

A GPU simulation of skyrmion

(Dated: September 12, 2018)

A GPU simulation of skyrmion.

I. LLG

The Landau-Lifshitz-Gilbert-Equation (LLG) can be written as (Eq. (13) of Ref. [1] and Eq. (7) of Ref. [2], NOTE that the sign of the last term is ‘+’ in Eq. (7) of Ref. [2])

$$\dot{\mathbf{n}} = \gamma \mathbf{B}_{\text{eff}} \times \mathbf{n} - \frac{\alpha\gamma}{|\mathbf{n}|} \mathbf{n} \times \dot{\mathbf{n}} - \frac{\hbar\gamma}{2e} (\mathbf{j} \cdot \nabla) \mathbf{n} \quad (1)$$

where \mathbf{n} is the magnetic momentum, $\gamma(\gamma > 0)$ is the gyromagnetic ratio, α represents Gilbert damping, \mathbf{B}_{eff} is the effective field arising from the spin Hamiltonian, and can be written as (H_S is from Ref. [2], before Eq. (1), also Eq. (4) in Ref. [3])

$$\begin{aligned} \mathbf{B}_{\text{eff}} &\equiv \frac{\delta H_S}{\delta \mathbf{n}} \\ H_S &= \int d^D x \frac{J}{2a} (\nabla \mathbf{n})^2 + \frac{D}{a^2} \mathbf{n} \cdot (\nabla \times \mathbf{n}) - \frac{\mu}{a^3} \mathbf{B} \cdot \mathbf{n} \end{aligned} \quad (2)$$

II. LLG IN 2D LATTICE

In the following, for simplicity, we use $\delta = a(\mathbf{e}_x, \mathbf{e}_y)$, $\delta_x = a\mathbf{e}_x$, $\delta_y = a\mathbf{e}_y$, where a is the distance between two lattice.

A. Spin torque

Written in lattice, so that

$$(\mathbf{j} \cdot \nabla) \mathbf{n} = \sum_i j_i \partial_i n_x \mathbf{e}_x + \sum_i j_i \partial_i n_y \mathbf{e}_y + \sum_i j_i \partial_i n_z \mathbf{e}_z \quad (3)$$

In 2D, it is

$$\begin{aligned} (\mathbf{j} \cdot \nabla) \mathbf{n} &= \sum_{i=x,y} j_i (\partial_i n_x \mathbf{e}_x + \partial_i n_y \mathbf{e}_y + \partial_i n_z \mathbf{e}_z) \\ &= \frac{1}{a} \sum_{i=x,y} j_i \left(\frac{n_x(\mathbf{r} + \delta_i) - n_x(\mathbf{r} - \delta_i)}{2} \mathbf{e}_x + \frac{n_y(\mathbf{r} + \delta_i) - n_y(\mathbf{r} - \delta_i)}{2} \mathbf{e}_y + \frac{n_z(\mathbf{r} + \delta_i) - n_z(\mathbf{r} - \delta_i)}{2} \mathbf{e}_z \right) \\ &= \frac{1}{a} \sum_{i=x,y} j_i \frac{\mathbf{n}(\mathbf{r} + \delta_i) - \mathbf{n}(\mathbf{r} - \delta_i)}{2} \end{aligned} \quad (4)$$

Sometimes, it is also written as (see last term in Eq. (8) in Ref. [4], it also says this is the discrete version of the continuous term $(\mathbf{j} \cdot \nabla) \mathbf{n}$ before Eq. (10))

$$(\mathbf{j} \cdot \nabla) \mathbf{n} = \frac{1}{a} \sum_{i=x,y} j_i \mathbf{n}(\mathbf{r}) \times \left(\frac{\mathbf{n}(\mathbf{r} + \delta_i) - \mathbf{n}(\mathbf{r} - \delta_i)}{2} \times \mathbf{n}(\mathbf{r}) \right) \quad (5)$$

that is because, for a unit vector, one have

$$\mathbf{n} \cdot \partial_i \mathbf{n} = 0 \quad (6)$$

the discrete version is

$$\mathbf{n}(\mathbf{r}) \cdot \frac{\mathbf{n}(\mathbf{r} + \delta_i) - \mathbf{n}(\mathbf{r} - \delta_i)}{2a} = 0 \quad (7)$$

so that

$$\begin{aligned}
& \mathbf{n}(\mathbf{r}) \times \left(\frac{\mathbf{n}(\mathbf{r} + \delta_i) - \mathbf{n}(\mathbf{r} - \delta_i)}{2} \times \mathbf{n}(\mathbf{r}) \right) \\
&= \frac{\mathbf{n}(\mathbf{r} + \delta_i) - \mathbf{n}(\mathbf{r} - \delta_i)}{2} (\mathbf{n}(\mathbf{r}) \cdot \mathbf{n}(\mathbf{r})) - \mathbf{n}(\mathbf{r}) \left(\mathbf{n}(\mathbf{r}) \cdot \frac{\mathbf{n}(\mathbf{r} + \delta_i) - \mathbf{n}(\mathbf{r} - \delta_i)}{2} \right) \\
&= \frac{\mathbf{n}(\mathbf{r} + \delta_i) - \mathbf{n}(\mathbf{r} - \delta_i)}{2}
\end{aligned} \tag{8}$$

to be consist with the references, we use

$$(\mathbf{j} \cdot \nabla) \mathbf{n} = \frac{1}{a} \sum_{i=x,y} j_i \mathbf{n}(\mathbf{r}) \times \left(\frac{\mathbf{n}(\mathbf{r} + \delta_i) - \mathbf{n}(\mathbf{r} - \delta_i)}{2} \times \mathbf{n}(\mathbf{r}) \right) \tag{9}$$

In simulation, to simplify, we always take the direction of \mathbf{j} as \mathbf{x} -axis, so we only have

$$\frac{1}{a} j_x \mathbf{n}(\mathbf{r}) \times \left(\frac{\mathbf{n}(\mathbf{r} + \delta_x) - \mathbf{n}(\mathbf{r} - \delta_x)}{2} \times \mathbf{n}(\mathbf{r}) \right) \tag{10}$$

B. J term

$(\nabla \mathbf{n})^2$ is confusing, in 2D, it is in fact $\sum_{i=x,y} (\partial_i \mathbf{n}) \cdot (\partial_i \mathbf{n})$ Consider in 1-dimension, we have

$$\begin{aligned}
& \mathbf{n}(x) \cdot \partial_i \mathbf{n}(x) = 0 \\
& 0 = \sum_{i=x,y} \partial_i (\mathbf{n}(x) \cdot \partial_i \mathbf{n}(x)) = \sum_{i=x,y} (\partial_i \mathbf{n}) \cdot (\partial_i \mathbf{n}) + \mathbf{n} \cdot \left(\sum_{i=x,y} (\partial_i)^2 \right) \mathbf{n}
\end{aligned} \tag{11}$$

In lattice, the derivate can be written as

$$\frac{d^2}{dx^2} \mathbf{n}(\mathbf{r}) = \frac{1}{a} \left(\mathbf{n}'(\mathbf{r} + \frac{a}{2} \mathbf{e}_x) - \mathbf{n}'(\mathbf{r} - \frac{a}{2} \mathbf{e}_x) \right) \tag{12}$$

with

$$\begin{aligned}
\mathbf{n}'(\mathbf{r} + \frac{a}{2} \mathbf{e}_x) &= \frac{1}{a} (\mathbf{n}(\mathbf{r} + a \mathbf{e}_x) - \mathbf{n}(\mathbf{r})) \\
\mathbf{n}'(\mathbf{r} - \frac{a}{2} \mathbf{e}_x) &= \frac{1}{a} (\mathbf{n}(\mathbf{r}) - \mathbf{n}(\mathbf{r} - a \mathbf{e}_x))
\end{aligned} \tag{13}$$

so that

$$\left(\sum_{i=x,y} (\partial_i)^2 \right) \mathbf{n} = \frac{1}{a^2} \sum_{i=x,y,-x,-y} \mathbf{n}(\mathbf{r} + \delta_i) - 4 \mathbf{n}(\mathbf{r}) \tag{14}$$

and

$$\mathbf{n} \cdot \left(\sum_{i=x,y} (\partial_i)^2 \right) \mathbf{n} = \frac{1}{a^2} \sum_{\langle i \rangle} \mathbf{n} \cdot \mathbf{n}_i - 4 \tag{15}$$

Throw away the constant term

$$\int d^D x \frac{J}{2a} (\nabla \mathbf{n})^2 = -\frac{J}{2a^3} \sum_{\langle i,j \rangle} \mathbf{n}_i \cdot \mathbf{n}_j \tag{16}$$

where j are all neighbours of i , $j = i + \delta_x, i - \delta_x, i + \delta_y, i - \delta_y$.

When J is constant, it can also been written as

$$\int d^D x \frac{J}{2a} (\nabla \mathbf{n})^2 = -\frac{J}{a^3} \sum_{\mathbf{r}} \mathbf{n}(\mathbf{r}) \cdot (\mathbf{n}(\mathbf{r} + \delta_x) + \mathbf{n}(\mathbf{r} + \delta_y)) \tag{17}$$

C. D term

In 2D, we find

$$\begin{aligned}
\mathbf{n} \cdot (\nabla \times \mathbf{n}) &= n_x \partial_y n_z - n_y \partial_x n_z + n_z \partial_x n_y - n_z \partial_y n_x \\
&= \frac{1}{2a} [(n_x(\mathbf{r})(n_z(\mathbf{r} + \delta_y) - n_z(\mathbf{r} - \delta_y)) - n_z(\mathbf{r})(n_x(\mathbf{r} + \delta_y) - n_x(\mathbf{r} - \delta_y))) \\
&\quad + (n_z(\mathbf{r})(n_y(\mathbf{r} + \delta_x) - n_y(\mathbf{r} - \delta_x)) - n_y(\mathbf{r})(n_z(\mathbf{r} + \delta_x) - n_z(\mathbf{r} - \delta_x)))] \\
&= \frac{1}{2a} [(n_x(\mathbf{r})n_z(\mathbf{r} + \delta_y) - n_z(\mathbf{r})n_x(\mathbf{r} + \delta_y)) - (n_x(\mathbf{r})n_z(\mathbf{r} - \delta_y) - n_z(\mathbf{r})n_x(\mathbf{r} - \delta_y)) \\
&\quad + (n_z(\mathbf{r})n_y(\mathbf{r} + \delta_x) - n_y(\mathbf{r})n_z(\mathbf{r} + \delta_x)) - (n_z(\mathbf{r})n_y(\mathbf{r} - \delta_x) - n_y(\mathbf{r})n_z(\mathbf{r} - \delta_x))] \\
&= -\frac{1}{2a} [\mathbf{n}(\mathbf{r}) \cdot (\mathbf{n}(\mathbf{r} + \delta_y) \times \mathbf{e}_y) - \mathbf{n}(\mathbf{r}) \cdot (\mathbf{n}(\mathbf{r} - \delta_y) \times \mathbf{e}_y) \\
&\quad + \mathbf{n}(\mathbf{r}) \cdot (\mathbf{n}(\mathbf{r} + \delta_x) \times \mathbf{e}_x) - \mathbf{n}(\mathbf{r}) \cdot (\mathbf{n}(\mathbf{r} - \delta_x) \times \mathbf{e}_x)] \\
&= -\frac{1}{2a^2} [\mathbf{n}(\mathbf{r}) \times \mathbf{n}(\mathbf{r} + \delta_y) \cdot \delta_y - \mathbf{n}(\mathbf{r}) \times \mathbf{n}(\mathbf{r} - \delta_y) \cdot \delta_y \\
&\quad + \mathbf{n}(\mathbf{r}) \times \mathbf{n}(\mathbf{r} + \delta_x) \cdot \delta_x - \mathbf{n}(\mathbf{r}) \times \mathbf{n}(\mathbf{r} - \delta_x) \cdot \delta_x]
\end{aligned} \tag{18}$$

so, one have

$$\int d^D x \frac{D}{a^2} \mathbf{n} \cdot (\nabla \times \mathbf{n}) = -\frac{D}{a^4} \sum_{\mathbf{r}} (\mathbf{n}(\mathbf{r}) \times \mathbf{n}(\mathbf{r} + \delta_x) \cdot \delta_x + \mathbf{n}(\mathbf{r}) \times \mathbf{n}(\mathbf{r} + \delta_y) \cdot \delta_y) \tag{19}$$

It is also

$$\int d^D x \frac{D}{a^2} \mathbf{n} \cdot (\nabla \times \mathbf{n}) = -\frac{D}{a^4} \sum_{\mathbf{r}} (\mathbf{n}(\mathbf{r} + \delta_x) \times \delta_x + \mathbf{n}(\mathbf{r} + \delta_y) \times \delta_y) \cdot \mathbf{n}(\mathbf{r}) \tag{20}$$

D. Effective magnetic field

Using the results, we can write the discrete version of H_s as

$$H_S = \sum_{\mathbf{r}} \left[-\frac{J}{a^3} (\mathbf{n}(\mathbf{r} + \delta_x) + \mathbf{n}(\mathbf{r} + \delta_y)) - \frac{D}{a^3} (\mathbf{n}(\mathbf{r} + \delta_x) \times \mathbf{e}_x + \mathbf{n}(\mathbf{r} + \delta_y) \times \mathbf{e}_y) - \frac{\mu}{a^3} \mathbf{B} \right] \cdot \mathbf{n}(\mathbf{r}) \tag{21}$$

which leads to

$$\begin{aligned}
\mathbf{B}_{\text{eff}}(\mathbf{r}) &= \frac{\delta H_S}{\delta \mathbf{n}} = -\frac{1}{a^3} \sum_{i=x,y} [J(\mathbf{r})\mathbf{n}(\mathbf{r} + \delta_i) + J(\mathbf{r} - \delta_i)\mathbf{n}(\mathbf{r} - \delta_i)] \\
&\quad - \frac{1}{a^3} \sum_{i=x,y} [D(\mathbf{r})\mathbf{n}(\mathbf{r} + \delta_i) \times \mathbf{e}_i - D(\mathbf{r} - \delta_i)\mathbf{n}(\mathbf{r} - \delta_i) \times \mathbf{e}_i] - \frac{\mu}{a^3} \mathbf{B}(\mathbf{r})
\end{aligned} \tag{22}$$

E. anisotropy

In some material, the effective Hamiltonian can be written with anisotropy terms as (if ignore a , this is as same as Eq. (10) in Ref. [5])

$$\begin{aligned}
H_S &= \sum_{\mathbf{r}} \left[-\frac{J}{a^3} (\mathbf{n}(\mathbf{r} + \delta_x) + \mathbf{n}(\mathbf{r} + \delta_y)) - \frac{D}{a^3} (\mathbf{n}(\mathbf{r} + \delta_x) \times \mathbf{e}_x + \mathbf{n}(\mathbf{r} + \delta_y) \times \mathbf{e}_y) - \frac{\mu}{a^3} \mathbf{B} \right. \\
&\quad \left. - \mathbf{h} \cdot \mathbf{n}(\mathbf{r}) - K (\mathbf{e}_z \cdot \mathbf{n}(\mathbf{r}))^2 \right]
\end{aligned} \tag{23}$$

The contribution of \mathbf{h} can be just put into the applied magnetic field \mathbf{B} , we also include the contribution of K .

F. dimensionless LLG

Using $\tau = \gamma t$

$$\frac{d}{d\tau} \mathbf{n} = \mathbf{B}_{\text{eff}} \times \mathbf{n} - \frac{\alpha\gamma}{|\mathbf{n}|} \mathbf{n} \times \dot{\mathbf{n}} - \frac{\hbar}{2e} (\mathbf{j} \cdot \nabla) \mathbf{n} \quad (24)$$

In the following, we use $t \rightarrow \tau$, and use dimensionless parameters.

Using dimensionless parameters, the LLG can be written as

$$\begin{aligned} \dot{\mathbf{n}} &= \mathbf{B}_{\text{eff}} \times \mathbf{n} - \alpha \mathbf{n} \times \dot{\mathbf{n}} - \sum_{i=x,y} j_i \mathbf{n}(\mathbf{r}) \times \left(\frac{\mathbf{n}(\mathbf{r} + \delta_i) - \mathbf{n}(\mathbf{r} - \delta_i)}{2} \times \mathbf{n}(\mathbf{r}) \right) \\ \mathbf{B}_{\text{eff}}(\mathbf{r}) &= \frac{\delta H_S}{\delta \mathbf{n}} = - \sum_{i=x,y} [J(\mathbf{r}) \mathbf{n}(\mathbf{r} + \delta_i) + J(\mathbf{r} - \delta_i) \mathbf{n}(\mathbf{r} - \delta_i)] \\ &\quad - \sum_{i=x,y} [D(\mathbf{r}) \mathbf{n}(\mathbf{r} + \delta_i) \times \mathbf{e}_i - D(\mathbf{r} - \delta_i) \mathbf{n}(\mathbf{r} - \delta_i) \times \mathbf{e}_i] - \mathbf{B}(\mathbf{r}) - 2K (\mathbf{e}_z \cdot \mathbf{n}(\mathbf{r})) \mathbf{e}_z \end{aligned} \quad (25)$$

Ignoring the anisotropy term, this is as same as Eq. (9) in Ref. [4].

III. EVALUATION

Let $\mathbf{N} = \mathbf{B}_{\text{eff}} \times \mathbf{n} - \sum_{i=x,y} j_i \mathbf{n}(\mathbf{r}) \times \left(\frac{\mathbf{n}(\mathbf{r} + \delta_i) - \mathbf{n}(\mathbf{r} - \delta_i)}{2} \times \mathbf{n}(\mathbf{r}) \right)$, we have

$$\dot{\mathbf{n}} = \mathbf{N} - \alpha \mathbf{n} \times \dot{\mathbf{n}} \quad (26)$$

This is in fact a combine of 3 equations which can be written as

$$\begin{aligned} \frac{d\mathbf{n}}{dt} &= \frac{1}{1 + \alpha^2} \\ &\times (N_x + \alpha(N_y n_z - n_y N_z) + \alpha^2 n_x \mathbf{n} \cdot \mathbf{N}, N_y + \alpha(n_x N_z - N_x n_z) + \alpha^2 n_y \mathbf{n} \cdot \mathbf{N}, N_z + \alpha(N_x n_y - n_x N_y) + \alpha^2 n_z \mathbf{n} \cdot \mathbf{N}) \end{aligned} \quad (27)$$

It can be written as

$$\frac{d\mathbf{n}}{dt} = \frac{1}{1 + \alpha^2} (\mathbf{N} + \alpha \mathbf{N} \times \mathbf{n} + \alpha^2 (\mathbf{n} \cdot \mathbf{N}) \mathbf{n}) \quad (28)$$

Then note that $\mathbf{n} \cdot \frac{d\mathbf{n}}{dt} = 0$ because it is a unit vector, so

$$\mathbf{n} \cdot \frac{d\mathbf{n}}{dt} = 0 = \mathbf{n} \cdot \mathbf{N}. \quad (29)$$

So

$$\frac{d\mathbf{n}}{dt} = \frac{1}{1 + \alpha^2} (\mathbf{N} + \alpha \mathbf{N} \times \mathbf{n}) \quad (30)$$

One can use this to evaluate

$$\mathbf{n}(t + \Delta t) = \mathbf{n}(t) + \Delta t (\mathbf{N}(t) + \alpha \mathbf{N}(t) \times \mathbf{n}(t)) \quad (31)$$

IV. SIMULATION

The simulation is running on GPU using the compute shader in Unity3D. The implementation of LLG is the LLG_c.compute, the content is

```

1 // Each #kernel tells which function to compile; you can have many kernels
2 #pragma kernel CSMMain
3
4 // Create a RenderTexture with enableRandomWrite flag and set it
5 // with cs.SetTexture
6 //RWStructuredBuffer<float3> magneticMomentum;
7 RWTexture2D<float4> magneticMomentumX;
8 RWTexture2D<float4> magneticMomentumY;
9 RWTexture2D<float4> magneticMomentumZ;
10
11 Texture2D<float4> boundaryCondition;
12 Texture2D<float4> exchangeStrength;
13 Texture2D<float4> jxPeroidFunction;
14
15 uint2 size;
16 float K;
17 float D;
18 float D0;
19 float B;
20 float alpha;
21 float timestep;
22 uint jxstep;
23 uint jxperoid;
24
25 [numthreads(8, 8, 1)]
26 void CSMMain (uint3 id : SV_DispatchThreadID)
27 {
28     float3 zero3 = float3(0.0f, 0.0f, 0.0f);
29
30     float3 s = float3(magneticMomentumX[id.xy].x, magneticMomentumY[id.xy].x, magneticMomentumZ[id.xy].x);
31
32     float3 sleft = id.x > 1 ? float3(magneticMomentumX[id.xy - uint2(1, 0)].x, magneticMomentumY[id.xy - uint2(1, 0)].x, magneticMomentumZ[id.xy - uint2(1, 0)].x) : zero3;
33     float3 sright = id.x < (size.x - 1) ? float3(magneticMomentumX[id.xy + uint2(1, 0)].x, magneticMomentumY[id.xy + uint2(1, 0)].x, magneticMomentumZ[id.xy + uint2(1, 0)].x) : zero3;
34     float3 sup = id.y > 1 ? float3(magneticMomentumX[id.xy - uint2(0, 1)].x, magneticMomentumY[id.xy - uint2(0, 1)].x, magneticMomentumZ[id.xy - uint2(0, 1)].x) : zero3;
35     float3 sdown = id.y < (size.y - 1) ? float3(magneticMomentumX[id.xy + uint2(0, 1)].x, magneticMomentumY[id.xy + uint2(0, 1)].x, magneticMomentumZ[id.xy + uint2(0, 1)].x) : zero3;
36
37     float edge = boundaryCondition[id.xy].r > 0.5f ? 1.0f : 0.0f;
38
39     float j_s = exchangeStrength[id.xy].x;
40     float j_left = id.x > 1 ? exchangeStrength[id.xy - uint2(1, 0)].x : 0.0f;
41     float j_up = id.y > 1 ? exchangeStrength[id.xy - uint2(0, 1)].x : 0.0f;
42
43     float d_s = D0 + D * j_s;
44     float d_left = id.x > 1 ? (D0 + D * j_left) : 0.0f;
45     float d_up = id.y > 1 ? (D0 + D * j_up) : 0.0f;
46
47     float3 vright = float3(1.0, 0.0, 0.0);
48     float3 vdown = float3(0.0, 1.0, 0.0);
49
50     float3 beff = (j_left * sleft + j_s * sright + j_up * sup + j_s * sdown)
51         + (d_s * cross(sright, vright) - d_left * cross(sleft, vright) + d_s * cross(sdown, vdown) - d_up *
52             cross(sup, vdown))
53         + float3(0.0f, 0.0f, B) + 2.0 * K * float3(0.0f, 0.0f, s.z);
54
55     jxstep++;
56     float jx = jxperoid > 0 ? jxPeroidFunction[uint2(jxstep % jxperoid, 0)].x : 0.0f;
57     float3 stt = -jx * cross(s, cross((sright - sleft) * 0.5f, s));
58
59     float3 newS = cross(s, beff) + stt;
60     newS = (newS - alpha * cross(s, newS)) / (1 + alpha * alpha);
61
62     float3 retColor = edge < 0.5f ? zero3 : normalize(s + timestep * newS);
63
64     magneticMomentumX[id.xy] = float4(retColor.x * edge, 0.0f, 0.0f, 1.0f);
65     magneticMomentumY[id.xy] = float4(retColor.y * edge, 0.0f, 0.0f, 1.0f);

```

```

65     magneticMomentumZ[id.xy] = float4(retColor.z * edge, 0.0f, 0.0f, 1.0f);
66 }

```

The magnetic momentum is a 512×512 64-bit ARGB texture, only R, G, B channel used. $\mathbf{n} = (2 \times r - 1, 2 \times g - 1, 2 \times b - 1)$.

The boundary condition is a 512×512 alpha 8-bit texture, only R channel used, when $R < 0.5$, it is a defect.

The exchange strength is a 32-bit RFloat texture, generated from a Lua script. For example, a constant exchange strength can be generated from a lua file as

```

1  -- Exchange Strength is constant
2  function GetJValueByLatticeIndex(x, y)
3      return 2.0
4  end
5
6  -- Need to register the function
7  return {
8      GetJValueByLatticeIndex = GetJValueByLatticeIndex,
9  }

```

While a pin with $J = 1 + \exp(-0.001\rho^2)$ at lattice index (255, 255) can be written as

```

1  -- Exchange Strength is pin
2  function GetJValueByLatticeIndex(x, y)
3      local j0 = 1
4      local j1 = 1
5      local j2 = 0.001
6      local rho = (x - 255) * (x - 255) + (y - 255) * (y - 255)
7
8      return j0 + j1 * math.exp(-1.0 * j2 * rho)
9  end
10
11 -- Need to register the function
12 return {
13     GetJValueByLatticeIndex = GetJValueByLatticeIndex,
14 }

```

Manual.pdf is a document introduce how to use the pre-built software.

-
- [1] Gen Tatara, Hiroshi Kohno, Junya Shibata, Phys. Rep. 468, 213-301 (2008), 10.1016/j.physrep.2008.07.003, arXiv:0807.2894.
 - [2] Jiadong Zang, Maxim Mostovoy, Jung Hoon Han, and Naoto Nagaosa, 10.1103/PhysRevLett.107.136804.
 - [3] Hong Chul Choi, Shi-Zeng Lin, Jian-Xin Zhu, Phys. Rev. B 93, 115112 (2016), 10.1103/PhysRevB.93.115112, arXiv:1601.00933.
 - [4] Ye-Hua Liu, You-Quan Li, J. Phys.: Condens. Matter 25 076005, 10.1088/0953-8984/25/7/076005, arXiv:1206.5661.
 - [5] Junichi Iwasaki, Wataru Koshibae, and Naoto Nagaosa, Nano. Lett. 2014, 14, 4432-4437, 10.1021/nl501379k.