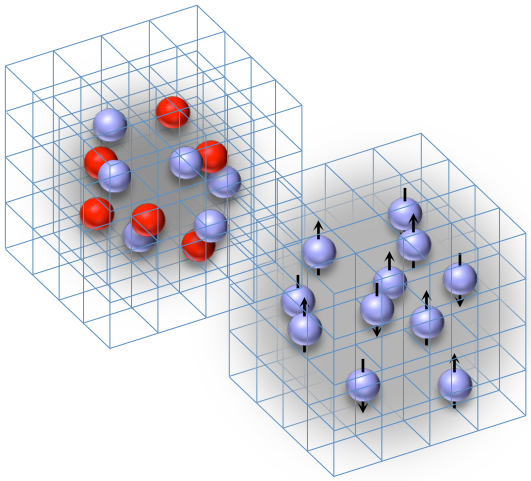
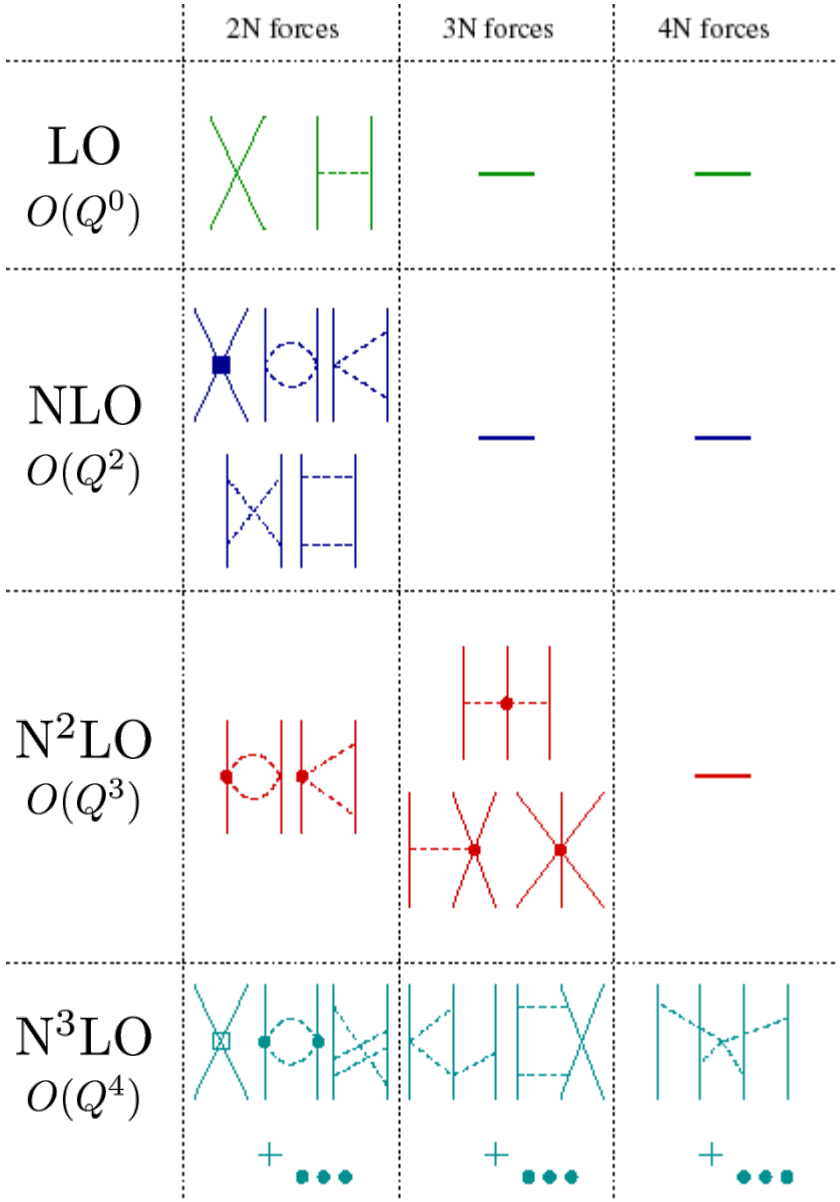
Lattice Methods for Nuclear Physics

Lecture 5: Chiral Effective Field Theory on the Lattice II

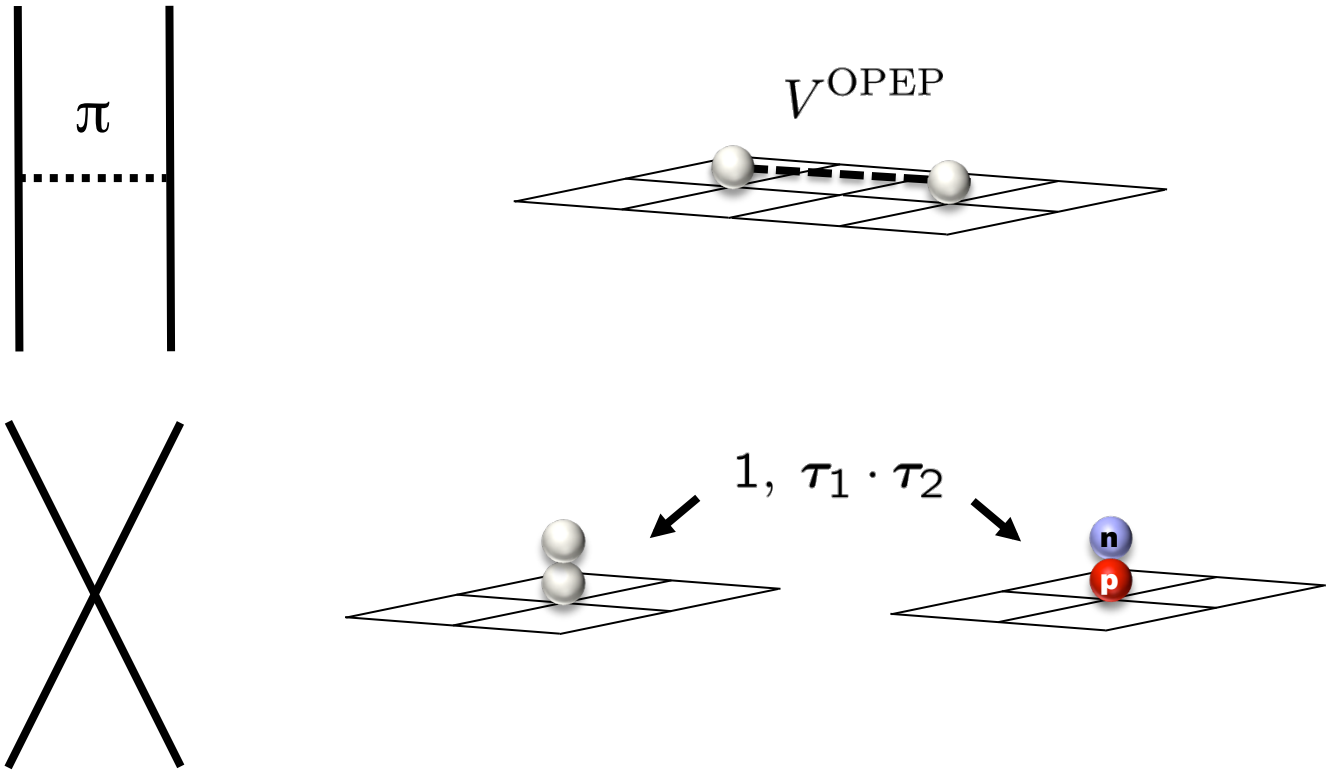
Dean Lee (NC State)
Nuclear Lattice EFT Collaboration



TALENT School
on Nuclear Quantum Monte Carlo Methods
North Carolina State University
July 27, 2016


2N forces
3N forces
4N forces
LO
$O(Q^0)$
NLO
$O(Q^2)$
N^{2}LO
$O(Q^3)$
N^{3}LO
$O(Q^4)$
+ ...
+ ...
+ ...

Leading order interactions



We first consider the leading order chiral EFT interaction on the lattice in the Grassmann path integral formalism

$$\mathcal{Z} = \int DcDc^* \exp\left[-S\left(c^*, c\right)\right]$$

$$S(c^*, c) = S_{\text{free}}(c^*, c) + S_{\text{int}}(c^*, c)$$

$$S_{\text{free}}(c^*, c) = \sum_{\vec{n}, n_t, i} \boxed{c_i^*(\vec{n}, n_t)\left[c_i(\vec{n}, n_t + 1) - c_i(\vec{n}, n_t)\right]} \rightarrow c_i^* \frac{\partial c_i}{\partial t}$$

$$-\frac{\alpha_t}{2m} \sum_{\vec{n}, n_t, i} \sum_{l=1,2,3} \boxed{c_i^*(\vec{n}, n_t)\left[c_i(\vec{n} + \hat{l}, n_t) - 2c_i(\vec{n}, n_t) + c_i(\vec{n} - \hat{l}, n_t)\right]}$$

$$\rightarrow c_i^* \frac{\partial^2 c_i}{\partial x_l^2}$$

It is convenient to view c without indices as a column vector and c^* without indices as a row vector

$$c^* = \left[c_{\uparrow,p}^* c_{\downarrow,p}^* c_{\uparrow,n}^* c_{\downarrow,n}^*\right] \qquad c = \begin{bmatrix} c_{\uparrow,p} \\ c_{\downarrow,p} \\ c_{\uparrow,n} \\ c_{\downarrow,n} \end{bmatrix}$$

The first interaction we consider is the short-range interaction between nucleons which is independent of spin and isospin

$$S^C_{\text{int}}(c^*, c) = \alpha_t \frac{C}{2} \sum_{\vec{n}, n_t} \left[c^*(\vec{n}, n_t) c(\vec{n}, n_t)\right]^2$$

Using the auxiliary field s, we can write this interaction as

$$\exp\left[-S^C_{\text{int}}(c^*, c)\right] = \int Ds \ \exp\left[-S_{ss}(s) - S_s(c^*, c, s)\right]$$

where

$$S_{ss}(s) = \frac{1}{2} \sum_{\vec{n}, n_t} s^2(\vec{n}, n_t)$$

$$S_s(c^*, c, s) = \sqrt{-C\alpha_t} \sum_{\vec{n}, n_t} s(\vec{n}, n_t) c^*(\vec{n}, n_t) c(\vec{n}, n_t)$$

Next we have the short-range interaction dependent on isospin

$$S_{\text{int}}^{C'}(c^*, c) = \alpha_t \frac{C'}{2} \sum_{\vec{n}, n_t, I} \left[c^*(\vec{n}, n_t)\tau_I c(\vec{n}, n_t)\right]^2$$

where we are using the Pauli matrices in isospin space

$$\tau_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}_{\text{isospin}} \qquad \tau_2 = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}_{\text{isospin}} \qquad \tau_3 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}_{\text{isospin}}$$

In terms of three auxiliary fields s_I, we can write the interaction as

$$\exp\left[-S_{\text{int}}^{C'}(c^*, c)\right] = \int \prod_I Ds_I \exp\left[-S_{s_I s_I}(s_I) - S_{s_I}(c^*, c, s_I)\right]$$

$$S_{s_I s_I}(s_I) = \frac{1}{2} \sum_{\vec{n}, n_t, I} s_I^2(\vec{n}, n_t)$$

$$S_{s_I}(c^*, c, s_I) = \sqrt{-C'\alpha_t} \sum_{\vec{n}, n_t, I} s_I(\vec{n}, n_t) c^*(\vec{n}, n_t)\tau_I c(\vec{n}, n_t)$$

The remaining interaction is the one pion exchange potential (OPEP). We will not include time derivatives in the free pion action, and hence the the pion is not treated as a dynamical field. Instead it resembles an auxiliary field that produces an exchange potential for the nucleons.

$$\exp\left[-S_{\rm int}^{\rm OPEP}(c^*,c)\right] = \int \prod_I D\pi_I \exp\left[-S_{\pi_I\pi_I}(\pi_I) - S_{\pi_I}(c^*,c,\pi_I)\right]$$

$$\begin{aligned} S_{\pi_I\pi_I}(\pi_I) &= \tfrac{1}{2}\alpha_t m_\pi^2 \textstyle\sum_{\vec{n},n_t,I} \pi_I^2(\vec{n},n_t) \\ &\quad -\tfrac{1}{2}\alpha_t \textstyle\sum_{\vec{n},n_t,I,\hat{l}} \pi_I(\vec{n},n_t)\left[\pi_I(\vec{n}+\hat{l},n_t) - 2\pi_I(\vec{n},n_t) + \pi_I(\vec{n}-\hat{l},n_t)\right] \end{aligned}$$

The pion coupling to the nucleon is

$$S_{\pi_I}(c^*, c, \pi_I) = \frac{g_A \alpha_t}{2 f_\pi} \sum_{\vec{n}, n_t, l, I} \Delta_k \pi_I(\vec{n}, n_t) c^*(\vec{n}, n_t) \sigma_k \tau_I c(\vec{n}, n_t)$$

where g_A is the axial charge, f_π is the pion decay constant, and we have used the Pauli spin matrices

$$\sigma_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}_{\text{spin}} \qquad \sigma_2 = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}_{\text{spin}} \qquad \sigma_3 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}_{\text{spin}}$$

And the gradient of the pion field is

$$\Delta_l \pi_I(\vec{n}, n_t) = \frac{1}{2} \left[\pi_I(\vec{n} + \hat{l}, n_t) - \pi_I(\vec{n} - \hat{l}, n_t) \right]$$

We can reexpress everything in terms of normal-ordered transfer matrix operators

$$\mathcal{Z} = \int Ds \prod_I (Ds_I D\pi_I)$$
$$\exp\left[-S_{ss}(s) - S_{s_I s_I}(s_I) - S_{\pi_I \pi_I}(\pi_I)\right] \mathrm{Tr}\left\{M^{(L_t-1)} \dots M^{(0)}\right\}$$

where

$$M^{(n_t)} =: \exp\left[-H^{(n_t)}(a^\dagger, a, s, s_I, \pi_I)\alpha_t\right]$$

$$H^{(n_t)}(a^\dagger, a, s, s_I, \pi_I)\alpha_t = H_{\mathrm{free}}\alpha_t + S_s^{(n_t)}(a^\dagger, a, s) + S_{s_I}^{(n_t)}(a^\dagger, a, s_I) + S_{\pi_I}^{(n_t)}(a^\dagger, a, \pi_I)$$

with

$$S_s^{(n_t)}(a^\dagger, a, s) = \sqrt{-C\alpha_t} \sum_{\vec{n}} s(\vec{n}, n_t) a^\dagger(\vec{n}) a(\vec{n})$$

$$S_{s_I}^{(n_t)}(a^\dagger, a, s_I) = \sqrt{-C'\alpha_t} \sum_{\vec{n}, I} s_I(\vec{n}, n_t) a^\dagger(\vec{n}) \tau_I a(\vec{n})$$

$$S_{\pi_I}^{(n_t)}(a^\dagger, a, \pi_I) = \frac{g_A \alpha_t}{2 f_\pi} \sum_{\vec{n}, k, I} \Delta_k \pi_I(\vec{n}, n_t) a^\dagger(\vec{n}) \sigma_k \tau_I a(\vec{n})$$

For the auxiliary-field projection Monte Carlo calculation we compute

$$Z(L_t) = \int Ds \prod_I (Ds_I D\pi_I)$$
$$= \exp\left[-S_{ss}(s) - S_{s_I s_I}(s_I) - S_{\pi_I \pi_I}(\pi_I)\right] Z(s, s_I, \pi_I, L_t)$$

where

$$Z(s, s_I, \pi_I, L_t) = \det \mathbf{Z}(s, s_I, \pi_I, L_t)$$

and the matrix of single nucleon amplitudes is

$$\mathbf{Z}_{i,j}(s, s_I, \pi_I, L_t) = \langle f_i | M^{(L_t-1)} \dots M^{(0)} | f_j \rangle$$

We store the set of vectors for each single-particle initial state at each time step

$$|v_j^{(n_t)}\rangle = M^{(n_t-1)} \cdots M^{(0)} |f_j\rangle$$

as well as the dual vectors at each time step propagating in the reverse temporal direction

$$\langle v_i^{(n_t)}| = \langle f_i| M^{(L_t-1)} \cdots M^{(n_t)}$$

These are useful in computing the update to an auxiliary field value at time step n_t, using the following relations:

$$Z(s, s_I, \pi_I, L_t) = \det \mathbf{Z}(s, s_I, \pi_I, L_t)$$

$$\mathbf{Z}_{i,j}(s, s_I, \pi_I, L_t) = \langle v_i^{(n_t+1)} | M^{(n_t)}(s, s_I, \pi_I) | v_j^{(n_t)} \rangle$$

It is convenient to redefine the normalization of $\boldsymbol{\pi}_I$

$$\pi'_I(\vec{n}, n_t) = \pi_I(\vec{n}, n_t)\sqrt{\alpha_t(m_\pi^2 + 6)}$$

So that

$$S_{\pi_I \pi_I}(\pi_I) = \tfrac{1}{2}\alpha_t m_\pi^2 \sum_{\vec{n},n_t,I} \pi_I^2(\vec{n}, n_t)$$
$$-\tfrac{1}{2}\alpha_t \sum_{\vec{n},n_t,I,\hat{l}} \pi_I(\vec{n}, n_t)\left[\pi_I(\vec{n}+\hat{l}, n_t) - 2\pi_I(\vec{n}, n_t) + \pi_I(\vec{n}-\hat{l}, n_t)\right]$$
$$= \tfrac{1}{2}\sum_{\vec{n},n_t,I} \pi_I'^2(\vec{n}, n_t) - \tfrac{\alpha_t}{2q_\pi}\sum_{\vec{n},n_t,I,\hat{l}} \pi'_I(\vec{n}, n_t)\left[\pi'_I(\vec{n}+\hat{l}, n_t) + \pi'_I(\vec{n}-\hat{l}, n_t)\right]$$

and

$$S_{\pi_I}^{(n_t)}(a^\dagger, a, \pi_I) = \tfrac{g_A \alpha_t}{2 f_\pi \sqrt{q_\pi}} \sum_{\vec{n},k,I} \Delta_k \pi'_I(\vec{n}, n_t) a^\dagger(\vec{n}) \sigma_k \tau_I a(\vec{n})$$

with

$$q_\pi = \alpha_t(m_\pi^2 + 6)$$

```fortran
SUBROUTINE getzvecs(s,sI,zvecs,zvecsinit,nt2,nt1, &
     pion,ztau2x2,num)

  IMPLICIT INTEGER(i-n)
  IMPLICIT DOUBLE PRECISION(a-h,o-y)
  IMPLICIT COMPLEX*16(z)
  INCLUDE "input.f90"

  DIMENSION s(0:L-1,0:L-1,0:L-1,0:Lt-1)
  DIMENSION sI(0:L-1,0:L-1,0:L-1,0:Lt-1,1:3)
  DIMENSION zvecs(0:L-1,0:L-1,0:L-1,0:Lt,0:1,0:1,0:num-1)
  DIMENSION zvecsinit(0:L-1,0:L-1,0:L-1,0:1,0:1,0:num-1)
  DIMENSION pion(0:L-1,0:L-1,0:L-1,0:Lt-1,1:3)
  DIMENSION zpi2x2(0:1,0:1)
  DIMENSION zsI2x2(0:1,0:1)
  DIMENSION zsSI2x2(0:1,0:1,0:1,0:1)
  DIMENSION ztau2x2(0:1,0:1,0:3)

  !   nt2 > nt1
```

⋮

```
qpi3 = atovera*(ampi3**2+6.D0)

DO ni = 0,1; DO ns = 0,1
   DO nz = 0,L-1; DO ny = 0,L-1; DO nx = 0,L-1
      DO npart = 0,num-1
         zvecs(nx,ny,nz,nt1,ns,ni,npart) = zvecsinit(nx,ny,nz,ns,ni,npart)
      END DO
   END DO; END DO; END DO
END DO; END DO

DO nt = nt1+1, nt2
   DO np = 0,num-1
      DO nz = 0,L-1; DO ny = 0,L-1; DO nx = 0,L-1
         DO ni = 0,1; DO ns = 0,1

            zvecs(nx,ny,nz,nt,ns,ni,np) = zvecs(nx,ny,nz,nt-1,ns,ni,np) &
                 * (1.D0-6.D0*h+CDSQRT(-c0*atovera*(1.D0,0.D0))*s(nx,ny,nz,nt-1))

            zvecs(nx,ny,nz,nt,ns,ni,np) = zvecs(nx,ny,nz,nt,ns,ni,np) &
                 + h*zvecs(MOD(nx+1,L),ny,nz,nt-1,ns,ni,np) &
                 + h*zvecs(MOD(nx-1+L,L),ny,nz,nt-1,ns,ni,np) &
                 + h*zvecs(nx,MOD(ny+1,L),nz,nt-1,ns,ni,np) &
                 + h*zvecs(nx,MOD(ny-1+L,L),nz,nt-1,ns,ni,np) &
                 + h*zvecs(nx,ny,MOD(nz+1,L),nt-1,ns,ni,np) &
                 + h*zvecs(nx,ny,MOD(nz-1+L,L),nt-1,ns,ni,np)

         END DO; END DO
      END DO; END DO; END DO
   END DO

            ⋮
```

```
DO nz = 0,L-1; DO ny = 0,L-1; DO nx = 0,L-1

   DO nii = 0,1; DO ni = 0,1
      DO nss = 0,1; DO ns = 0,1
         zsSI2x2(ns,nss,ni,nii) = (0.D0,0.D0)
      END DO; END DO
   END DO; END DO

   zsI2x2(0,0) = sI(nx,ny,nz,nt-1,3)
   zsI2x2(1,1) = -sI(nx,ny,nz,nt-1,3)
   zsI2x2(0,1) = sI(nx,ny,nz,nt-1,1) - (0.D0,1.D0)*sI(nx,ny,nz,nt-1,2)
   zsI2x2(1,0) = sI(nx,ny,nz,nt-1,1) + (0.D0,1.D0)*sI(nx,ny,nz,nt-1,2)

   DO nii = 0,1; DO ni = 0,1
      DO ns = 0,1
         zsSI2x2(ns,ns,ni,nii) = zsSI2x2(ns,ns,ni,nii) &
              + (0.D0,1.D0)*CDSQRT(cI*atovera*(1.D0,0.D0))*zsI2x2(ni,nii)
      END DO
   END DO; END DO
```

⋮

```
DO iso = 1,3

   pi1 = 0.D0 &
        + 1.D0/2.D0*pion(MOD(nx+1,L),ny,nz,nt-1,iso) &
        - 1.D0/2.D0*pion(MOD(nx-1+L,L),ny,nz,nt-1,iso)

   pi2 = 0.D0 &
        + 1.D0/2.D0*pion(nx,MOD(ny+1,L),nz,nt-1,iso) &
        - 1.D0/2.D0*pion(nx,MOD(ny-1+L,L),nz,nt-1,iso)

   pi3 = 0.D0 &
        + 1.D0/2.D0*pion(nx,ny,MOD(nz+1,L),nt-1,iso) &
        - 1.D0/2.D0*pion(nx,ny,MOD(nz-1+L,L),nt-1,iso)

   zpi2x2(0,0) = pi3
   zpi2x2(1,1) = -pi3
   zpi2x2(0,1) = pi1 - (0.D0,1.D0)*pi2
   zpi2x2(1,0) = pi1 + (0.D0,1.D0)*pi2

   DO nii = 0,1; DO ni = 0,1
      DO nss = 0,1; DO ns = 0,1
         zsSI2x2(ns,nss,ni,nii) = zsSI2x2(ns,nss,ni,nii) &
              - gA*atovera/(2.D0*fpi*DSQRT(qpi3))*zpi2x2(ns,nss) &
              * ztau2x2(ni,nii,iso)
      END DO; END DO
   END DO; END DO

END DO

      ⋮
```

```
        DO np = 0,num-1
           DO nii = 0,1; DO ni = 0,1
              DO nss = 0,1; DO ns = 0,1
                 zvecs(nx,ny,nz,nt,ns,ni,np) = zvecs(nx,ny,nz,nt,ns,ni,np) &
                      + zsSI2x2(ns,nss,ni,nii)*zvecs(nx,ny,nz,nt-1,nss,nii,np)
              END DO; END DO
           END DO; END DO
        END DO

     END DO; END DO; END DO

  END DO

END SUBROUTINE getzvecs
```

```fortran
SUBROUTINE getzdualvecs(s,sI,zdualvecs,zdualvecsinit, &
           nt2,nt1,pion,ztau2x2,num)

  IMPLICIT INTEGER(i-n)
  IMPLICIT DOUBLE PRECISION(a-h,o-y)
  IMPLICIT COMPLEX*16(z)
  INCLUDE "input.f90"

  DIMENSION s(0:L-1,0:L-1,0:L-1,0:Lt-1)
  DIMENSION sI(0:L-1,0:L-1,0:L-1,0:Lt-1,1:3)
  DIMENSION zdualvecs(0:L-1,0:L-1,0:L-1,0:Lt,0:1,0:1,0:num-1)
  DIMENSION zdualvecsinit(0:L-1,0:L-1,0:L-1,0:1,0:1,0:num-1)
  DIMENSION pion(0:L-1,0:L-1,0:L-1,0:Lt-1,1:3)
  DIMENSION zpi2x2(0:1,0:1)
  DIMENSION zsI2x2(0:1,0:1)
  DIMENSION zsSI2x2(0:1,0:1,0:1,0:1)
  DIMENSION ztau2x2(0:1,0:1,0:9)

  !   nt2 > nt1
```

⋮

```
qpi3 = atovera*(ampi3**2+6.D0)

DO ni = 0,1; DO ns = 0,1
   DO nz = 0,L-1; DO ny = 0,L-1; DO nx = 0,L-1
      DO npart = 0,num-1
         zdualvecs(nx,ny,nz,nt2,ns,ni,npart) = zdualvecsinit(nx,ny,nz,ns,ni,npart)
      END DO
   END DO; END DO; END DO
END DO; END DO

DO nt = nt2,nt1+1,-1

   DO np = 0,num-1
       DO nz = 0,L-1; DO ny = 0,L-1; DO nx = 0,L-1
          DO ni = 0,1; DO ns = 0,1

             zdualvecs(nx,ny,nz,nt-1,ns,ni,np) = zdualvecs(nx,ny,nz,nt,ns,ni,np) &
                  * (1.D0-6.D0*h+CDSQRT(-c0*atovera*(1.D0,0.D0))*s(nx,ny,nz,nt-1))

             zdualvecs(nx,ny,nz,nt-1,ns,ni,np) = zdualvecs(nx,ny,nz,nt-1,ns,ni,np) &
                  + h*zdualvecs(MOD(nx+1,L),ny,nz,nt,ns,ni,np) &
                  + h*zdualvecs(MOD(nx-1+L,L),ny,nz,nt,ns,ni,np) &
                  + h*zdualvecs(nx,MOD(ny+1,L),nz,nt,ns,ni,np) &
                  + h*zdualvecs(nx,MOD(ny-1+L,L),nz,nt,ns,ni,np) &
                  + h*zdualvecs(nx,ny,MOD(nz+1,L),nt,ns,ni,np) &
                  + h*zdualvecs(nx,ny,MOD(nz-1+L,L),nt,ns,ni,np)

          END DO; END DO
       END DO; END DO; END DO
    END DO
```

⋮

```
DO nz = 0,L-1; DO ny = 0,L-1; DO nx = 0,L-1

   DO nii = 0,1; DO ni = 0,1
      DO nss = 0,1; DO ns = 0,1
         zsSI2x2(ns,nss,ni,nii) = (0.D0,0.D0)
      END DO; END DO
   END DO; END DO

   zsI2x2(0,0) = sI(nx,ny,nz,nt-1,3)
   zsI2x2(1,1) = -sI(nx,ny,nz,nt-1,3)
   zsI2x2(0,1) = sI(nx,ny,nz,nt-1,1) - (0.D0,1.D0)*sI(nx,ny,nz,nt-1,2)
   zsI2x2(1,0) = sI(nx,ny,nz,nt-1,1) + (0.D0,1.D0)*sI(nx,ny,nz,nt-1,2)

   DO nii = 0,1; DO ni = 0,1
      DO ns = 0,1
         zsSI2x2(ns,ns,ni,nii) = zsSI2x2(ns,ns,ni,nii) &
              + (0.D0,1.D0)*CDSQRT(cI*atovera*(1.D0,0.D0))*zsI2x2(ni,nii)
      END DO
   END DO; END DO
```

⋮

```
DO iso = 1,3

   pi1 = 0.D0 &
        + 1.D0/2.D0*pion(MOD(nx+1,L),ny,nz,nt-1,iso) &
        - 1.D0/2.D0*pion(MOD(nx-1+L,L),ny,nz,nt-1,iso)

   pi2 = 0.D0 &
        + 1.D0/2.D0*pion(nx,MOD(ny+1,L),nz,nt-1,iso) &
        - 1.D0/2.D0*pion(nx,MOD(ny-1+L,L),nz,nt-1,iso)

   pi3 = 0.D0 &
        + 1.D0/2.D0*pion(nx,ny,MOD(nz+1,L),nt-1,iso) &
        - 1.D0/2.D0*pion(nx,ny,MOD(nz-1+L,L),nt-1,iso)

   zpi2x2(0,0) = pi3
   zpi2x2(1,1) = -pi3
   zpi2x2(0,1) = pi1 - (0.D0,1.D0)*pi2
   zpi2x2(1,0) = pi1 + (0.D0,1.D0)*pi2

   DO nii = 0,1; DO ni = 0,1
      DO nss = 0,1; DO ns = 0,1
         zsSI2x2(ns,nss,ni,nii) = zsSI2x2(ns,nss,ni,nii) &
              - gA*atovera/(2.D0*fpi*DSQRT(qpi3))*zpi2x2(ns,nss) &
              * ztau2x2(ni,nii,iso)
      END DO; END DO
   END DO; END DO

END DO

      ⋮
```

```
      DO np = 0,num-1
         DO nii = 0,1; DO ni = 0,1
            DO nss = 0,1; DO ns = 0,1
               zdualvecs(nx,ny,nz,nt-1,nss,nii,np) = zdualvecs(nx,ny,nz,nt-1,nss,nii,np) &
                    + zsSI2x2(ns,nss,ni,nii)*zdualvecs(nx,ny,nz,nt,ns,ni,np)
            END DO; END DO
         END DO; END DO
      END DO

    END DO; END DO; END DO

  END DO

END SUBROUTINE getzdualvecs
```

⋮

Hybrid Monte Carlo

We want to do efficient nonlocal updates of the auxiliary fields. Suppose we want to sample configurations according to the target probability

$$P_{\text{target}}(s) \propto \exp[-V(s)]$$

Hybrid Monte Carlo does this by introducing a conjugate momentum variable p_s for each variable s and sampling according classical molecular dynamics to the target probability

$$P_{\text{target}}[s, p_s] \propto \exp\{-H(s, p_s)\}$$
$$H(s, p_s) \equiv \frac{1}{2} \sum_{\vec{n}, n_t} [p_s(\vec{n}, n_t)]^2 + V(s)$$

Gottlieb, Liu, Toussaint, Renken, Sugar, Phys. Rev. D35, 2531 (1987)

Duane, Kennedy, Pendleton, Roweth, Phys. Lett. B195, 216 (1987)

We start by selecting the initial p_s configuration according to the random Gaussian distribution

$$P[p_s^0(\vec{n}, n_t)] \propto \exp\left\{-\tfrac{1}{2}\left[p_s^0(\vec{n}, n_t)\right]^2\right\}$$

Then we do classical molecular dynamics update of p_s and s which keep $H(s, p_s)$ approximately fixed. We use the leapfrog method which gives p_s a half step at the beginning and half step at the end, with full steps in between. In contrast s gets full steps at every stage.

Initial half step for p_s:

$$\tilde{p}_s^0(\vec{n}, n_t) = p_s^0(\vec{n}, n_t) - \frac{\varepsilon_{\text{step}}}{2}\left[\frac{\partial V(s)}{\partial s(\vec{n}, n_t)}\right]_{s=s^0}$$

Full steps for s and p_s:

$$s^{i+1}(\vec{n}, n_t) = s^i(\vec{n}, n_t) + \varepsilon_{\text{step}} \tilde{p}_s^i(\vec{n}, n_t)$$
$$\tilde{p}_s^{i+1}(\vec{n}, n_t) = \tilde{p}_s^i(\vec{n}, n_t) - \varepsilon_{\text{step}} \left[\frac{\partial V(s)}{\partial s(\vec{n}, n_t)} \right]_{s=s^{i+1}}$$

Cut the last step for p_s so it is a half step:

$$p_s^{N_{\text{step}}}(\vec{n}, n_t) = \tilde{p}_s^{N_{\text{step}}}(\vec{n}, n_t) + \frac{\varepsilon_{\text{step}}}{2} \left[\frac{\partial V(s)}{\partial s(\vec{n}, n_t)} \right]_{s=s^0}$$

We accept the new configurations for s and p_s if the uniform random number r between 0 and 1 satisfies

$$r < \exp\left[-H(s^{N_{\text{step}}}, p_s^{N_{\text{step}}}) + H(s^0, p_s^0) \right]$$

Then return back and repeat the steps listed above.

For our leading order auxiliary-field projection Monte Carlo calculations we have the target probability equal to

$$\exp[-V] = |Z(s, s_I, \pi_I, L_t)| \exp\left[-S_{ss}(s) - S_{s_I s_I}(s_I) - S_{\pi_I \pi_I}(\pi_I)\right]$$

where $Z(s, s_I, \pi_I, L_t)$ is the determinant of the $A \times A$ matrix of single nucleon amplitudes $\mathbf{Z}(s, s_I, \pi_I, L_t)$. The derivative of V with respect to an element of s is computed using

$$\begin{aligned}\frac{\partial V}{\partial s(\vec{n},n_t)} &= \frac{\partial S_{ss}(s)}{\partial s(\vec{n},n_t)} - \frac{\partial \mathrm{Re}[\ln(\det \mathbf{Z})]}{\partial s(\vec{n},n_t)} \\ &= \frac{\partial S_{ss}(s)}{\partial s(\vec{n},n_t)} - \mathrm{Re}\left[\sum_{k,l} \mathbf{Z}^{-1}_{lk} \frac{\partial \mathbf{Z}_{kl}}{\partial s(\vec{n},n_t)}\right]\end{aligned}$$

```fortran
PROGRAM nuclei

  IMPLICIT integer(i-n)
  IMPLICIT double precision(a-h,o-y)
  IMPLICIT complex*16(z)

  INCLUDE 'input.f90'
  INCLUDE 'mpif.h'

  DIMENSION s(0:L-1,0:L-1,0:L-1,0:Lt-1)
  DIMENSION p_s(0:L-1,0:L-1,0:L-1,0:Lt-1)
  DIMENSION snew(0:L-1,0:L-1,0:L-1,0:Lt-1)
  DIMENSION p_snew(0:L-1,0:L-1,0:L-1,0:Lt-1)
  DIMENSION sHMC(0:L-1,0:L-1,0:L-1,0:Lt-1,0:nHMC)
  DIMENSION p_sHMC(0:L-1,0:L-1,0:L-1,0:Lt-1,0:nHMC)

  DIMENSION sI(0:L-1,0:L-1,0:L-1,0:Lt-1,1:3)
  DIMENSION p_sI(0:L-1,0:L-1,0:L-1,0:Lt-1,1:3)
  DIMENSION sInew(0:L-1,0:L-1,0:L-1,0:Lt-1,1:3)
  DIMENSION p_sInew(0:L-1,0:L-1,0:L-1,0:Lt-1,1:3)
  DIMENSION sIHMC(0:L-1,0:L-1,0:L-1,0:Lt-1,1:3,0:nHMC)
  DIMENSION p_sIHMC(0:L-1,0:L-1,0:L-1,0:Lt-1,1:3,0:nHMC)

  DIMENSION pion(0:L-1,0:L-1,0:L-1,0:Lt-1,1:3)
  DIMENSION pionnew(0:L-1,0:L-1,0:L-1,0:Lt-1,1:3)
  DIMENSION p_pion(0:L-1,0:L-1,0:L-1,0:Lt-1,1:3)
  DIMENSION p_pionnew(0:L-1,0:L-1,0:L-1,0:Lt-1,1:3)
  DIMENSION pionHMC(0:L-1,0:L-1,0:L-1,0:Lt-1,1:3,0:nHMC)
  DIMENSION p_pionHMC(0:L-1,0:L-1,0:L-1,0:Lt-1,1:3,0:nHMC)

        ⋮
```

```
      DIMENSION dVds(0:L-1,0:L-1,0:L-1,0:Lt-1)
      DIMENSION dVdsI(0:L-1,0:L-1,0:L-1,0:Lt-1,1:3)
      DIMENSION dVdpion(0:L-1,0:L-1,0:L-1,0:Lt-1,1:3)
      DIMENSION zdVall(0:L-1,0:L-1,0:L-1,0:Lt-1,0:3,0:3)
      DIMENSION zvecs(0:L-1,0:L-1,0:L-1,0:Lt,0:1,0:1,0:n_f-1)
      DIMENSION zdualvecs(0:L-1,0:L-1,0:L-1,0:Lt,0:1,0:1,0:n_f-1)
      DIMENSION zwave(0:L-1,0:L-1,0:L-1,0:1,0:1,0:n_f-1)
      DIMENSION zdualwave(0:L-1,0:L-1,0:L-1,0:1,0:1,0:n_f-1)
      DIMENSION zcorrmatrix(0:n_f-1,0:n_f-1)
      DIMENSION zcorrinv(0:n_f-1,0:n_f-1)
      DIMENSION zdcorrmatrix(0:n_f-1,0:n_f-1)
      DIMENSION ztau2x2(0:1,0:1,0:3)

!*************************************************************

      CALL MPI_INIT(ierr)
      CALL MPI_COMM_RANK(MPI_COMM_WORLD,myid,ierr)
      CALL MPI_COMM_SIZE(MPI_COMM_WORLD,numprocs,ierr)

      IF (myid .eq. 0) THEN
         CPUtime_0 = MPI_Wtime()
      END IF

      !     spin/isospin conventions:
      !
      !     spin-up ---- ns = 0
      !     spin-down -- ns = 1
      !
      !     proton ----- ni = 0
      !     neutron ---- ni = 1

            ⋮
```

```
DO ntrial = 1,ntot

   nconfig = ntrial-ntherm
   mconfig = nconfig/measureevery

   DO nt = 0,Lt-1
      DO nz = 0,L-1; DO ny = 0,L-1; DO nx = 0,L-1
         CALL gaussrnd(gr)
         p_s(nx,ny,nz,nt) = gr
         DO iso = 1,3
            CALL gaussrnd(gr)
            p_sI(nx,ny,nz,nt,iso) = gr
         END DO
         DO iso = 1,3
            CALL gaussrnd(gr)
            p_pion(nx,ny,nz,nt,iso) = gr
         END DO
      END DO; END DO; END DO
   END DO
```

⋮

```
bose = 0.D0
DO nt = 0,Lt-1
   DO nz = 0,L-1; DO ny = 0,L-1; DO nx = 0,L-1
      bose = bose &
           + s(nx,ny,nz,nt)**2.D0/2.D0 &
           + p_s(nx,ny,nz,nt)**2.D0/2.D0
      DO iso = 1,3
         bose = bose &
              + sI(nx,ny,nz,nt,iso)**2.D0/2.D0 &
              + p_sI(nx,ny,nz,nt,iso)**2.D0/2.D0 &
              + pion(nx,ny,nz,nt,iso)**2.D0/2.D0 &
              + atovera/qpi3*pion(nx,ny,nz,nt,iso)*( &
              - pion(MOD(nx+1,L),ny,nz,nt,iso) &
              - pion(nx,MOD(ny+1,L),nz,nt,iso) &
              - pion(nx,ny,MOD(nz+1,L),nt,iso)) &
              + p_pion(nx,ny,nz,nt,iso)**2.D0/2.D0
      END DO
   END DO; END DO; END DO
END DO
```

⋮