1. Install(for Linux)

1.1. requirement

```
see
https://www.intel.cn/content/www/cn/zh/developer/tools/oneapi/toolkits.html#base-kit
Intel® oneAPI Math Kernel Library
Intel® Distribution for Python*
Intel® C++ Compiler Classic
```

Suggested to directly install two integrated packages:

1. Intel oneAPI Base Toolkit
2. Intel® oneAPI HPC Toolkit

The current version of oneAPI has removed Python from the above integrated packages, so a separate download of the corresponding Python interpreter,

Intel® Distribution for Python*

is required.

Please add the necessary configurations to the bashrc file:

```
echo "source /opt/intel/oneapi/setvars.sh" >> ~/.bashrc
```

1.2. compile

Unzip the compressed file `FQHE_SPHERE.zip`, enter the extracted directory, ensure that the icc and icpc compilation commands are available, and verify that the MKL (Math Kernel Library) path is correct.

```
hu@gugugu:~$ echo ${MKLROOT}
/opt/intel/oneapi/mkl/2022.0.2
```

Execute

```
bash compile.sh
```

2. Usage

```
hu@gugugu:~$ which python3
/opt/intel/oneapi/intelpython/latest/bin/python3
```

2.1. ED

The following script directly calculates the energy spectrum of the critical point when $N = 12$, $Z2 = 1$, $PH = 1$, and $L_z = 0$.

```
#!/opt/intel/oneapi/intelpython/latest/bin/python3
# -*- coding:utf-8 -*


import numpy as np
import sys
sys.path.append(r"/home/hu/study/Code/FQHE_Package/")  # the path to package
import FQHE_SPHERE as fqhe


n = 12;
Z2 = 1;
PH = 1;
Lz = 0;

inter = fqhe.interaction()
inter.setpseudo2body(inten=[0.475,0.1])
intra = fqhe.interaction()
intra.setpseudo2body(inten=[0.0,0.0])

kk = fqhe.fermion.QHFM(n,n,num_thread=8)
kk.setL2ab(a=0.0,b=0.0)

sol = fqhe.fermion.dl_solver(kk, intra,intra,inter, "H2L2",qn=
{"Lz":Lz,"Z_2":Z2,"PH":PH}, t=0.316)
eigvsL2_p = sol.eigsv(k=5,tol=10**(-6))
eigvs = sorted( eigvsL2_p )


ge = eigvsL2_p[0].get("energy")#-1.86886840
gap1 = eigvsL2_p[3].get("energy") - ge#-0.62067682 - ge
resu = "<Index>\t<L^2>\tPH\tZ_2\t<H+L^2>\t\tE_n=<H>\t\tEn-E0\t\tscale\n"
i = 0
for vec in eigvs:
    resu += "{:2d}\t".format( i )
    resu += "{:+.2f}\t".format( vec.get("L2") )
    resu += "{:+d}\t".format( int(vec.get("PH")) )
    resu += "{:+d}\t".format( int(vec.get("Z_2")) )
    resu += "{:+.8f}\t".format( vec.eig)
    resu += "{:+.8f}\t".format( vec.get("energy") )
    resu += "{:+.8f}\t".format( vec.get("energy")-ge )
    resu += "{:+.8f}\n".format(3*(vec.get("energy")-ge)/gap1)
    i += 1
print(resu)
```

output

```
matvec(self, vector):  96  times
eigsv: Measure: H2L2
total  0.7496089935302734 : Hamiltonian( 0.5623340606689453 ) lanczos(
0.17470121383666992 )
<Index> <L^2>   PH      Z_2     <H+L^2>         E_n=<H>         En-E0
scale
 0      -0.00   +1      +1      -1.86886840     -1.86886840     +0.00000000
+0.00000000
 1      -0.00   +1      +1      -1.28306494     -1.28306494     +0.58580346
+1.40796525
 2      +2.00   +1      +1      -0.87103274     -0.87103274     +0.99783566
+2.39827525
 3      +6.00   +1      +1      -0.62067682     -0.62067682     +1.24819158
+3.00000000
 4      +6.00   +1      +1      -0.45961655     -0.45961655     +1.40925185
+3.38710471
```

Detailed explanation:

2.1.1. fqhe.fermion.QHFM

Quantum Hall ferromagnetic(QHFM) class

```
class QHFMFermionDL(FermionDL):
    def __init__(self, n, n_o, dtype=np.float64, projectionQ=0, num_thread=8):
```

parameters:

```
1. n::Integer
   the number of electrons

2.  n_o::Integer
   the number of orbits(**not flux**)

3. dtype: Deprecated, not editable.

4. projectionQ: Deprecated, not editable.

5. num_thread::Integer
   The number of threads used for the computation.
```

method:

```
kk.setL2ab(a=0.0,b=0.0)
```

modify the Hamiltonian from H to:

$$H + a \cdot (L^2 - b)^2 \quad \text{if} \quad b \neq 0$$
$$H + a \cdot L^2 \quad \text{if} \quad b = 0$$

This modification is employed to select specific values of ℓ (angular momentum quantum number), but in general, convergence becomes difficult.

2.1.2. Interaction

Two body interaction $V_{\sigma\sigma'}(r_{12})$ class

```
inter = fqhe.interaction()
inter.setpseudo2body(inten=[V0,V1,V2,......])
```

V_i are Haldane Pseudo-potential coefficients.

2.1.3. solver

eigensolver

```
class fqhe_dl_solver():
    def __init__(self, ins, intra1, intra2, inter, itype, qn={"Lz": 0}, cdw=np.zeros(2, dtype=np.float64), t=0.0):
        '''
        ins: fqhe.fermion.QHFM object used for the computation.
        intra1, intra2: Intralayer interactions.
        inter: Interlayer interaction.
        itype: "twobody", "L2" or "H2L2"
                "twobody": H = H2 = V_{updown} + V_{upup} + V_{downdown} + t * n^x_{00}
                "L2": H = L^2
                "H2L2": H = H2 + a * (L^2 - b)^2 or H = H2 + a * L^2
        qn: Quantum numbers of the Hilbert space
                {"Lz": 0}: Lz=0
                {"Lz": 0, "Z_2": 1}: Lz=0 and Z_2=1
                {"Lz": 0, "Z_2": 1, "PH": 1}: Lz=0, Z_2=1, and PH=1
        cdw: (invalid)
        dtype: (invalid)
        t: Hopping term
        '''
```

Explanation:
This is a Python class named `fqhe_dl_solver`. The class constructor `__init__` takes several parameters to initialize the solver object. Here's an explanation of the parameters:

- `ins`: An object of the `fqhe.fermion.QHFM` class is used for the computation.
- `intra1`, `intra2`: Intralayer interactions.
- `inter`: Interlayer interaction.
- `itype`: Specifies the type of Hamiltonian to be used for the computation. It can take one of three values: "twobody", "L2", or "H2L2". Depending on the value, the Hamiltonian will be set accordingly.
- `qn`: Quantum numbers of the Hilbert space represented as a dictionary. It can contain keys such as "Lz", "Z_2", and "PH" to specify specific quantum numbers for the Hilbert space.

- `cdw`: (invalid) An unspecified parameter, possibly intended for future use.
- `dtype`: (invalid) An unspecified parameter, possibly intended for future use.
- `t`: Hopping term used in the computation.

Note: The comments inside the `__init__` function provide explanations for each parameter's purpose and usage.

```
eigvsL2_p = sol.eigsv(k=5,tol=10**(-6))
'''
    k: the number of states from Lanczos
    tol: the tolerence of Lanczos
'''
```

2.2. Measure

```
eigvsL2_p = sol.eigsv(k=5,tol=10**(-6))
```

After running the Lanczos algorithm to solve for low-energy states, it will return a list composed of instances of the `FQHE_SPHERE.core.eigenvector.eigenvector` class.

```
>> type(eigvsL2_p[0])
FQHE_SPHERE.core.eigenvector.eigenvector
```

`FQHE_SPHERE.core.eigenvector.eigenvector` is a class that encapsulates an eigenvector along with various quantum numbers (energy, L_z, L^2, Z_2, PH). It allows direct access to these properties for inspection.

```
>> eigvsL2_p[0]
The eigenvector in sector: {'Lz': 0, 'Z_2': 1, 'PH': 1}with E =
-1.8688683997894542, dimension = 31681.
measured qn: {'energy': -1.8688683997894506, 'L2': -5.064787549425714e-16}
'''
1. The eigenvector in sector: {'Lz': 0, 'Z_2': 1, 'PH': 1}
    {'Lz': 0, 'Z_2': 1, 'PH': 1} the quantum number of the Hilbert space
2. E = -1.8688683997894542
    eigenvalue of Hamiltonian <H2 + a*(L^2-b)^2> or <H = H2 + a*L^2>
3. dimension = 31681
    the dimension of Hilbert space
4. measured qn: {'energy': -1.8688683997894506, 'L2': -5.064787549425714e-16}
    the measured quantum number
    energy  <H2>
    L^2     <L2>
'''
```

By using the `.vec` method, you can directly obtain the eigenvector in `numpy` array format.

```
>> eigvsL2_p[0].vec # vacuum
array([ 0.16680578,  0.0822094 , -0.0822094 , ..., -0.04158522,
        0.04135224,  0.0822094 ])
>> np.dot( eigvsL2_p[0].vec, eigvsL2_p[0].vec)
0.9999999999999994
>> eigvsL2_p[3].vec # energy stress tensor
The eigenvector in sector: {'Lz': 0, 'Z_2': 1, 'PH': 1}with E =
-0.6206768230208931, dimension = 31681.
measured qn: {'energy': -0.6206768230208921, 'L2': 5.999999999999992}
>> np.dot( eigvsL2_p[0].vec, eigvsL2_p[3].vec)
-4.85722573273506e-17
```

When selecting the parameter `itype="H2L2"` in `fqhe.fermion.dl_solver`, it will automatically compute all quantum numbers and save them in the result class.

Alternatively, you can also use class methods to perform the calculation.

```

class eigenvector():
    .
    .
    .
    @property
    def density(self):
        return self.fqhe.density(self)

    @property
    def sigma_x(self):
        return self.fqhe.sigma_x(self)

    @property
    def Z2(self):
        if self.qn.get("Z_2")!=None:
            return self.qn.get("Z_2")
        elif self.measured_qn.get("Z_2")!=None:
            return self.measured_qn.get("Z_2")
        resu = self.fqhe.Z2(self)
        self.add_measured_qn({"Z_2":resu})
        return resu

    @property
    def PH(self):
        if self.qn.get("PH")!=None:
            return self.qn.get("PH")
        elif self.measured_qn.get("PH")!=None:
            return self.measured_qn.get("PH")
        resu = self.fqhe.PH(self)
        self.add_measured_qn({"PH":resu})
        return resu
```

The calculation of L^2 can be a bit complicated. To compute L^2 in `fqhe.fermion.dl_solver`, choose the parameter `itype="L2"`, and then use a class method to obtain the sparse matrix representation of the L^2 operator.

```
>> kk = fqhe.fermion.QHFM(n,n,num_thread=8)
>> sol = fqhe.fermion.dl_solver(kk, intra,intra,inter, "L2",qn=
{"Lz":Lz,"Z_2":Z2,"PH":PH})
>> L2 = sol.Hamiltonian() # get the matrix form of L^2
>> T = eigvsL2_p[3].vec
>> L2.matvec(T)@T # <T|L^2|T>
5.999999999999992
```

3. Density operator

3.1. $\hat{n}^A_{l,m}(m \neq 0)$

Def

$$
\begin{aligned}
\hat{n}^A_{l,m} &= \int d\Omega Y^*_{l,m}(\Omega)\hat{n}^A(\Omega) \\
&= (2s+1)\sqrt{\frac{2l+1}{4\pi}}\sum_{m_1}(-1)^{3s+m_1+l}\begin{pmatrix} s & l & s \\ -s & 0 & s \end{pmatrix}\begin{pmatrix} s & l & s \\ -m_1 & m & m_1-m \end{pmatrix}\hat{c}^\dagger_{m_1}A\hat{c}_{m_1-m} \\
&= \sqrt{\frac{(2s+1)^2}{4\pi(2l+1)}}\sum_{m_1}(-1)^{3s+l+m+m_1}\langle s,s;s,-s|s,s;l,0\rangle\langle s,m_1-m;s,-m_1|s,s;l,-m\rangle\hat{c}^\dagger_{m_1}A\hat{c}_{m_1-m} \\
&= \sum_{j_1,j_2}\mathcal{C}_{j_1,j_2}\hat{c}^\dagger_{j_1}A\hat{c}_{j_2}\delta_{j_1,j_2+m}
\end{aligned}
$$

and coefficient

$$
\mathcal{C}_{j_1,j_2} = \sqrt{\frac{(2s+1)^2}{4\pi(2l+1)}}(-1)^{3s+l+m+j_1}\langle s,s;s,-s|l,0\rangle\langle s,j_2;s,-j_1|l,-m\rangle\delta_{j_1,j_2+m}
$$

Since the operator $\hat{n}^A_{l,m}(m \neq 0)$ change the QN, see next Sec. for more detailed usage.

The recommended standard usage is

```
kk = fqhe.fermion.QHFM(n,n,dtype=np.float64, num_thread=8)
kk.Hilbertspace(qn={"Lz":0})
psi1 = kk.nlmAOp(tt[0].vec, l=4, m=-3, A=np.array([[1.0,2],[3,4]]), qn={"Lz":0})
```

and the definition of this interface

```
def nlmAOp(self, vec, l=0, m=0, A=np.array([[1.0,0],[0,-1]]), qn={"Lz":0}):
'''
    vec: input state |a>
    l, m: parameters in operator(defnition)
    A: 2*2 matrix
    qn: the quantum number of the input state |a>
'''
```

Measure L^2

```
print(np.linalg.norm(psi1))
psi1 /= np.linalg.norm(psi1) # normalization
L2 = kk.L2H(qn={"Lz":-3}) # L^2 m=-3
L2.matvec(psi1)@psi1/(np.linalg.norm(psi1)**2) # measur L^2
>> 20.0
```

4. Pairing Operator

4.1. Definition

$$\Delta^A_{l,m} = \sum_{m_1}(-1)^m\sqrt{2l+1}\begin{pmatrix} s & s & l \\ m_1 & m-m_1 & -m\end{pmatrix}c_{m-m_1}Ac_{m_1} = \langle s,m_1;s,m-m_1|l,m\rangle c_{m-m_1}Ac_{m_1},$$

$$(\Delta^A_{l,m})^\dagger = \sum_{m_1}(-1)^m\sqrt{2l+1}\begin{pmatrix} s & s & l \\ m_1 & m-m_1 & -m\end{pmatrix}c^\dagger_{m_1}A^\dagger c^\dagger_{m-m_1} = \langle s,m_1;s,m-m_1|l,m\rangle c^\dagger_{m_1}A^\dagger c^\dagger_{m-m_1}$$

and

$$\begin{aligned}(\mathcal{O}^{l_a,l_b}_{l,m})^{AB} &= \sum_{m_a,m_b}(-1)^{m_b}\begin{pmatrix} l_a & l_b & l \\ m_a & -m_b & -m\end{pmatrix}(\Delta^A_{l_a,m_a})^\dagger(\Delta^B_{l_b,m_b}) \\ &= \sum_{m_a,m_b}\frac{(-1)^{l_a-l_b+m_b+m}}{\sqrt{2l+1}}\langle l_a,m_a;l_b,-m_b|l,m\rangle(\Delta^A_{l_a,m_a})^\dagger(\Delta^B_{l_b,m_b})\end{aligned}$$

4.2. $\Delta^A_{l,m}$ and $(\Delta^A_{l,m})^\dagger$

First, the operator $\Delta^A_{l,m}$ transforms Hilbert space from $\mathcal{H}(N_e,L_z)$ to $\mathcal{H}(N_e-2,L_z-m)$ and $(\Delta^A_{l,m})^\dagger$ transforms from $\mathcal{H}(N_e,L_z)$ to $\mathcal{H}(N_e+2,L_z+m)$. For example, if $|a\rangle \in \mathcal{H}(N_e,L_z)$ and

$$|b\rangle = \Delta^A_{l,m}|a\rangle$$

we have $|b\rangle \in \mathcal{H}(N_e-2,L_z-m)$.

Second, the fermion(boson) statistics require that the operator $\Delta^A_{l,m}$ vanish if $2s+1+l$ is odd(even).

Since the change of particle number N_e and L_z is not considered in the python-cpp interface, we should carefully call this interface.

The recommended standard usage is

```
kk = fqhe.fermion.QHFM(n,n,dtype=np.float64, num_thread=8)
kk.Hilbertspace(qn={"Lz":0})
psi1 = kk.PairingAlmOp(tt[0].vec, l=2, m=-2, A=np.array([[1.0,0],[0,0]]),
daggerQ=0, qn={"Lz":0})
print(np.linalg.norm(psi1))
```

where the first two lines are used to set the Hilbert space in cpp code to $\mathcal{H}(N_e = 12, L_z = 0)$. The first line is used to set the particle number and the second is to set L_z.

The third line is the interface of operators $\Delta^A_{l,m}$ and $(\Delta^A_{l,m})^\dagger$

```
def PairingAlmOp(self, vec, l=0, m=0, A=np.array([[1.0,0],[0,-1]]), daggerQ=0, qn=
{"Lz":0}):
'''
    vec: input state |a>
    l, m: parameters in operator(defnition)
    A: 2*2 matrix
    daggerQ: 0 for $\Delta_{l,m}^A$ and 1 for $(\Delta_{l,m}^A)^\dagger$
    qn: the quantum number of the input state |a>
'''
```

and the forth line is used to check the norm of result.

In this example, we set $N_e = 12, L_z = 0, l = 2, m = -2$, thus $2s + 1 + l = 14$ is even and the result state is in Hilbert space $\mathcal{H}(N_e = 10, L_z = 2)$.

After calling this function, the code automatically set the Hilbert space in cpp code to $\mathcal{H}(N_e = 10, L_z = 2)$.

Measure L^2

```
psi1 /= np.linalg.norm(psi1) # normalization
L2 = kk.L2H(qn={"Lz":2}) # L^2
L2.matvec(psi1)@psi2/(np.linalg.norm(psi1)**2) # measur L^2
>> 6.0000000000000036
```

Although the Hilbert space in cpp code has been set to the correct, we still need to tell the program the value of the angular momentum $L_z = 2$.(`qn={"Lz":2}`).

Example for $(\Delta^A_{l,m})^\dagger$

```
kk = fqhe.fermion.QHFM(n,n,dtype=np.float64, num_thread=8)
kk.Hilbertspace(qn={"Lz":0})
psi1 = kk.PairingAlmOp(tt[0].vec, l=4, m=3, A=np.array([[1.0,2],[3,4]]),
daggerQ=0, qn={"Lz":0})#daggerQ=1
print(np.linalg.norm(psi1))
psi1 /= np.linalg.norm(psi1) # normalization
L2 = kk.L2H(qn={"Lz":-3}) # L^2 qn={"Lz":3}
L2.matvec(psi1)@psi1/(np.linalg.norm(psi1)**2) # measur L^2
>> 19.999999999999996
```

4.3. $(\mathcal{O}_{l,m}^{l_a,l_b})^{AB}$

The right pairing operator

$$
\begin{aligned}
(\mathcal{O}_{l,m}^{l_a,l_b})^{AB} &= \sum_{m_a,m_b} (-1)^{m_b} \begin{pmatrix} l_a & l_b & l \\ m_a & -m_b & -m \end{pmatrix} (\Delta^A_{l_a,m_a})^\dagger (\Delta^B_{l_b,m_b}) \\
&= \frac{(-1)^{l_a-l_b+m_b+m}}{\sqrt{2l+1}} \sum_{m_a,m_b} \langle l_a, m_a; l_b, -m_b | l_a, l_b; l, m\rangle (\Delta^A_{l_a,m_a})^\dagger (\Delta^B_{l_b,m_b}) \\
&= \frac{(-1)^{l_a-l_b+m_b+m}}{\sqrt{2l+1}} \sum_{m_a,m_b,m_1,m_2} \langle l_a, m_a; l_b, -m_b | l_a, l_b; l, m\rangle \langle s, m_1; s, m_a - m_1 | s, s; l_a, m_a\rangle \\
&\quad \langle s, m_2; s, m_b - m_2 | s, s; l_b, m_b\rangle c^\dagger_{m_1} A^\dagger c^\dagger_{m_a-m_1} c_{m_b-m_2} B c_{m_2} \\
&= \sum_{j_1,j_2,j_3,j_4} \mathcal{C}_{j_1,j_2,j_3,j_4} c^\dagger_{j_1} A^\dagger c^\dagger_{j_2} c_{j_3} B c_{j_4} \delta_{j_1+j_2-j_3-j_4,m}
\end{aligned}
$$

and let

$$
\begin{pmatrix}
j_1 & \to & m_1 & m_1 & \to & j_1 \\
j_2 & \to & m_a - m_1 & m_2 & \to & j_4 \\
j_3 & \to & m_b - m_2 & m_a & \to & j_1 + j_2 \\
j_4 & \to & m_2 & m_b & \to & j_3 + j_4
\end{pmatrix}
$$

we have the coefficient

$$
\begin{aligned}
\mathcal{C}_{j_1,j_2,j_3,j_4} = &\frac{(-1)^{l_a-l_b+m_b+m}}{\sqrt{2l+1}} \langle l_a, j_1 + j_2; l_b, -j_3 - j_4 | l_a, l_b; l, m\rangle \\
&\langle s, j_1; s, j_2 | s, s; l_a, j_1 + j_2\rangle \langle s, j_4; s, j_3 | s, s; l_b, j_3 + j_4\rangle \delta_{j_1+j_2-j_3-j_4,m}
\end{aligned}
$$

```
kk = fqhe.fermion.QHFM(n,n,dtype=np.float64, num_thread=8)
kk.Hilbertspace(qn={"Lz":0})
psi1 = kk.PairingABlmlalbOp(tt[0].vec, l=4, m=-2, la=2, lb=2, A=np.array([[1.0,2],
[3,4]]),  B=np.array([[1.0,3],[5,7]]), qn={"Lz":0}) #
```

The third line is the interface of operators

```
def PairingABlmlalbOp(self, vec, l=0, m=0, la=0, lb=0, A=np.array([[1.0,0],
[0,-1]]), B=np.array([[1.0,0],[0,-1]]), qn={"Lz":0}):
'''
    vec: input state |a>
    l, m, la, lb: parameters in operator(defnition)
    A, B: 2*2 matrix
    qn: the quantum number of the input state |a>
'''
```

check the L^2 of result

```
print(np.linalg.norm(psi1))
psi1 /= np.linalg.norm(psi1) # normalization
L2 = kk.L2H(qn={"Lz":-2}) # L^2
L2.matvec(psi1)@psi1/(np.linalg.norm(psi1)**2) # measur L^2
>> 20.0
```