D}\hat{D}^\dagger\right)^{-1} \phi \tag{12}$$

where pf is short for pseudofermion.

2.1.2 Basic idea, force from gauge field

The basic idea is to use a molecular dynamics simulation, i.e, it is a integration of Langevin equation.

Treating $SU(N)$ matrix U on links as coordinate, HMC will generate a pair of configurations, (P, U), where P is momentum and $P \in \mathfrak{su}(N)$.

One can:

1. Create a random $P = i \sum_a \omega_a T_a$, where $\omega_a \in \mathbb{R}$.
2. Obtain $\dot{P}$, $\dot{U}$. Note that, dot is $d/d\tau$, where τ is 'Markov time'.
3. Numerically evaluate the differential equation, and use a Metropolis accept / reject to update.

- About the randomized P

The randomized P is chosen according to normal distribution $\exp(-P^2/2)$

Note that, here P corresponds to Q, not U, for $U = \exp(i\sum q_a T^a)$, there are 8 **real** variables denoting as ω_i.

Using $P = \sum \omega_a T^a$, $tr((T^a)\cdot(T^b)) = \frac{1}{2}\delta_{ab}$. So one have $\frac{1}{2}\sum_a \omega_a^2 = tr[P^2]$.

It is usually written as distribution $\exp(-tr(P^2))$ (where P is a matrix, and $tr[P^2] = \frac{1}{2}p^2$ where $p = (\omega_1, \omega_2, \ldots, \omega_8)$).

Using the property of normal distribution

$$\begin{aligned} &if\ \{X\} \sim N(\mu_X, \sigma_X^2),\ \ \{Y\} \sim N(\mu_Y, \sigma_Y^2), \\ &\{X+Y\} \sim N(\mu_X + \mu_Y, \sigma_X^2 + \sigma_Y^2). \end{aligned} \tag{13}$$

One can randomize ω_a using $\exp(-\omega_a\omega_a)$. Then using $P = \frac{1}{\sqrt{8N}}\sum \omega_a T^i$, where N is the number of links.

Note: Here is a difference between Refs. [2] and [1] and Bridge++ [7]

Note, by Eq. (8.16) of Ref. [2], $P^2 = \sum_{n\in\Lambda} P^2(n)$, so when the lattice is large, P become very small. See also the definition of $\langle P, P\rangle$ below Eq. (2.42) of Ref. [1].

However, in Bridge++, it uses distribution $\exp(-tr(P^2)/DOF)$, where 'DOF' is the degrees of freedom, i.e., number of links.

We use the distribution same as in Bridge++. Imagining that for a very small (hot) $\beta \to 0$, the force is also almost 0 so momentum is unchanged when evolution. Considering a very large lattice such that the momentum is very small when using distribution $\exp(-tr(P^2))$, the gauge field will stay near the initial value rather then becoming hot (randomized). So we think it should be $\exp(-tr(P^2)/DOF)$.

- Force

Defined by Newton, dp/dt is a force, so $\dot{P}$ is called 'force'. See Eqs. (2.53), (2.56) and (2.57) of Ref. [1], for $SU(N)$,

$$\begin{aligned} S_G[U_\mu(n)] &= -\frac{\beta}{N}\mathrm{Retr}\left[U_\mu(n)\Sigma_\mu^\dagger(n)\right] \\ \Sigma_\mu(n) &= \sum_{\mu\neq\nu}\left(U_\nu(n)U_\mu(n+a\nu)U_\nu^{-1}(n+a\mu) + U_\nu^{-1}(n-a\nu)U_\mu(n-a\nu)U_\nu(n-a\nu+a\mu)\right) \end{aligned} \tag{14}$$

Note that $S_G \neq \sum_{\mu,n} S_G[U_\mu(n)]$. $S_G[U_\mu(n)]$ is convenient for derivate which collecting all terms related to the specified bond. For plaquettes with 4 edges, $S_G = \frac{1}{4}\sum_{\mu,n} S_G[U_\mu(n)]$.

S_G the action for a particular $U_\mu(n)$. Σ is the 'staple'(see Eq. (75)). The staple for $U_\mu(n)$ is independent of $U_\mu(n)$, denoting

$$U_\mu(n) = \exp\left(i\sum_a \omega_a(\mu,n)T_a\right) U_\mu^0(n) \tag{15}$$

so

$$\begin{aligned}
&\frac{\partial}{\partial\omega_a(\mu,n)}S_G = -\frac{\beta}{2N}\mathrm{Retr}\left[\frac{\partial}{\partial\omega_a}U_\mu(n)\Sigma_\mu^\dagger(n)\right] = -\frac{\beta}{2N}\mathrm{tr}\left[\frac{\partial}{\partial\omega_a}\left(U_\mu(n)\Sigma_\mu^\dagger(n)+\Sigma_\mu(n)U_\mu^\dagger(n)\right)\right]\\
&= -i\frac{\beta}{2N}\mathrm{tr}\left[T_aU_\mu(n)\Sigma_\mu^\dagger(n)-\Sigma_\mu(n)T_a^\dagger U_\mu^\dagger(n)\right] = -i\frac{\beta}{2N}\mathrm{tr}\left[T_a\left(U_\mu(n)\Sigma_\mu^\dagger(n)-\Sigma_\mu(n)U_\mu^\dagger(n)\right)\right]\\
&= \frac{\beta}{N}\mathrm{Im\ tr}\left[T_aU_\mu(n)\Sigma_\mu^\dagger(n)\right]
\end{aligned} \tag{16}$$

This is the Eq. (8.41) of Ref. [2].

Using (Checked by Mathematica that Eq. (8.42) of Ref. [2] is incompatable with our notation, but replacing the $UA - A^\dagger U^\dagger$ of Eq. (8.42) with $\{UA\}_{TA}$ is correct. Also, Eq. (2.58) of Ref. [1] is different from ours, in our formulism, it is correct by replacing $2T_a\mathrm{Re}[tr[T_a\cdot W]]$ of Eq. (2.58) with $2iT_a\mathrm{Im}[tr[T_a\cdot W]]$)

$$\begin{aligned}
&\sum_a \mathrm{tr}\left[T_a\left(U_\mu(n)\Sigma_\mu^\dagger(n)-\Sigma_\mu(n)U_\mu^\dagger(n)\right)\right]T_a = 2i\sum_a \mathrm{Im}\left[T_aU_\mu(n)\Sigma_\mu^\dagger(n)\right]T_a = \{U_\mu(n)\Sigma_\mu^\dagger(n)\}_{TA}\\
&\{W\}_{TA} = \frac{W-W^\dagger}{2} - \mathrm{tr}\left(\frac{W-W^\dagger}{2N}\right)\mathbb{I}
\end{aligned} \tag{17}$$

where $\mathbb{I}$ is identity matrix. Therefor

$$\begin{aligned}
&\dot\omega_a = -\frac{\partial}{\partial\omega_a(\mu,n)}S_G\\
&F_\mu(x) = \dot P_\mu(x) = i\sum \dot\omega_aT_a = -i\frac{\partial}{\partial\omega_a(\mu,n)}S_GT_a = -\frac{\beta}{2N}\{U_\mu(n)\Sigma_\mu^\dagger(n)\}_{TA}
\end{aligned} \tag{18}$$

Note that, $\dot\omega_a = \frac{\beta}{N}\mathrm{Im}[tr[T_a\cdot W]]$ is still a **real** number.

Eq. (18) is same as Eqs. (2.53), (2.56) and (2.57) of Ref. [1].

- Integrator

  Knowing $\dot P$, and $\dot U$, to obtain U and P is simply

$$U(\tau+d\tau)\approx \dot U d\tau + U(\tau),\quad P(\tau+d\tau)\approx \dot P d\tau + P(\tau) \tag{19}$$

A more accurate calculation is done by integrator, for example, the leap frog integrator, the M step leap frog integral is described in Ref. [2],

$$\epsilon = \frac{\tau}{M} \tag{20a}$$

$$U_\mu(x,(n+1)\epsilon) = U_\mu(x,n\epsilon) + \epsilon P_\mu(x,n\epsilon) + \tfrac{1}{2}F_\mu(x,n\epsilon)\epsilon^2 \tag{20b}$$

$$P_\mu(x,(n+1)\epsilon) = P_\mu(x,n\epsilon) + \tfrac{1}{2}\left(F_\mu(x,(n+1)\epsilon) + F_\mu(x,n\epsilon)\right)\epsilon \tag{20c}$$

So, knowing $U(n\epsilon)$ we can calculate $F(n\epsilon)$ using Eq. (18). Knowing $U(n\epsilon), P(n\epsilon), F(n\epsilon)$, we can calculate $U((n+1)\epsilon)$ using Eq. (20).b. Then we are able to calculate $F((n+1)\epsilon)$ again using Eq. (18). Then we can calculate $P((n+1)\epsilon)$ using Eq. (20).c.

2.1.3 Force of pseudofermions

For important sampling, one can generate both U and ϕ by e^{-S}. In molecular dynamics simulation, it can be simplified as:

1. Evaluate U use force of U and ϕ on U.
2. Evaluate ϕ use force of U and ϕ on ϕ.

The second step can be simplified as, generating random complex numbers ϕ according to $\exp(-\phi^\dagger\left(\hat{D}\hat{D}^\dagger\right)^{-1}\phi) = \exp(-\phi^\dagger(\hat{D}^\dagger)^{-1}\hat{D}^{-1}\phi)$. $D[U]$ is a function of U.

How to get randomized ϕ? Let χ be random **complex** numbers according to $\exp(-\chi^\dagger\chi)$. Let $\hat{D}^{-1}\phi = \chi$, ϕ is the random **complex** number satisfying distribution we want $(\exp(-\phi^\dagger(\hat{D}^\dagger)^{-1}\hat{D}^{-1}\phi))$. So, first get χ and then let $\phi = D\chi$.

Using the Wilson Fermion action

$$\begin{aligned}\hat{D} &= C(D+1)\\ D &= -\kappa\sum_\mu\left((1-\gamma_\mu)U_\mu(x_L)\delta_{x_L,(x+\mu)_R} + (1+\gamma_\mu)U_\mu^{-1}(x_L-\mu)\delta_{x_L,(x-\mu)_R}\right)\end{aligned} \tag{21}$$

with $C = m_f + (4/a) = 1/2a\kappa$ and $\kappa = 1/(2am_f+8)$. One can rescale the field and set $C=1$.

The force of ϕ on U is obtained as $\partial_{\omega_a}S_{pf}$. The result for Wilson Fermion action is shown

in Eqs. (8.39), (8.44) and (8.45) of Ref. [2] as

$$
\begin{aligned}
&F = i\sum_a \dot{\omega}_a T_a = i\sum_a \left(-\partial_{\omega_a}\left(S_G[U_\mu(n)] + S_{pf}[U_\mu(n)]\right)\right) T_a = F_G + F_{pf}.\\
&F_{pf} = i\sum_a \left(-\partial_{\omega_a} S_{pf}[U_\mu(n)]\right) T_a = -i\sum_a T^a \frac{\partial}{\partial\omega_a}\left(\phi^\dagger\left(\hat{D}\hat{D}^\dagger\right)^{-1}\phi\right).\\
&\frac{\partial}{\partial\omega_a}\left(\phi^\dagger\left(\hat{D}\hat{D}^\dagger\right)^{-1}\phi\right) = -\left(\left(\hat{D}\hat{D}^\dagger\right)^{-1}\phi\right)^\dagger\left(\frac{\partial D}{\partial\omega_\mu^a}\hat{D}^\dagger + \hat{D}\frac{\partial D^\dagger}{\partial\omega_\mu^a}\right)\left(\left(\hat{D}\hat{D}^\dagger\right)^{-1}\phi\right).\\
&\frac{\partial\hat{D}}{\partial\omega_\mu^a} = \left(\frac{\partial D}{\partial\omega_\mu^a}\right)_{x_L.x_R} = -i\kappa\left\{(1-\gamma_\mu)T^a U_\mu(x)\delta_{x,x_L}\delta_{x,(x+\mu)_R} - (1+\gamma_\mu)U_\mu^{-1}(x)T^a\delta_{x,(x+\mu)_L}\delta_{x,x_R}\right\}\\
&\hat{D}^\dagger = \gamma_5\hat{D}\gamma_5,\quad \frac{\partial D^\dagger}{\partial\omega_\mu^i} = \gamma_5\frac{\partial D}{\partial\omega_\mu^a}\gamma_5
\end{aligned}
\tag{22}
$$

where F_G is force from U introduced in Sec. 2.1.2, T^a are $SU(3)$ generators. x_L, x_R are coordinate index of the left and right pseudofermion field. And

$$
\begin{aligned}
&U_\mu = \exp(i\sum_a \omega_\mu^a T^a)U_0,\quad \frac{\partial U_\mu}{\partial\omega_\mu^a} = iT^a U_\mu,\quad \frac{\partial U_\mu^\dagger}{\partial\omega_\mu^a} = -iU_\mu^\dagger T^a,\\
&(T^a)^\dagger = T^a,\quad \frac{\partial M^{-1}}{\partial\omega_\mu^a} = -M^{-1}\frac{\partial M}{\partial\omega_\mu^a}M^{-1}
\end{aligned}
\tag{23}
$$

are used. (Note that, Eq. (8.45) of Ref. [2] has a sign typo, see also Eq. (2.82) of Ref. [6])

We can simplify it further by $\left(\hat{D}^\dagger(\hat{D}\hat{D}^\dagger)^{-1}\phi\right)^\dagger = \left((\hat{D}\hat{D}^\dagger)^{-1}\phi\right)^\dagger\hat{D}$, so

$$
\begin{aligned}
&\phi_1 = \left(\left(\hat{D}\hat{D}^\dagger\right)^{-1}\phi\right),\quad \phi_2 = \hat{D}^\dagger\left(\left(\hat{D}\hat{D}^\dagger\right)^{-1}\phi\right) = D^{-1}\phi,\quad \phi_1^\dagger D = \phi_2^\dagger,\\
&\frac{\partial}{\partial\omega_a}\left(\phi^\dagger\left(\hat{D}\hat{D}^\dagger\right)^{-1}\phi\right) = -\left(\left(\hat{D}\hat{D}^\dagger\right)^{-1}\phi\right)^\dagger\left(\frac{\partial D}{\partial\omega_\mu^a}\hat{D}^\dagger + \hat{D}\frac{\partial D^\dagger}{\partial\omega_\mu^a}\right)\left(\left(\hat{D}\hat{D}^\dagger\right)^{-1}\phi\right)\\
&= -\left(\phi_1^\dagger\frac{\partial D}{\partial\omega_\mu^a}\phi_2 + \phi_2^\dagger\frac{\partial D^\dagger}{\partial\omega_\mu^a}\phi_1\right) = -2\mathrm{Re}\left[\left(\phi_1^\dagger\frac{\partial D}{\partial\omega_\mu^a}\phi_2\right)\right]
\end{aligned}
\tag{24}
$$

and

$$
\begin{aligned}
&\frac{\partial D}{\partial\omega_\mu^a} = -i\kappa M_a,\\
&(M_a)_{x_L,x_R} = \left\{(1-\gamma_\mu)T^a U_\mu\delta_{x_L,(x+\mu)_R} - (1+\gamma_\mu)U_\mu^{-1}T^a\delta_{(x+\mu)_L,x_R}\right\}\\
&\frac{\partial}{\partial\omega_a}\left(\phi^\dagger\left(\hat{D}\hat{D}^\dagger\right)^{-1}\phi\right) = -2\kappa\mathrm{Im}\left[\left(\phi_1^\dagger M\phi_2\right)\right]
\end{aligned}
\tag{25}
$$

Again, $\dot{\omega}$ is a **real** number, and

$$F_{pf} = -i\sum_a T^a \frac{\partial}{\partial \omega_a}\left(\phi^\dagger \left(\hat{D}\hat{D}^\dagger\right)^{-1}\phi\right) = 2i\kappa \sum_a \mathrm{Im}\left[\left(\phi_1^\dagger M_a \phi_2\right)\right] T_a \tag{26}$$

So we can calculate ϕ_1 first, then $\phi_2 = \hat{D}^\dagger\phi_1$. Then contract the spinor and color space with $\partial D/\partial\omega$.

Note that, D is changing when integrating the Langevin equation.

The last part is how to calculate $(\hat{D}\hat{D}^\dagger)^{-1}$.

- Anti-Hermitian traceless of the force

  See from Eq. (18), the force from the gauge field is an anti-Hermitian traceless matrix.

  The result above can be further simplified. Note that

$$\begin{aligned} &\phi_{L1}(n) = \phi_1(n), \quad \phi_{R1}(n) = (1-\gamma_\mu)\phi_2(n+\mu), \\ &\phi_{L2}(n) = \phi_1(n+\mu), \quad \phi_{R1}(n) = (1+\gamma_\mu)\phi_2(n), \end{aligned} \tag{27}$$

One have

$$\begin{aligned} &\mathrm{Im}\left[\phi_1^\dagger M \phi_2\right]_\mu^a(n) = \mathrm{Im}\left[\phi_{L1}^\dagger T^a U_\mu(n)\phi_{R1}\right] - \mathrm{Im}\left[\phi_{L2}^\dagger U_\mu^\dagger(n) T^a \phi_{R2}\right] \\ &= \mathrm{Im}\left[\phi_{L1}^\dagger T^a U_\mu(n)\phi_{R1}\right] + \mathrm{Im}\left[\phi_{R2}^\dagger T^a U_\mu(n)\phi_{L2}\right] \end{aligned} \tag{28}$$

For any vector

$$\mathrm{Im}\left[L^\dagger T U R\right] = \mathrm{Im}\left[\sum_{\alpha,\beta,\rho} L_\alpha^* T_{\alpha\beta} U_{\beta\rho} R_\rho\right] = \mathrm{Im}\left[\sum_{\alpha,\beta,\rho} T_{\alpha\beta} U_{\beta\rho} R_\rho L_\alpha^*\right] = \mathrm{Im}\left[\mathrm{tr}\left[TU(RL^\dagger)\right]\right] \tag{29}$$

So

$$\begin{aligned} &F_\mu^{pf}(n) = 2i\kappa \mathrm{Im}\left[\phi_1^\dagger M \phi_2\right]_\mu(n) = \kappa\left(2i\sum_a \mathrm{Imtr}\left[T^a U_\mu(n)\left(\phi_{R1}\phi_{L1}^\dagger + \phi_{R2}\phi_{L2}^\dagger\right)\right] T^a\right) \\ &= \kappa\left\{U_\mu(n)\left(\phi_{R1}\phi_{L1}^\dagger + \phi_{R2}\phi_{L2}^\dagger\right)\right\}\Big|_{TA} \end{aligned} \tag{30}$$

which is also an anti-Hermitian traceless matrix.

So, the momentum is always anti-Hermitian traceless..

For anti-Hermitian traceless matrix M, the $\exp(M)$ can be simplified as Appendix. A of Ref. [8].

2.1.4 Solver in HMC

To calculate $(\hat{D}\hat{D}^\dagger)^{-1}$, we need a solver. The detail of solvers will be introduced in Sec. 3. Here we establish a simple introduction.

Let M be a matrix operating on a vector, for example, $M = (\hat{D}\hat{D}^\dagger)$, the goal of the solver is to find x such $b = M \cdot x$, and therefor $x = (\hat{D}\hat{D}^\dagger)^{-1}b$.

We first introduce the CG algorithm for real vector and real matrix, define

$$Q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \cdot A \cdot \mathbf{x} - \mathbf{x}^T\mathbf{b}. \tag{31}$$

so that one can try to find the minimum of Q, and at the minimum

$$\frac{\partial}{\partial \mathbf{x}} Q(\mathbf{x}) = 0 = A \cdot \mathbf{x} - \mathbf{b}. \tag{32}$$

To find the minimum, one can use gradient. Starting from a random point on a curve, calculate the falling speed and move it until it is stable.

For complex vector, one can use BiCGStab in Table. 6.2 in Ref. [2]. It can be described as

Algorithm 1 BiCGStab, note that, the numbers are **complex** number.

$\mathbf{x} = \mathbf{b}$ ▷ Use $\mathbf{b}$ as trail solution and start.
for $i = 0$ to r **do**
  $\mathbf{r} = \mathbf{b} - A\mathbf{x}$ ▷ Restart r times
  $\mathbf{r}_h = \mathbf{r}$
  for $j = 0$ to $itera$ **do**
    $\rho = \mathbf{r}_h^* \cdot \mathbf{r}_j$
    if $j = 0$ **then**
      $\mathbf{p} = \mathbf{r}$
    else
      $\beta = \alpha \times \rho / (\omega \times \rho_p)$
      $\mathbf{p} = \mathbf{r} + \beta\,(\mathbf{p} - \omega\mathbf{v})$
    end if
    $\mathbf{v} = A\mathbf{p}$
    $\alpha = \rho / \left(\mathbf{r}_h^* \cdot \mathbf{v}\right)$
    $\mathbf{s} = \mathbf{r} - \alpha\mathbf{v}$
    if $0 \neq j$ and $0 =$ mod $(j, 5)$ **then**
      $er = \|\mathbf{s}\|$ ▷ Check deviation every 5 steps
      if $er < \epsilon$ **then**
  return x
      end if
    end if
    $\mathbf{t} = A\mathbf{s}$
    $\omega = \mathbf{s}^* \cdot \mathbf{t} / \|\mathbf{t}\|$
    $\mathbf{r} = \mathbf{s} - \omega\mathbf{t}$
    $\mathbf{x} = \mathbf{x} + \alpha\mathbf{p} + \omega\mathbf{s}$
    $\rho_p = \rho$ ▷ Preserve the last calculated ρ become we still need it
  end for
end for

2.1.5 Leap frog integrator

In Sec. 2.1.2, the basic idea is introduced. However, the implementation is slightly different.

$$U_\mu(0,x) = gauge(x), \quad P_\mu(0,x) = \sum_a r_a(\mu,x)T_a \tag{33a}$$

$$F_\mu(n\epsilon,x) = -\tfrac{\beta}{2N}\{U_\mu(n\epsilon,x)\Sigma_\mu(n\epsilon,x)\}_{TA} \tag{33b}$$

$$P_\mu(\tfrac{1}{2}\epsilon,x) = P_\mu(0,x) + \tfrac{\epsilon}{2}F_\mu(0,x) \tag{33c}$$

$$U_\mu((n+1)\epsilon,x) = \exp\left(i\epsilon P_\mu((n+\tfrac{1}{2})\epsilon,x)\right)U_\mu(n\epsilon,x) \tag{33d}$$

$$P_\mu((n+\tfrac{1}{2})\epsilon,x) = P_\mu((n-\tfrac{1}{2})\epsilon,x) + \epsilon F_\mu(n\epsilon,x) \tag{33e}$$

Note that, the sign of F is '+' here which is different from Ref. [2], because in Ref. [2], $F = \partial_{\mu,n}S = -\dot{P}$. Here we define $F = \dot{P} = -\partial_{\mu,n}S$.

Or simply written as

$$P_\epsilon \circ U_\epsilon \circ P_{\frac{1}{2}\epsilon}\,(P_0, U_0) \tag{34}$$

The pseudo code can be written as

Algorithm 2 leap-frog integration

```
f = CalculateForce(actions, U)
p = p + 0.5 × ϵf
for i = 1 to n do
    U = exp(ϵp)U
    f = CalculateForce(actions, U)
    if i = n then
        p = p + 0.5 × ϵf        ▷ We still need to update p for the Metropolis step.
    else
        p = p + ϵf
    end if
end for
```

2.1.6 A summary of HMC with pseudofermions

Now, every part is ready. We summary the HMC following the Sec.8.2.3 in Ref. [2]. The HMC with fermions can be divided into 6 steps.

1. Generate a complex Bosonic field with $\chi \sim \exp(-\chi^\dagger \chi)$, and $\phi = \hat{D}\chi$.
2. Generate a momentum field P by $\exp(-tr(P^2))$.
3. Calculate $E = tr(P^2) + S_G(U) + S_{pf}(U, \phi)$.
4. Use U_0 to calculate F, evaluate P and U using integrator. Here, ϕ is treated as a constant field.
5. Finally, use P', U' to calculate Calculate $E' = tr(P'^2) + S_G(U') + S_{pf}(U', \phi)$. Use a Metropolis to accept or reject the result (configurations) **Note, by Refs. [2] and [6] 'reject' means add a duplicated old configuration.**.
6. Iterate from 1 to 5, until the number of configurations generated is sufficient.

- More on Metropolis step:

If the hybrid Monte Carlo can be implemented exactly, then, when equilibrium is reached, H should be unchanged, so, in some implementation, the Metropolis step can be ignored to archive a better accept rate. The parameter `Metropolis` of parameter `Updator` can be set to 1 if Metropolis step is enabled and 0 otherwise.

2.2 Optimization of HMC

2.2.1 Omelyan integrator

The Omelyan integrator can be simply written as (c.f. Eq. (2.80) of Ref. [1])

$$P_{\lambda\epsilon} \circ U_{\frac{1}{2}\epsilon} \circ P_{(1-2\lambda)\epsilon} \circ U_{\frac{1}{2}\epsilon} \circ P_{\lambda\epsilon} \left(P_0, U_0\right) \tag{35}$$

with

$$\lambda = \frac{1}{2} - \frac{\left(2\sqrt{326} + 36\right)^{\frac{1}{3}}}{12} + \frac{1}{6\left(2\sqrt{326} + 36\right)^{\frac{1}{3}}} \approx 0.19318332750378364 \tag{36}$$

In practical, the λ is a tunable parameter, and usually, $2\lambda = 0.3 \sim 0.5$ [6]. The `Omelyan2Lambda` parameter of `Updator` is a input parameter to set 2λ, which if left blank is set to be 0.38636665500756728 by default.

Usually, for each sub-step, it is 2 times slower then leap-frog, and for one trajectory, it is 1.5 time faster [6], implying the number of sub-step needed is about 1/3 of leap-frog.

2.2.2 Omelyan force-gradient integrator

Start from the approximation

$$\log\left(\exp(\frac{\epsilon S}{6})\exp(\frac{\epsilon T}{2})\exp\left(\frac{2}{3}\epsilon S+\frac{\epsilon^3}{72}[S,[S,T]]\right)\exp(\frac{\epsilon T}{2})\exp(\frac{\epsilon S}{6})\right)=S+T+\mathcal{O}(\epsilon^4)+\mathcal{O}(\epsilon^6) \tag{37}$$

with [9]

$$\mathcal{O}(\epsilon^4)\sim 10^{-4}\epsilon^4 \tag{38}$$

The other 4 steps are the usual ones, except for $\exp\left(\frac{2}{3}\epsilon S+\frac{\epsilon^3}{72}[S,[S,T]]\right)$ which correspond to

$$\begin{aligned}\omega_i &\to \omega_i-\frac{2}{3}\tau\frac{\partial}{\partial\omega_i}S+\frac{1}{36}\tau^3\sum_j\left(\frac{\partial}{\partial\omega_j}S\right)\frac{\partial}{\partial\omega_j}\frac{\partial}{\partial\omega_i}S\\ p_i &=\sum_a\omega_i^aT^a\end{aligned} \tag{39}$$

Use the approximation [10]

$$\begin{aligned}&\frac{2}{3}\tau\frac{\partial}{\partial\omega_i}S-\frac{1}{36}\tau^3\sum_j\left(\frac{\partial}{\partial\omega_j}S\right)\frac{\partial}{\partial\omega_j}\frac{\partial}{\partial\omega_i}S\\ &=\frac{2}{3}\tau\exp\left(-\frac{1}{24}\tau^2\sum_j\left(\frac{\partial}{\partial\omega_j}S\right)\frac{\partial}{\partial\omega_j}\right)\frac{\partial}{\partial\omega_i}S+\mathcal{O}(\tau^5)\end{aligned} \tag{40}$$

Let U' be a function of U, solving

$$\begin{aligned}&\frac{2}{3}\tau\exp\left(-\frac{1}{24}\tau^2\sum_j\left(\frac{\partial}{\partial\omega_j}S\right)\frac{\partial}{\partial\omega_j}\right)\frac{\partial}{\partial\omega_i}S(U)=\frac{2}{3}\tau\frac{\partial}{\partial\omega_i}S(U')\\ &\exp\left(-\frac{1}{24}\tau^2\sum_j\left(\frac{\partial}{\partial\omega_j}S\right)\frac{\partial}{\partial\omega_j}\right)\frac{\partial}{\partial\omega_i}U\frac{\partial S(U)}{\partial U}=\frac{\partial}{\partial\omega_i}U'\frac{\partial S(U')}{\partial U'}\\ &\frac{\partial}{\partial\omega_i}\left(\exp\left(-\frac{1}{24}\tau^2\sum_j\left(\frac{\partial}{\partial\omega_j}S\right)\frac{\partial}{\partial\omega_j}\right)U\right)=\frac{\partial}{\partial\omega_i}U'\\ &\exp\left(-\frac{1}{24}\tau^2\sum_j\left(\frac{\partial}{\partial\omega_j}S\right)\frac{\partial}{\partial\omega_j}\right)U=U',\\ &U'=\exp\left(-\frac{1}{24}\tau^2\sum_j\left(\frac{\partial}{\partial\omega_j}S\right)\frac{\partial}{\partial\omega_j}\right)U=\exp\left(-\frac{1}{24}\tau^2\sum_j\left(\frac{\partial}{\partial\omega_j}S\right)T_i\right)U\end{aligned} \tag{41}$$

This approximation can be divided into 3 steps:

1. Calculate $U' = \exp\left(-\frac{1}{24}\tau^2 \sum_j \left(\frac{\partial}{\partial\omega_j} S\right) T_i\right) U$.
2. Use $S[U']$ and $\frac{2}{3}\tau$ to update P.
3. Restore U.

It is easy to implement, we do not need to calculate second derivative, and $\sum_j \left(\frac{\partial}{\partial\omega_j} S\right) T_i$ is already implemented, it is nothing but the **force**.

It is almost as accurate as force-gradient, see the compare in Ref. [11]

2.2.3 Multi-rate integrator (nested integrator)

Following Ref. [11]

Assuming the action is $S = S_F + S_G$ with $S_G \gg S_F$, one can evaluate S_G more often than S_F. In the case of lattice QCD, often, the S_G is the cheap gauge force, and S_F is the expansive fermion force.

The nested scheme is different for leap-frog Omelyan and force-gradient integrator, but they are similar

- Nested leap-frog

$$\begin{aligned} \Delta(h) &= \exp(\frac{h}{2}S_F)\Delta_m(h)\exp(\frac{h}{2}S_F), \\ \Delta_m(h) &= \left(\exp(\frac{h}{2m}S_G)\exp(\frac{h}{m}T)\exp(\frac{h}{2m}S_G)\right)^m. \end{aligned} \tag{42}$$

- Nested Omelyan

$$\begin{aligned} \Delta(h) &= \exp(\lambda h S_F)\Delta_m(\frac{h}{2})\exp((1-2\lambda)hS_F)\Delta_m(\frac{h}{2})\exp(\epsilon h S_F), \\ \Delta_m(h) &= \left(\exp(\frac{\lambda h}{m}S_G)\exp(\frac{h}{2m}T)\exp(\frac{1-2\lambda}{m}hS_G)\exp(\frac{h}{2m}T)\exp(\frac{\lambda h}{m}S_G)\right)^m. \end{aligned} \tag{43}$$

Note, it is $\Delta_m(\frac{h}{2})$ in the first line.

- Nested force-gradient

$$\Delta(h) = \exp(\frac{h}{6}S_F)\Delta_m(\frac{h}{2})\exp(\frac{2}{3}hS_F + \frac{1}{72}h^3C_F)\Delta_m(\frac{h}{2})\exp(\frac{h}{6}S_F),$$
$$\Delta_m(h) = \left(\exp(\frac{h}{6m}S_G)\exp(\frac{h}{2m}T)\exp\left(\frac{2}{3}\frac{h}{m}S_G + \frac{1}{72}\left(\frac{h}{m}\right)^3 C_G\right)\exp(\frac{h}{2m}T)\exp(\frac{h}{6m}S_G)\right)^m. \tag{44}$$

Note about the integrator: when analyzing the error of the integrators, it is assumed e^T, e^{S_G} and e^{S_F} can be calculated accurately. It is almost true for e^{S_G}, and almost true for e^T as long as ϵ is not too large, but it is not true for e^{S_F}. Typically, using an optimized integrator, it needs more accurate criterion for solvers.

2.2.4 Cached solution

The pseudo fermion field is generate only once for a trajectory and is not changed. Also, the gauge field is changing slowly in one trajectory, this make the solutions for $\mathbf{x}_1 = D^{-1}\mathbf{b}$ or $\mathbf{x}_2 = \left(DD^\dagger\right)^{-1}\mathbf{b}$, where D depends on U and $\mathbf{b}$ is the pseudo fermion field, only change slowly.

So, once $\mathbf{x}_{1,2}$ is obtained, in the same trajectory, $\mathbf{x}_{1,2}$ can be set as the initial trail solution for the solver.

3 Sparse linear algebra solver

Given a matrix A and a vector $\mathbf{b}$. The solver works out the solution

$$\mathbf{b} = A\mathbf{x}, \quad \mathbf{x} = A^{-1}\mathbf{b}. \tag{45}$$

3.1 Krylov subspace

In short, the Krylov subspace methods assumes

$$\mathbf{x} \approx \sum_{l=0}^{k-1} C_l A^l \mathbf{b} \in K_k = span\left\{\mathbf{b}, A\mathbf{b}, \ldots, A^{k-1}\mathbf{b}\right\}. \tag{46}$$

with finite k, where C_k are coefficients. The equation $0 = \mathbf{b} - A\mathbf{x}$ becomes

$$\|\mathbf{b} - \sum_{l=0}^{k-1} C_l A^{l+1}\mathbf{b}\| = 0 \tag{47}$$

This is a problem in $k+1$ dimension, where k is independent of the dimension of $\mathbf{b}$, and usually significantly smaller than the dimension of $\mathbf{b}$. The Eq. (47) can be understand that, if $\mathbf{x}_k \approx A^{-1}\mathbf{b}$ is approximation of the solution in k dimension, in the $k+1$ dimension

$$(\mathbf{b} - A\mathbf{x})_{k+1} \perp K_k \tag{48}$$

That is, if we have a multi-dimension vector v_n, and its projection in 3-dimension ($D = k+1$) is a vector $\mathbf{v}_3$, if we want to find a plane ($D = k$) such that the projection of v_n in the plane is minimized, the plane is chosen to be the one orthogonal to $\mathbf{v}_3$.

3.2 GMRES

This section we follow Refs. [12] and [13].

Assume a set of basis has been found. For example, if the subspace is found by using modified Gram-Schmidt as

Algorithm 3 Arnoldi with modified Gram-Schmidt

$\mathbf{v}^{(0)} = \mathbf{x}_0/\|\mathbf{x}_0\|$
for $i = 0$ to $k-1$ **do**
  $\mathbf{w} = A\mathbf{v}^{(i)}$
  for $j = 0$ to i **do**
    $c = \mathbf{v}^{(j)^*} \cdot \mathbf{w}$
    $\mathbf{w} -= c\mathbf{v}^{(j)}$
    $h[j,i] = c$
  end for
  $h[i+1,i] = \|\mathbf{w}\|$
  $\mathbf{v}^{(i+1)} = \mathbf{w}/\|\mathbf{w}\|$
end for

Note that $(\mathbf{w} - (\mathbf{v}_i^* \cdot \mathbf{w})\,\mathbf{v}_i)^* \cdot \mathbf{v}_i = 0$, and $\mathbf{x}_0$ is a trail solution, which can be set to be $\mathbf{b}$ at first. Now we obtain $k+1$ unitary orthogonal vectors, such that

$$\mathbf{v}_i^* \cdot \mathbf{v}_j = \delta_{ij}, \quad A\mathbf{v}_{i-1} = \sum_{j=0}^{i} h[j,i-1]\mathbf{v}_j, \tag{49}$$

That is

$$\begin{pmatrix} Av_0 \\ Av_1 \\ Av_2 \\ \dots \\ Av_{k-1} \end{pmatrix} = (v_0, v_1, \dots, v_{k-1}, v_k) \begin{pmatrix} \text{h[0,0]} & \text{h[0,1]} & \dots & \text{h[0,k-2]} & \text{h[0,k-1]} \\ \text{h[1,0]} & \text{h[1,1]} & \dots & \text{h[1,k-2]} & \text{h[1,k-1]} \\ 0 & \text{h[2,1]} & \dots & \text{h[2,k-2]} & \text{h[2,k-1]} \\ 0 & 0 & \dots & \dots & \dots \\ \dots & \dots & \dots & \text{h[k-1,k-2]} & \text{h[k-1,k-1]} \\ 0 & 0 & \dots & 0 & \text{h[k,k-1]} \end{pmatrix} \tag{50}$$

which can be written as

$$(Av)_k = v_{k+1}H \tag{51}$$

The solution can be written as

$$\mathbf{x} = \mathbf{x}_0 + \sum_{i=0}^{k-1} y_i\mathbf{v}_i = \mathbf{x}_0 + \mathbf{y} = \mathbf{x}_0 + v_k y, \tag{52}$$

Using $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$, to minimize $\|\mathbf{b} - A\mathbf{x}\|$ is to minimize $\|\mathbf{r}_0 - A\mathbf{y}\|$. We always choose $\mathbf{v}_0 = \mathbf{r}_0/\|\mathbf{r}_0\|$, denote $\beta = \|\mathbf{r}_0\|$, it is to minimize

$$\text{argmin}\|\beta\mathbf{e}_0 - Hy\|. \tag{53}$$

Or, to solve an equation in k dimension

$$\beta \mathbf{e}_0 - Hy = 0, \;\; y = H^{-1}\beta \mathbf{e}_0 = H^{-1} g \tag{54}$$

Now, we need to solve H^{-1}, we can do this by applying rotation matrix, defining

$$J_0 = \begin{pmatrix} R & 0 \\ 0 & \mathbb{I}_{k-2} \end{pmatrix}_{D=k} = \begin{pmatrix} c_0^* & s_0^* & 0 & \dots & 0 \\ -s_0 & c_0 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ 0 & 0 & \dots & \dots & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}_{D=k} \tag{55}$$

Note that c_0^* and s_0^* is necessary to keep unitary. (s_0^* seems not necessary? we only need to keep the length of g unchanged) So that

$$0 = g - Hy \rightarrow 0 = J_0 g - J_0 H y \tag{56}$$

with (Note that the first 2 lines are changed entirely)

$$\begin{aligned} H' &= \begin{pmatrix} h'_{0,0} & h'_{0,1} & h'_{0,2} & \dots & h'_{0,k-1} \\ 0 & h'_{1,1} & h'_{1,2} & \dots & h'_{1,k-1} \\ 0 & h_{2,1} & h_{2,2} & \dots & h_{2,k-1} \\ 0 & 0 & \dots & \dots & 0 \\ 0 & 0 & 0 & 0 & h_{k,k-1} \end{pmatrix} \\ g' &= (c_0^* \beta, -s_0 \beta, 0, \dots) \\ c_0 &= \frac{h_{00}}{\sqrt{h_{00}^2 + h_{10}^2}}, \;\; s_0 = \frac{h_{10}}{\sqrt{h_{00}^2 + h_{10}^2}} \end{aligned} \tag{57}$$

where $\mathbb{I}_l$ is dimension l identity matrix. Similarly, after this, one can rotation matrices

$$J_1 = \begin{pmatrix} \mathbb{I}_1 & 0 & 0 \\ 0 & R & 0 \\ 0 & 0 & \mathbb{I}_{k-3} \end{pmatrix}_{D=k}, J_2 = \begin{pmatrix} \mathbb{I}_2 & 0 & 0 \\ 0 & R & 0 \\ 0 & 0 & \mathbb{I}_{k-4} \end{pmatrix}_{D=k}, \dots \tag{58}$$

To make H triangular.

The algorithm is

Algorithm 4 Rotate H

$g[0] = \beta$
for $i = 0$ to $k-1$ **do**
  $d = 1/\sqrt{|h[i,i]|^2 + |h[i+1,i]|^2}$
  $cs = h[i,i] \times d, sn = h[i+1,i] \times d$
  for $j = i$ to $k-1$ **do**
    $h_{ij} = h[i,j]$
    $h[i,j] = cs^* \times h_{ij} + sn^* \times h[i+1,j]$
    $h[i+1,j] = cs \times h[i+1,j] - sn \times h_{ij}$
  end for
  $minus_g = -g[i]$
  $g[i] = cs^* \times g[i]$
  $g[i+1] = sn \times minus_g$
end for

After the rotation, $g[k]$ is the residue. If it is small enough, the last step is to solve $y = H^{-1}g$, where H is a upper triangular matrix. It can be iterated as

$$y[k-1] = \frac{g[k-1]}{h[k-1,k-1]}. \; y[k-2] = \frac{1}{h[k-2,k-2]} \left(g[k-2] - h[k-2,k-1]y[k-1]\right), \ldots \tag{59}$$

The algorithm is

Algorithm 5 Solve Y

for $i = k-1$ to 0 **do**
  for $j = i+1$ to $k-1$ **do**
    $g[i] -= h[i,j] \times y[j]$
  end for
  $y[i] = g[i]/h[i,i]$
end for
  return $\mathbf{x}_0 + \sum_{i=0}^{k-1} y[i]\mathbf{v}^{(i)}$

Note that, the first step, the modified Gram-Schmidt step will produce more and more unitary normalized vectors, so the GMRES usually has a restart step. Let r denote the restart times, for example, the full algorithm with k is (GMRES(m) means GMRES with modified Gram-Schmidt, there is also GMRES with Household, etc)

Algorithm 6 GMRES(m)

$\mathbf{x}_0 = \mathbf{b}$ ▷ Use $\mathbf{b}$ as trail and start
for $i = 1$ to r **do**
  $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$
  $\beta = \|\mathbf{r}_0\|$
  $\mathbf{v}^{(0)} = \mathbf{r}_0/\beta$
  for $i = 0$ to $k-1$ **do**
    $\mathbf{w} = A\mathbf{v}^{(i)}$
    for $j = 0$ to i **do**
      $c = \mathbf{v}^{(j)*} \cdot \mathbf{w}$
      $\mathbf{w} -= c\mathbf{v}^{(j)}$
      $h[j, i] = c$
    end for
    $h[i+1, i] = \|\mathbf{w}\|$
    $\mathbf{v}^{(i+1)} = \mathbf{w}/\|\mathbf{w}\|$
  end for
  $RotateH(k)$
  $\mathbf{x} = SolveY(k)$
  if $|g[k]| < \epsilon$ **then**
   return $\mathbf{x}$ ▷ Succeed, with the solution
  end if
  $\mathbf{x}_0 = \mathbf{x}$ ▷ Use the last solution as trail and restart
end for
  return x ▷ Failed, with the last best solution

where $RotateH(k)$ and $\mathbf{x} = SolveY(k)$ is described in Algorithms. 4 and 5.

3.3 GCR

This section we follow Ref. [13].

The GCR solver is similar to GMRES in Sec. 3.2, but the orthogonal basis are obtained in a different way. If one have a set of orthogonal basis such that

$$A\mathbf{p}_i^* \cdot A\mathbf{p}_j = \delta_{ij}, \tag{60}$$

The solution $\mathbf{x}$ is the residue projected into this basis (Note, here we do NOT assume the basis are normalized)

$$\begin{aligned} \mathbf{r}_0 &= \mathbf{b} - A\mathbf{x}_0 \\ \mathbf{x} &= \mathbf{x}_0 + \sum_{\infty} \frac{\mathbf{r}_0^* \cdot A\mathbf{p}_i}{\|A\mathbf{p}_i\|}\mathbf{p}_i \end{aligned} \tag{61}$$

So, the iteration is

$$\mathbf{x} \approx \mathbf{x}_k = \mathbf{x}_0 + \sum_{i=0}^{k} \frac{\mathbf{r}_0^* \cdot A\mathbf{p}_i}{\|A\mathbf{p}_i\|}\mathbf{p}_i \tag{62}$$

which can be obtained order by order as

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \frac{(\mathbf{b} - A\mathbf{x}_{k-1})^* \cdot A\mathbf{p}_{k-1}}{\|A\mathbf{p}_{k-1}\|}\mathbf{p}_{k-1} \tag{63}$$

There are GCR, ORTHOMIN, ORTHODIR. Both GCR and ORTHOMIN have oscillation (tested with random Gaussian pseudo fermion field and random gauge field), when iterating, sometimes, $\|\mathbf{p}^i\| \gg \|\mathbf{p}^{i-1}$, and $\|\mathbf{p}^{i+1}\| \ll \|\mathbf{p}^i$ and $\|\mathbf{p}^{i+2}\| \gg \|\mathbf{p}^{i+1}$, so we use ORTHODIR. The algorithm is

Algorithm 7 incomplete GCR with restart

```
x = b                                              ▷ Use b as trail and start
for i = 0 to r do                                  ▷ restart r times
    r = b − Ax, p_0 = r
    for j = 0 to k − 1 do
        α = (Ap_j)* · r/‖Ap_j‖²
        x = x + αp_j
        r = r − αAp_j
        if ‖r‖ < ε then return x                   ▷ Success
        end if
        p_{j+1} = Ap_j
        for k = j − l + 1 to j do
            β = (Ap_k)* · A²p_j/‖Ap_k‖²
            p_{j+1} = p_{j+1} + βp_k
        end for
    end for
end for
    return x                                       ▷ Failed with the closest result
```

Note that, GCR is much slower than GMRES and BiCGStab. A strategy to improve the speed is to restart quickly.

3.4 GCRO-DR and GMRES-MDR

'A comparison with the methods seen in the previous chapter indicates that in many cases, GMRES will be faster if the problem is well conditioned, resulting in a moderate number of steps required to converge. If many steps (say, in the hundreds) are required, then BICGSTAB and TFQMR may perform better. If memory is not an issue, GMRES or DQGMRES, with a large number of directions, is often the most reliable choice. The issue then is one of trading ribustness for memory usage. In general, a sound strategy is to focus on finding a good preconditioner rather than the best accelerator'. [13].

From Fig. 1. 9 of Ref. [14], the low mode is the most critical problem, so CLGLib first implement low mode deflation preconditioner.

In the following, we follow Ref. [15].

3.4.1 Brief introduction to deflation preconditioner

In short, the preconditioner means, one solve

$$M^{-1}Ax = M^{-1}b, \tag{64}$$

or

$$\begin{cases} AM^{-1}u = b \\ x = M^{-1}u \end{cases} \tag{65}$$

instead of $Ax = b$. If M is chosen carefully, it is usually faster.

Now, considering $A \in \mathbb{C}^{n\times n}$ and a matrix $Z \in \mathbb{C}^{n\times k}$ such that $Z = (v_1, v_2, \ldots, v_k)$ and each row is a vector $v_i \in \mathbb{C}^n$ such that $v_i^\dagger v_j = \delta_{ij}$. So Z acts like a Unitary matrix $Z^\dagger Z = \mathbb{I}^{k\times k}$. Then we can use Z to project A on a subspace, as

$$T = Z^\dagger AZ, \quad Z^\dagger A = TZ^\dagger \tag{66}$$

so

$$Ax = b \Rightarrow \left(A - ZZ^\dagger A\right)x + ZT^{-1}Z^\dagger Ax = b - ZZ^\dagger b + ZT^{-1}Z^\dagger b \tag{67}$$

Note that $ZZ^\dagger \in \mathbb{C}^{n\times n}$ is not unitary matrix. $T \in \mathbb{C}^{k\times k}$ is a small matrix. And then, one can solve

$$ZT^{-1}Z^\dagger Ax = ZT^{-1}Z^\dagger b \Rightarrow x = ZT^{-1}Z^\dagger b \tag{68}$$

exactly, while solving $\left(A - ZZ^\dagger A\right)x = b - ZZ^\dagger b$ by iteration methods such as GMRES.

This is the so-called **subspace deflation**.

3.4.2 GCRO-DR

Start from Eq. (51). Assume after the first-step GMRES, we have the orthogonal-normal basis v_i which can be written as a matrix $V_m \in \mathbb{C}^{n\times m}, V_{m+1} \in \mathbb{C}^{n\times(m+1)}, H \in \mathbb{C}^{(m+1)\times m}$. On the other hand, will be introduced later, we have a set of deflation vectors, or a matrix $P_k \in \mathbb{C}^{m\times k}$, such that

$$\begin{aligned} &AV_mP_k = V_{m+1}HP_k \\ &\tilde{Y}_k \equiv V_mP_k \in \mathbb{C}^{n\times k} \end{aligned} \tag{69}$$

Then $\tilde{Y}_k$ is the deflation matrix.

Consider the matrix $HP_k = QR$, where QR is the QR factorization, with $Q \in \mathbb{C}^{(m+1)\times k}$ and $R \in \mathbb{C}^{k\times k}$. And define

$$C_k \equiv V_{m+1}Q \in \mathbb{C}^{n\times k}. \tag{70}$$

So, if R which is a small upper triangular matrix such that R^{-1} can be easily calculated, it is

$$\begin{aligned} &AV_mP_k = A\tilde{Y}_k = V_{m+1}HP_k = V_{m+1}QR = C_kR \\ &C_k = A\tilde{Y}_kR^{-1} = AU, \quad U \equiv \tilde{Y}_kR^{-1} \in \mathbb{C}^{n\times k} \end{aligned} \tag{71}$$

Finally, the problem in GMRES Eq. (51) is changed as

$$\begin{aligned}
&\tilde{U}_k = U_k D_k = U_k \begin{pmatrix} \frac{1}{\|\mathbf{u}_1\|} & 0 & 0 & 0 \\ 0 & \frac{1}{\|\mathbf{u}_2\|} & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \frac{1}{\|\mathbf{u}_k\|} \end{pmatrix} \in \mathbb{C}^{n\times k} \\
&V_m^{(1)} = (U_k, V_{m-k}) \in \mathbb{C}^{n\times m} \\
&V_{m+1}^{(2)} = (C_k, V_{m-k+1}) \in \mathbb{C}^{n\times(m+1)} \\
&H' = \begin{pmatrix} D_k & B_{m-k} \\ 0 & H_{m-k} \end{pmatrix} \in \mathbb{C}^{(m+1)\times m} \\
&AV_m^{(1)} = V_{m+1}^{(2)} H'
\end{aligned} \tag{72}$$

where V are orthogonal-normal basis obtained in GMRES, and $B_{m-k} = AV_{m-k}$. Note that $B_{m-k} \in \mathbb{C}^{(m-k)\times k}$ but $H_{m-k} \in \mathbb{C}^{(m-k+1)\times(m-k)}$.

From Eq. (72), we find

- The subspace is k dimension subspace of m dimension Krylov space.
- With H, V and P known, we are able to calculate $QR = HP$, $U = V_m P R^{-1}$, $C = V_{m+1}Q$.

3.4.3 The choice of deflation subspace

In the above, we have assumed $P_k \in \mathbb{C}^{m\times k}$ is already known. Now we concentrate on this part.

Let $A \in \mathbb{C}^{n\times n}, V \in \mathbb{C}^{n\times k}$, **If V is formed as orthogonal normal basis of subspace** S, then, if $(\lambda, w \in \mathbb{C}^m)$ is eigne-pair of $V^\dagger AV$, $(\lambda, u = V^\dagger w \in \mathbb{C}^n)$ is eigne-pair of A.

Therefor, **P_k is a matrix with k rows, and each row is a eigne-vector of H_m (H_m denoting the first m row of H_{m+1})**, then, VP_k is a matrix with k rows such that each row is a eigne-vector of A (approximately since $AV \approx VH \Rightarrow H \approx V^\dagger AV$).

The first GMRES cycle will generate H_m (denoting the first m row of H_{m+1}), and $H_m\omega = \theta\omega$ is solved. However, starting from the second cycle of GCRO-DR, it is not $AV_m = V_{m+1}H_{m+1}$ but $AV_m^{(1)} = V_{m+1}^{(2)}H'_{m+1}$, such that **$V_{m+1}^{(2)}$ are orthogonal basis but $V_m^{(1)}$ are not orthogonal basis! (Therefor $V^\dagger AV$ does not hold!)**. In this case, it is another eigne-problem which should be solved. This will be listed below without explain.

By Ref. [15], there are three strategies, Ritz value engine vector (REV), harmonic Ritz value engine vector (HEV) and singular value decomposition (SVD). Either it is $REV >$

$HEV > SVD$ or $SVD > HEV > REV$, so we only list REV and SVD here.

- REV

  The k small eigne value of m, such that m is

$$\begin{cases} H_m\omega = \theta\omega, \\ \begin{pmatrix} \tilde{U}_k^\dagger C_k & \tilde{U}_k^\dagger V_{m-k+1} \\ 0 & (I_{m-k}, 0) \end{pmatrix} H'_{m+1}\omega = \theta \begin{pmatrix} \tilde{U}_k^\dagger \tilde{U}_k & \tilde{U}_k^\dagger V_{m-k} \\ V_{m-k}^\dagger \tilde{U}_k & I_{m-k} \end{pmatrix} \omega, \end{cases} \tag{73}$$

- SVD

  The k small eigne value of m, such that m is

$$\begin{cases} H_m^\dagger H_m\omega = \theta\omega, \\ {H'_{m+1}}^\dagger H'_{m+1}\omega = \theta \begin{pmatrix} \tilde{U}_k^\dagger \tilde{U}_k & 0 \\ 0 & I_{m-k} \end{pmatrix} \omega, \end{cases} \tag{74}$$

Although H_m is usually a small matrix, we still need to known how to calculate the eigne-value and eigne-vectors.

3.4.4 Eigne solver

There are many strategies. For example using Lanczos to transform H to a tri-diagonal matrix, and then using divide-and-conquer algorithm

Algorithm 8 Lanczos algorithm

$\mathbf{v}_0$ is a random vector $\|\mathbf{v}_0\| = 1$
$\mathbf{w} = H\mathbf{v}_0$, $\alpha_0 = \mathbf{w}^\dagger\mathbf{v}_0$, $\mathbf{w} = \mathbf{w} - \alpha_0\mathbf{v}_0$
for $i = 1$ to $m - 1$ **do**
  $\beta_i = \|\mathbf{w}\|$, $\mathbf{v}_i = \mathbf{w}/\beta_i$
  $\mathbf{w} = H\mathbf{v}_i$, $\alpha_i = \mathbf{w}^\dagger\mathbf{v}_i$, $\mathbf{w} = \mathbf{w} - \alpha_i\mathbf{v}_i - \beta_i\mathbf{v}_{i-1}$
end for

3.4.5 Implementation of GCRO-DR

Now, we concentrate on the implementation of GCRO-DR. First of all, we need to know how to apply $\mathbf{x} - AB^\dagger\mathbf{v}$, where $A, B \in \mathbb{C}^{n\times k}$ and $\mathbf{v} \in \mathbb{C}^n$.

Algorithm 9 $\mathbf{x} = \mathbf{x} - AB^{\dagger}\mathbf{v}$

for $i = 0$ to $k-1$ **do**
  $\mathbf{x} = \mathbf{x} - \left(\mathbf{b}_k^{\dagger}\mathbf{v}\right)\mathbf{a}_k$
end for
  return x

The second thing is QR decompose of $\mathbb{C}^{n\times k}$ and $\mathbb{C}^{(m+1)\times k}$ matrix. For the $\mathbb{C}^{n\times k}$ matrix, the usually Arnoldi with modified Gram-Schmidt, i.e. Algorithm. 3 can be used.

Algorithm 10 modified Gram-Schmidt for QR factorization decompose of $A\tilde{Y}_k$

for $i = 0$ to $k-1$ **do**
  $y_i = Ay_i$
end for
$\mathbf{v}^{(0)} = y_0/\|\mathbf{y}_0\|$
for $i = 0$ to $k-2$ **do**
  $\mathbf{w} = \mathbf{y}_{i+1}$
  for $j = 0$ to $i+1$ **do**
    $c = \mathbf{v}^{(j)^*} \cdot \mathbf{w}$
    $\mathbf{w} -= c\mathbf{v}^{(j)}$
    $r[j,i] = c$
  end for
  $\mathbf{v}^{(i+1)} = \mathbf{w}/r[i+1,i+1]$
end for
  return $Q = (\mathbf{v}_0, \ldots, \mathbf{v}_{k-1})$, $R = r[i,j]$.

For the $\mathbb{C}^{(m+1)\times k}$ matrix, we use standard Gram-Schmidt for multi-thread since usually both k and m are small.

Algorithm 11 classical Gram-Schmidt for QR factorization decompose of HP_k

$\mathbf{q}[k] = hp[m, k]$
for $i = 0$ to $k - 1$ **do**
  for $j \leq i$ **do** ▷ pallial for each j
    $r[ji] = \mathbf{q}_j^\dagger \mathbf{q}_i$
  end for
  if $i < k - 1$ **then**
    for $j \leq i$ **do** ▷ pallial for each j
      $\mathbf{q}_{i+1} = \mathbf{q}_{i+1} - \frac{r[ji]}{r[jj]} \mathbf{q}_j$ ▷ Atomic add
    end for
  end if
end for
for $i = 0$ to $k - 1$ **do**
  $\mathbf{q}_i = \mathbf{q}_i / r[ii]$
end for
  return $Q = (\mathbf{q}_1, \ldots, \mathbf{q}_{k-1})$, $R = r[i, j]$.

Then, we need to calculate R^{-1} where R is a upper triangular matrix, this is nothing but a modification of Algorithm. 5.

Algorithm 12 $U_k = \tilde{Y}_k R^{-1}$

for $i = k - 1$ to 0 **do**
  for $j = i + 1$ to $k - 1$ **do**
    $\mathbf{y}[i] -= r[i, j]\mathbf{y}[j]$
  end for
  $\mathbf{y}[i] = \mathbf{y}[i] / r[i, i]$
end for
  return $\mathbf{u}[k] = \mathbf{y}[k]$.

4 Measurement

4.1 Plaquette Energy

For $SU(N)$, for square lattice, the gauge action can be written as

$$\begin{aligned}
S_G &= \beta\frac{1}{N}\sum_n\sum_{\mu>\nu}\left(N-\mathrm{tr}\left[U_\mu(n)U_\nu(n+a\mu)U_\mu^{-1}(n+a\nu)U_\nu^{-1}(n)\right]\right)\\
S_G &= \frac{1}{4}\beta\frac{1}{N}\sum_n\left((2(D-1))N-\mathrm{tr}\left[U_\mu(n)\Sigma_\mu(n)\right]\right),\\
\Sigma_\mu(n) &= \sum_{\mu\neq\nu}\left(U_\nu(n)U_\mu(n+a\nu)U_\nu^{-1}(n+a\mu)+U_\nu^{-1}(n-a\nu)U_\mu(n-a\nu)U_\nu(n-a\nu+a\mu)\right)
\end{aligned} \tag{75}$$

The plaquette energy is defined as

$$\begin{aligned}
\langle S\rangle &= \frac{1}{N\Lambda}\sum_n\sum_{\mu>\nu}\left(\mathrm{tr}\left[U_\mu(n)U_\nu(n+a\mu)U_\mu^{-1}(n+a\nu)U_\nu^{-1}(n)\right]\right).\\
&= \frac{1}{N\Lambda}\sum_{n,\mu}\mathrm{tr}\left[U_\mu(n)\Sigma_\mu(n)\right].
\end{aligned} \tag{76}$$

which is the average (average according to configurations) energy of plaquettes per plaquette (average according to plaquettes).

This is also $\langle W^{1\times1}\rangle$.

4.2 Meson Correlator

4.2.1 Meson Wave Function

We need at first construct an observable which is a bound state of two fermions and **has the same quantum number** as mesons. In short, we want to know

$$O(x)=\bar\psi(x)\Gamma\psi(x) \tag{77}$$

where Γ is a (product of) gamma matrix.

4.2.2 Meson Correlator

The correlator is defined as

$$C(x,y)=\langle\bar O(x)O(y)\rangle \tag{78}$$

where

$$\begin{aligned}\langle W\rangle &= \frac{1}{Z}\int \mathcal{D}[U,\bar{\psi},\psi]W\exp\left(-S\right)\\ Z &= \int \mathcal{D}[U,\bar{\psi},\psi]\exp\left(-S\right),\quad S=S_G+S_{pf}\end{aligned} \tag{79}$$

- iso-triplet

  Denote the variables as C_T and O_T.

  We need to calculate (green variables are constant)

$$\begin{aligned}C_T(n,m) &= \langle\bar{\psi}^{f_1}(n)\Gamma\psi^{f_2}(n)\bar{\psi}^{f_2}(m)\Gamma\psi^{f_1}(m)\rangle\\ &= \sum_{a,b,c_i}\Gamma_{a_1,b_1}\Gamma_{a_2,b_2}\langle\bar{\psi}^{f_1}_{a_1,c_1}(n)\psi^{f_2}_{b_1,c_1}(n)\bar{\psi}^{f_2}_{a_2,c_2}(m)\psi^{f_1}_{b_2,c_2}(m)\rangle\end{aligned} \tag{80}$$

Note that, they are all Grassman numbers (exchange three times will introduce a minus sign), and they can be averaged according to different fields, so

$$C_T(n,m) = -\sum_{a,b,c_i}\Gamma_{a_1,b_1}\Gamma_{a_2,b_2}\langle\psi^{f_2}_{b_1,c_1}(n)\bar{\psi}^{f_2}_{a_2,c_2}(m)\rangle_{f_1}\langle\psi^{f_1}_{b_2,c_2}(m)\bar{\psi}^{f_1}_{a_1,c_1}(n)\rangle_{f_2} \tag{81}$$

Using the Wick theorem for Grassman numbers (f is flavour index, c is color index, a,b are spinor index).

$$\begin{aligned}\langle\ldots\rangle &= \frac{1}{Z_f}\int\mathcal{D}\left[\psi\right]\ldots\exp\left(-\sum_{l,m}^{N}\bar{\psi}_l M_{lm}\psi_m\right).\\ \langle\psi_{i_1}\ldots\psi_{i_n}\bar{\psi}_{j_1}\ldots\bar{\psi}_{j_n}\rangle &= \sum_P \mathrm{sign}(P)\prod_n^N\left(M^{-1}\right)_{i_n,j_{P_n}}.\\ \langle\psi^f(n)_{a,c_1}\bar{\psi}^f_{b,c_2}(m)\rangle &= -D^{-1}_{f,a,b,c_1,c_2}(n,m).\end{aligned} \tag{82}$$

Then we can multiply gamma matrix back

$$C_T(n,m) = -\mathrm{tr}_{c,s}\left[\Gamma D^{-1}_{f_1}(n,m)\Gamma D^{-1}_{f_2}(m,n)\right] \tag{83}$$

The trace is for both color and spinor space.

- iso-singlet

  Denote the variables as C_S and O_S.

4.2.3 Sources

- Fourier transform

Usually, one need to know the observable in momentum space, which is

$$\tilde{C}(\mathbf{p}, n_t; \mathbf{0}, 0) \equiv \frac{1}{\sqrt{\Lambda_3}} \sum_{\mathbf{n}\in\Lambda_3} \exp(-ia\mathbf{n}\cdot\mathbf{p}) C(\mathbf{n}, n_t; \mathbf{0}, 0) \tag{84}$$

where Λ_3 denotes the spatial lattice.

For hadron spectroscopy,

$$\tilde{C}(\mathbf{p}, n_t; \mathbf{0}, 0) \propto \exp\left(-an_t E_0(\mathbf{p})\right) \times \left(1 + \mathcal{O}\left(e^{-an_t \Delta E}\right)\right) \tag{85}$$

where $E_0(\mathbf{p})$ is the ground state energy (dissipative relation?) and ΔE is the energy gap between ground state and the lowest excitation, and

$$E_0(\mathbf{p}) = \sqrt{m_H^2 + |\mathbf{p}|^2} \times \left(1 + \mathcal{O}\left(a|\mathbf{p}|\right)\right) \tag{86}$$

For zero momentum, we find m_H. That is why the lattice at t-dir is usually larger than the spatial directions.

From Eq. (84), we only need to calculate $C(n, 0)$ for all n. That is a **point source**.

Using $\{\gamma_\mu, \gamma_5\} = 0$ and $\gamma_5^2 = 1$, $\{\gamma_\mu, \gamma_5\} = 0$, so

$$\begin{aligned}
&\left(\Gamma D^{-1}(n,m)\Gamma D^{-1}(m,n)\right) = \left(\Gamma D^{-1}(n,m)\Gamma\gamma_5 \left(D^{-1}(n,m)\right)^\dagger \gamma_5\right) \\
&\mathrm{tr}_{c,s}\left[\Gamma D^{-1}(n,m)\Gamma\gamma_5 \left(D^{-1}(n,m)\right)^\dagger \gamma_5\right] = \mathrm{tr}_{c,s}\left[\gamma_5\Gamma D^{-1}(n,m)\Gamma\gamma_5 \left(D^{-1}(n,m)\right)^\dagger\right] \\
&= \pm\mathrm{tr}_{c,s}\left[\Gamma' D^{-1}(n,m)\Gamma' \left(D^{-1}(n,m)\right)^\dagger\right] \\
&= \pm\mathrm{tr}_{c,s}\left[\Gamma'^\dagger D^{-1}(n,m)\Gamma' \left(D^{-1}(n,m)\right)^\dagger\right]
\end{aligned} \tag{87}$$

where $\Gamma' = \Gamma\gamma_5$ and $\pm$ come from $\gamma_5\Gamma = \pm\Gamma\gamma_5$, and $\pm$ come from both $\gamma_5\Gamma = \pm\Gamma\gamma_5$ and $\Gamma^\dagger = \pm\Gamma^\dagger$. Note that it is in fact a **real** number because

$$\begin{aligned}
&\mathrm{tr}_{c,s}\left[\Gamma'^\dagger D^{-1}(n,m)\Gamma' \left(D^{-1}(n,m)\right)^\dagger\right] = \mathrm{tr}_{c,s}\left[D^{-1}(n,m)\Gamma' \left(D^{-1}(n,m)\right)^\dagger \Gamma'^\dagger\right] \\
&\left(\mathrm{tr}_{c,s}\left[\Gamma'^\dagger D^{-1}(n,m)\Gamma' \left(D^{-1}(n,m)\right)^\dagger\right]\right)^* = \mathrm{tr}_{c,s}\left[\left(D^{-1}(n,m)\Gamma' \left(D^{-1}(n,m)\right)^\dagger \Gamma'^\dagger\right)^\dagger\right] \\
&= \mathrm{tr}_{c,s}\left[\Gamma' D^{-1}(n,m)\Gamma'^\dagger \left(D^{-1}(n,m)\right)^\dagger\right] = \mathrm{tr}_{c,s}\left[\Gamma'^\dagger D^{-1}(n,m)\Gamma' \left(D^{-1}(n,m)\right)^\dagger\right]
\end{aligned} \tag{88}$$

With point source, we need only to calculate $D^{-1}(n,0)$, which is a $12\times 12 = 144$ elements matrix field on each site, with the matrix element

$$\begin{aligned} D^{-1}(n,m_0)_{c_1,c_2,s_1,s_2} &= \sum_{m,c_3,s_3} D^{-1}(n,m)_{c_1,c_3,s_1,s_3}\left(S(m_0,c_2,s_2;m,c_3,s_3)\right) \\ D^{-1}(n,m_0)_{:,c_2,:,s_2} &= D^{-1}\phi^S_{m_0,c_2,s_2} \end{aligned} \tag{89}$$

In the last line, $:,c_2,:,s_2$ denote one column of the 12×12 matrix, and $\phi^S_{m_0,c_2,s_2}$ is pseudo-fermion field with only one none-zero element (the **point source** at m_0, in our case, $m_0 = (\mathbf{0},0)$)

$$\phi^S_{m_0,c_2,s_2}(m)_{c,s} = \delta(m-m_0)\delta(c-c_2)\delta(s-s_2) \tag{90}$$

In matrix form it is

$$\begin{pmatrix} D^{-1}_{1,cs} \\ D^{-1}_{2,cs} \\ D^{-1}_{3,cs} \\ \dots \\ D^{-1}_{10,cs} \\ D^{-1}_{11,cs} \\ D^{-1}_{12,cs_2} \end{pmatrix} = \begin{pmatrix} D^{-1}_{1,1} & D^{-1}_{1,2} & \dots & D^{-1}_{1,cs} & \dots & D^{-1}_{1,11} & D^{-1}_{1,12} \\ D^{-1}_{2,1} & D^{-1}_{2,2} & \dots & D^{-1}_{2,cs} & \dots & D^{-1}_{2,11} & D^{-1}_{2,12} \\ D^{-1}_{3,1} & D^{-1}_{3,2} & \dots & D^{-1}_{3,cs} & \dots & D^{-1}_{3,11} & D^{-1}_{3,12} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ D^{-1}_{10,1} & D^{-1}_{0,2} & \dots & D^{-1}_{10,cs} & \dots & D^{-1}_{10,11} & D^{-1}_{10,12} \\ D^{-1}_{11,1} & D^{-1}_{11,2} & \dots & D^{-1}_{11,cs} & \dots & D^{-1}_{11,11} & D^{-1}_{11,12} \\ D^{-1}_{12,1} & D^{-1}_{12,2} & \dots & D^{-1}_{12,cs} & \dots & D^{-1}_{12,11} & D^{-1}_{12,12} \end{pmatrix} \begin{pmatrix} 0 \\ \dots \\ 0 \\ 1_{idx=cs} \\ 0 \\ \dots \\ 0 \end{pmatrix} \tag{91}$$

So, we need 12 point sources to fill the 12×12 matrix. Now, for each site, we can calculate the trace, and obtain a **real** field defined on sites

$$\Phi(n) = \mathrm{tr}_{c,s}\left(\Gamma D^{-1}\Gamma D^{-1}\right) \tag{92}$$

The final step is to sum the spatial lattice with the weight $e^{-ia\mathbf{n}\cdot\mathbf{p}}$ for each n_t, note that, the result should be calculated for each (assume periodic boundary condition for spatial directions)

$$\mathbf{p} \in \left\{ (p_1,p_2,p_3) | p_i = \frac{2\pi}{aN_i}k_i, k_i = -\frac{N_i}{2}-1,\dots\frac{N_i}{2} \right\} \tag{93}$$

where N_i is the length of the lattice at i direction.

Therefor, $\tilde{C}(\mathbf{p})$ is a complex field defined on spatial **reciprocal space**.

For spectroscopy, we need only the data for $\mathbf{p}=0$, because when $\Delta E \ll 1$

$$\tilde{C}(n_t) \equiv \tilde{C}(\mathbf{0},n_t;\mathbf{0},0) \propto \exp\left(-an_t m_H\right). \tag{94}$$

Note that, for $k_i = 0$, we need **even** number of length at spatial directions.

- Detail of implementation

We can write D^{-1} in the form of 4×4 matrices, with elements as 3×3 matrices, as

$$D^{-1} = \begin{pmatrix} U_{11} & U_{12} & U_{13} & U_{14} \\ U_{21} & U_{22} & U_{23} & U_{24} \\ U_{31} & U_{32} & U_{33} & U_{34} \\ U_{41} & U_{42} & U_{43} & U_{44} \end{pmatrix} \tag{95}$$

If our pseudo-fermion field is organized as

$$D^{-1}\phi^S \equiv \phi^{s\times 3+c}(n) = (d_0, d_1, d_2, d_3)\,, d_s = (v_0, v_1, v_2) \tag{96}$$

Using Eq. (91), one have

$$\begin{aligned} U_{ij} &= \begin{pmatrix} \phi^{j\times 3+0}(d_i, v_0) & \phi^{j\times 3+1}(d_i, v_0) & \phi^{j\times 3+2}(d_i, v_0) \\ \phi^{j\times 3+0}(d_i, v_1) & \phi^{j\times 3+1}(d_i, v_1) & \phi^{j\times 3+2}(d_i, v_1) \\ \phi^{j\times 3+0}(d_i, v_2) & \phi^{j\times 3+1}(d_i, v_2) & \phi^{j\times 3+2}(d_i, v_2) \end{pmatrix}, \\ U^T_{ij} &= \begin{pmatrix} \phi^{j\times 3+0}(d_i, v_0) & \phi^{j\times 3+0}(d_i, v_1) & \phi^{j\times 3+0}(d_i, v_2) \\ \phi^{j\times 3+1}(d_i, v_0) & \phi^{j\times 3+1}(d_i, v_1) & \phi^{j\times 3+1}(d_i, v_2) \\ \phi^{j\times 3+2}(d_i, v_0) & \phi^{j\times 3+2}(d_i, v_1) & \phi^{j\times 3+2}(d_i, v_2) \end{pmatrix} \end{aligned} \tag{97}$$

The Γ intersect between D^{-1} and $(D^{-1})^\dagger$ is a permutation of rows in spinor space, for example

$$\gamma_1 = \begin{pmatrix} 0 & 0 & 0 & c_1 \\ 0 & 0 & c_2 & 0 \\ 0 & c_3 & 0 & 0 \\ c_4 & 0 & 0 & 0 \end{pmatrix}, \quad \begin{matrix} p(1)=4, & p(2)=3, & p(3)=2, & p(4)=1 \\ c_1=-i, & c_2=-i, & c_3=i, & c_4=i \end{matrix} \tag{98}$$

where c_i are coefficients and $c_i \in \mathbb{Z}_4$ group. $p(i) = j$ denote that the none-zero of the i-th row is j-element, (also, the none-zero of the i-th column is j-element, because $\gamma_\mu^\dagger = \gamma_\mu$.).

So one have such that

$$\begin{aligned} \Gamma' D^{-1} &= \begin{pmatrix} c_1 U_{p(1)1} & c_1 U_{p(1)2} & c_1 U_{p(1)3} & c_1 U_{p(1)4} \\ c_2 U_{p(2)1} & c_2 U_{p(2)2} & c_2 U_{p(2)3} & c_2 U_{p(2)4} \\ c_3 U_{p(3)1} & c_3 U_{p(3)2} & c_3 U_{p(3)3} & c_3 U_{p(3)4} \\ c_4 U_{p(4)1} & c_4 U_{p(4)2} & c_4 U_{p(4)3} & c_4 U_{p(4)4} \end{pmatrix}, \\ \Gamma' D^{-1} \Gamma'^\dagger &= \begin{pmatrix} c_1 c_1^* U_{p(1)p(1)} & c_1 c_2^* U_{p(1)p(2)} & c_1 c_3^* U_{p(1)p(3)} & c_1 c_4^* U_{p(1)p(4)} \\ c_2 c_1^* U_{p(2)p(1)} & c_2 c_2^* U_{p(2)p(2)} & c_2 c_3^* U_{p(2)p(3)} & c_2 c_4^* U_{p(2)p(4)} \\ c_3 c_1^* U_{p(3)p(1)} & c_3 c_2^* U_{p(3)p(2)} & c_3 c_3^* U_{p(3)p(3)} & c_3 c_4^* U_{p(3)p(4)} \\ c_4 c_1^* U_{p(4)p(1)} & c_4 c_2^* U_{p(4)p(2)} & c_4 c_3^* U_{p(4)p(3)} & c_4 c_4^* U_{p(4)p(4)} \end{pmatrix} \end{aligned} \tag{99}$$

Finally we have (Note U is not a $SU(3)$ matrix)

$$\mathrm{tr}_{c,s}\left[\Gamma' D^{-1}\Gamma'^{\dagger}\left(D^{-1}\right)^{\dagger}\right] = \sum_{ij} c_i c_j^* \mathrm{tr}_c\left[U_{p(i)p(j)}U_{ij}^{\dagger}\right] = \sum_{ij} c_i c_j^* \mathrm{tr}_c\left[U_{ij}^{\dagger}U_{p(i)p(j)}\right] \tag{100}$$

This can be further simplified, note that the result should be a real number, so for $i \neq j$, if tr $\left[U_{p(i)p(j)}^{\dagger}U_{ij}\right]$ is present, so must be tr $\left[U_{ij}^{\dagger}U_{p(i)p(j)}\right]$ with the same sign. This is guaranteed by symmetric matrix, i.e. if $p(i) = a$, one must have $p(a) = i$, and also $c_i c_j^* = c_{p(i)}c_{p(j)}^*$ as shown below.

To prove $c_i c_j^* = c_{p(i)}c_{p(j)}^*$, we need to consider:

1. $\Gamma^{\dagger} = \pm\Gamma$ and $p(i) = i$. In this case, $c_i = c_{p(i)}$, and $c_i c_j^* = c_{p(i)}c_{p(j)}^*$ is straightforward.
2. $\Gamma^{\dagger} = \Gamma$ and $p(i) \neq i$. In this case, $c_i = c_{p(i)}^*$, so $c_i c_j^* = c_{p(i)}^* c_{p(j)}$. Note that, c_i are either all real or all imaginary, so $c_i c_j^* = c_{p(i)}^* c_{p(j)} = c_{p(i)}c_{p(j)}^*$.
3. $\Gamma^{\dagger} = -\Gamma$ and $p(i) \neq i$. In this case, $c_i = -c_{p(i)}^*$, so $c_i c_j^* = c_{p(i)}^* c_{p(j)} = c_{p(i)}c_{p(j)}^*$.

So, we have two cases, one for $p(1) = 1$, and one for $p(1) \neq 1$

$$\begin{aligned}
&\mathrm{tr}_{c,s}\left[\Gamma' D^{-1}\Gamma'^{\dagger}\left(D^{-1}\right)^{\dagger}\right] \\
&= \begin{cases} 2\sum_{i>1,j>i} c_i c_j^* \mathrm{Retr}_c\left[U_{ij}^{\dagger}U_{p(i)p(j)}\right] + \sum_{i=1,2,3,4}\mathrm{tr}_c\left[U_{ii}^{\dagger}U_{ii}\right] & p(i) = i \\ 2\sum_{i>1,j>i} c_i c_j^* \mathrm{Retr}_c\left[U_{ij}^{\dagger}U_{p(i)p(j)}\right] + 2\sum_{i=1,k}\mathrm{Retr}_c\left[U_{ii}^{\dagger}U_{p(i)p(i)}\right] & p(1) \neq 1,k \end{cases}
\end{aligned} \tag{101}$$

4.2.4 Gauge smearing

In HMC with fermions, the computer power is consumed mainly in solving the D^{-1}. The small eigenvalues of the D operator is the main reason to slow down the solver, which is the so called **low mode** or **exceptional configurations**.

Gauge smearing (gauge smoothing) is one of the method to ease the problem by replacing the original configuration with a gauge equivalent but easier configuration. There are several different smearing methods. In CLGLib, only two are implemented.

- APE

It use

$$U'_{\mu} = \mathcal{P}\left((1-\alpha)U_{\mu} + \frac{\alpha}{6}\Sigma_{\mu}\right) \tag{102}$$

where Σ_μ is the staple, (see Eq. (75)). In CLGLib, staples are cached. After smoothing, $\mathcal{P}$ is a projection project to result to $SU(3)$ and can be approximated as

Algorithm 13 $\mathcal{P}(U)$ approximately

$U = U/\sqrt{tr(U^\dagger U/3)}$
for $i = 0$ to r **do** ▷ iterate r times
  $x = U\left(\frac{3}{2} - \frac{1}{2}U^\dagger U\right)$
  $U = \left(1 - \frac{i}{3}\mathrm{Im}(\det(x))\right)x$
end for
  return U

Usually, iterate for 4 times, it can archive α accuracy.

- APE stout

In this approach, it construct a $SU(3)$ candidate directly by the staples. (Therefor, no need to project). Using

$$\Omega_\mu = \rho_\mu \Sigma_\mu U_\mu^\dagger,\ Q_\mu = \{\Omega_\mu\}_{TA},\ U'_\mu = \exp(Q_\mu)U_\mu \tag{103}$$

where ρ_μ is usually set to be $\rho_{1,2,3} = \rho, \rho_4 = 0$. Note that, there is no sum over μ in the above equation. Also, note that, exp is not accurate unless ρ is small enough, however, one can iterate the smearing for a few sub-steps.

4.3 Extend sources and gauge fixing

4.3.1 Extend sources

4.3.2 Gauge fixing

5 Programming

5.1 cuda

5.1.1 blocks and threads

5.1.2 device member function

According to https://stackoverflow.com/questions/53781421/cuda-the-member-field-with-device-ptr-and-device-member-function-to-visit-it-i

To call device member function, the content of the class should be on device.

- First, new a instance of the class.
- Then, create a device memory using cudaMalloc.
- Copy the content to the device memory

In other words, it will work as

```
__global__ void _kInitialArray(int* thearray)
{
    int iX = threadIdx.x + blockDim.x * blockIdx.x;
    int iY = threadIdx.y + blockDim.y * blockIdx.y;
    int iZ = threadIdx.z + blockDim.z * blockIdx.z;
    thearray[iX * 16 + iY * 4 + iZ] = iX * 16 + iY * 4 + iZ;
}

extern "C" {
    void _cInitialArray(int* thearray)
    {
        dim3 block(1, 1, 1);
        dim3 th(4, 4, 4);

        _kInitialArray << <block, th >> > (thearray);
        checkCudaErrors(cudaGetLastError());
    }
}

class B
{
public:
    B()
    {
```

```
25          checkCudaErrors(cudaMalloc((void**)&m_pDevicePtr, sizeof(int) * 64));
26          _cInitialArray(m_pDevicePtr);
27      }
28      ~B()
29      {
30          cudaFree(m_pDevicePtr);
31      }
32      __device__ int GetNumber(int index)
33      {
34          m_pDevicePtr[index] = m_pDevicePtr[index] + 1;
35          return m_pDevicePtr[index];
36      }
37      int* m_pDevicePtr;
38  };
39
40  __global__ void _kAddArray(int* thearray1, B* pB)
41  {
42      int iX = threadIdx.x + blockDim.x * blockIdx.x;
43      int iY = threadIdx.y + blockDim.y * blockIdx.y;
44      int iZ = threadIdx.z + blockDim.z * blockIdx.z;
45      thearray1[iX * 16 + iY * 4 + iZ] = thearray1[iX * 16 + iY * 4 + iZ] + pB->GetNumber(iX * 16 +
            iY * 4 + iZ);
46  }
47
48  extern "C" {
49      void _cAddArray(int* thearray1, B* pB)
50      {
51          dim3 block(1, 1, 1);
52          dim3 th(4, 4, 4);
53          _kAddArray << <block, th >> > (thearray1, pB);
54          checkCudaErrors(cudaGetLastError());
55      }
56  }
57
58  class A
59  {
60  public:
61      A()
62      {
63          checkCudaErrors(cudaMalloc((void**)&m_pDevicePtr, sizeof(int) * 64));
64          _cInitialArray(m_pDevicePtr);
65      }
66      ~A()
67      {
68          checkCudaErrors(cudaFree(m_pDevicePtr));
69      }
70      void Add(B* toAdd/*this should be a device ptr(new on device function or created by cudaMalloc)
```

```
            */)
71      {
72          _cAddArray(m_pDevicePtr, toAdd);
73      }
74      int* m_pDevicePtr;
75  };
76
77
78
79  int main(int argc, char * argv[])
80  {
81      B* pB = new B();
82      A* pA = new A();
83      B* pDeviceB;
84      checkCudaErrors(cudaMalloc((void**)&pDeviceB, sizeof(B)));
85      checkCudaErrors(cudaMemcpy(pDeviceB, pB, sizeof(B), cudaMemcpyHostToDevice));
86      pA->Add(pDeviceB);
87      int* res = (int*)malloc(sizeof(int) * 64);
88      checkCudaErrors(cudaMemcpy(res, pA->m_pDevicePtr, sizeof(int) * 64, cudaMemcpyDeviceToHost));
89      printf("----------␣A=");
90      for (int i = 0; i < 8; ++i)
91      {
92          printf("\n");
93          for (int j = 0; j < 8; ++j)
94              printf("res␣%d=%d␣␣", i * 8 + j, res[i * 8 + j]);
95      }
96      printf("\n");
97      //NOTE: We are getting data from pB, not pDeviceB, this is OK, ONLY because m_pDevicePtr is a
            pointer
98      checkCudaErrors(cudaMemcpy(res, pB->m_pDevicePtr, sizeof(int) * 64, cudaMemcpyDeviceToHost));
99      printf("----------␣B=");
100     for (int i = 0; i < 8; ++i)
101     {
102         printf("\n");
103         for (int j = 0; j < 8; ++j)
104             printf("res␣%d=%d␣␣", i * 8 + j, res[i * 8 + j]);
105     }
106     printf("\n");
107     delete pA;
108     delete pB;
109     return 0;
110 }
```

Note: this is a copy of the original instance! It is ONLY OK to change the content of $pDevicePtr->m\_pOtherPtr$, NOT $pDevicePtr->somevalue$

5.1.3 device virtual member function

According to https://stackoverflow.com/questions/26812913/how-to-implement-device-side-cuda-virtual-functions

To call a device virtual member function, unlike Sec. 5.1.2, the pointer to the virtual function table should also be on device,

- First, cudaMalloc a sizeof(void*), for the device pointer.
- Then, use a kernel function to new the instance on device, and assign it to the device pointer created by cudaMalloc.
- One can copy the pointer, by using cudaMemcpy(void**, void**, sizeof(void*), device-todevice).
- When copy it to elsewhere, one need to copy it back to host, then copy it again to device. The example shows how to copy it to constant.

in other words, it will work as

```
1
2  class CA
3  {
4  public:
5      __device__ CA() { ; }
6      __device__ ~CA() { ; }
7      __device__ virtual void CallMe() { printf("This␣is␣A\n"); }
8  };
9
10 class CB : public CA
11 {
12 public:
13     __device__ CB() : CA() { ; }
14     __device__ ~CB() { ; }
15     __device__ virtual void CallMe() { printf("This␣is␣B\n"); }
16 };
17
18 __global__ void _kernelCreateInstance(CA** pptr)
19 {
20     (*pptr) = new CB();
21 }
22
23 __global__ void _kernelDeleteInstance(CA** pptr)
24 {
```

```
25      delete (*pptr);
26  }
27
28  extern "C" {
29      void _kCreateInstance(CA** pptr)
30      {
31          _kernelCreateInstance << <1, 1 >> >(pptr);
32      }
33
34      void _kDeleteInstance(CA** pptr)
35      {
36          _kernelDeleteInstance << <1, 1 >> >(pptr);
37      }
38  }
39
40  __constant__ CA* m_pA;
41
42  __global__ void _kernelCallConstantFunction()
43  {
44      m_pA->CallMe();
45  }
46
47
48  extern "C" {
49      void _cKernelCallConstantFunction()
50      {
51          _kernelCallConstantFunction << <1, 1 >> > ();
52      }
53  }
54
55  int main()
56  {
57      CA** pptr;
58      cudaMalloc((void**)&pptr, sizeof(CA*));
59      _kCreateInstance(pptr);
60
61      //I can NOT use a kernel to set m_pA = (*pptr), because it is constant.
62      //I can NOT use cudaMemcpyToSymbol(m_pA, (*pptr)), because * operator on host is incorrect when
            pptr is a device ptr.
63      //I can NOT use cudaMemcpyToSymbol(m_pA, (*pptr)) in kernel, because cudaMemcpyToSymbol is a
            __host__ function
64      //I have to at first copy it back to host, then copy it back back again to constant
65      CA* pptrHost[1];
66      cudaMemcpy(pptrHost, pptr, sizeof(CA**), cudaMemcpyDeviceToHost);
67      cudaMemcpyToSymbol(m_pA, pptrHost, sizeof(CA*));
68      _cKernelCallConstantFunction();
69
```

```
70      _kDeleteInstance(pptr);
71      cudaFree(pptr);
72      return 0;
73  }
```

6 Testing

6.1 random number

7 Applications

7.1 Sample Producer

In HMC, the most time-consuming operation is $\left(DD^{\dagger}\right)^{-1}\phi$, which need to solve the Wilson-Dirac equation, a matrix equation $\mathbf{b} = A\mathbf{x}$, where $A = DD^{\dagger}$ is a matrix depending on the gauge field and acting on the pseudo-fermion field.

At the same time, applying machine learning algorithms to physics problems has gained more and more attentions. The machine learning algorithms has been applied to solve partial differential equations [16]. In Ref. [17], deep learning is applied to map between potential and energy bypassing the need to solve the Schrödinger equation, in other words, the Schrödinger equation is implicitly solved by the network. So, it is reasonable to ask whether the machine learning can also help to solve the Wilson-Dirac equation? For example, is it possible to train the network to output eigenvectors by inputting a gauge field, or even better output $\mathbf{x}$ by inputting a gauge field and a pseudo-fermion field $\mathbf{b}$?

索引

参考文献

[1] Michael Günther Francesco Knechtli and Michael Peardon. Lattice Quantum Chromodynamics Practical Essentials. 2017.

[2] C. Gattringer and C.B. Lang. Quantum Chromodynamics on the Lattice. 2010.

[3] Rajan Gupta. Introduction to Lattice QCD. 1998, arXiv:hep-lat/9807028.

[4] Alexander Altland and Ben Simons. Condensed Matter Field Theory 2nd edition. 2010.

[5] D. H. Weingarten and D. N. Petcher. Monte Carlo integration for lattice gauge theories with fermions. *Phys. Lett. B*, 99(4):333 – 338, 1981.

[6] Martin Lüscher. Computational Strategies in Lattice QCD. 2009, arXiv:1002.4232.

[7] S. Ueda et. al. Development of an object oriented lattice QCD code "Bridge++" on accelerators. *Journal of Physics: Conference Series*, 523:012046, 2014.

[8] Martin Lüscher. Schwarz-preconditioned HMC algorithm for two-flavor lattice QCD. *Computer Physics Communications*, 165:199–220, 2005.

[9] P. J. Silva A. D. Kennedy, M. A. Clark. Force Gradient Integrators. *PoS LAT*, 2009:021, 2009, arXiv:0910.2950.

[10] Robert D. Mawhinney Hantao Yin. Improving DWF Simulations: the Force Gradient Integrator and the Möbius Accelerated DWF Solver. *PoS LAT*, 2011:051, 2011, arXiv:1111.5059.

[11] Dmitry Shcherbakov et. al. Adapted nested force-gradient integrators: the Schwinger model case. 2015, arXiv:1512.03812.

[12] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods. 1994, Available at: http://www.netlib.org/templates/Templates.html.

[13] Yousef Saad. Iterative methods for sparse linear systems. 2003.

[14] Martin Lüscher. Computational Strategies in Lattice QCD. arXiv:arXiv:1002.4232.

[15] Hussam Al Daas et. al. Recycling Krylov subspaces and reducing deflation subspaces for solving sequence of linear systems. *RR-9206, Inria Paris*, 2018, Available at: https://hal.inria.fr/hal-01886546.

[16] Weinan E Jiequn Han, Arnulf Jentzen. Solving high-dimensional partial differential equations using deep learning. *PNAS*, 115(34):8505–8510, 2018.

[17] Isaac Tamblyn Kyle Mills, Michael Spanner. Deep learning and the Schrödinger equation. *Phys. Rev. A*, 96:042113, 2017, arXiv:1702.01361.