# 横场Ising

我们知道对于横场算符，可以拆成：

$$S^x = \frac{1}{2}(S^+ + S^-) \tag{1}$$

它对自旋仍然具有翻转作用（无论遇到什么状态的自旋，有+有-保证总能被翻转）

对于sigma算符（Ising model 需要用到）：

$$\sigma^x = (\sigma^+ + \sigma^-) \tag{2}$$

要小心这个系数，这主要是因为$S^{\pm}$和$\sigma^{\pm}$的矩阵形式不再有1/2的差距:

$$\sigma^+ = S^+, \sigma^- = S^- \tag{3}$$

现在的问题是，这该归到对角项那一类呢还是归到非对角项那一类呢？根据严的文章，我们发现它归为了非对角项并且增加了一个“过渡项”

$$\begin{aligned} H_{0,0} &= I \\ H_{-1,g} &= h(\sigma^+ + \sigma^-) \\ H_{0,g} &= h \\ H_{1,b} &= J(\sigma_i^z \sigma_j^z + 1) \end{aligned} \tag{4}$$

那么对角项具有：

$$\begin{aligned} \langle \uparrow | H_{0,g} | \uparrow \rangle &= h, \\ \langle \uparrow\uparrow | H_{1,b} | \uparrow\uparrow \rangle &= 2J \\ \langle \downarrow\downarrow | H_{1,b} | \downarrow\downarrow \rangle &= 2J \end{aligned} \tag{5}$$

反平行项在这里因为常数设为1而权重变为0

*AFMT Ising Model 插入/删去算符的概率推导过程见附录*

According detailed balance, we obtain:

$$\begin{aligned} P_d &= \frac{M-n+1}{\beta(Nh + 2JN_b)} \\ P_i &= \frac{\beta(Nh + 2JN_b)}{M-n} \end{aligned} \tag{6}$$

但是，对于插入算符实际上这不是最终的概率，我们在通过细致平衡计算的过程中，计算的$P_i$是“插入算符”这个动作的概率，这么说有点抽象，但换个说法就是还要加上：

1. 如果插入的是格点算符：

$$P_{site} = \frac{Nh}{Nh + 2JN_b} \tag{7}$$

2. 如果插入的是bond算符：

$$P_{bond} = \frac{2JN_b}{Nh + 2JN_b} \tag{8}$$

最终插入一个算符（根据算符类型，在这里以插入的是bond为例）

$$P = \frac{\beta(Nh + 2JN_b)}{M - n} \cdot \frac{2JN_b}{Nh + 2JN_b} \cdot \frac{1}{N_b} = \frac{2J\beta}{M - n} \tag{9}$$

反过来思考，删去算符的时候为什么没有这样的情况？我能否在细致平衡中完整地推导出这个概率，而不是只通过细致平衡推出一部分？

!我发现，我理解错这个论文在这里概率的意思了。你看他后文还提了一嘴：

> 我们先以1/N 的概率选择一个位置来插入算符,当位置选定之后,我们的算符类型也就选定了....

其实就是我们在程序中，刚判断出是单位算符，然后进行抛随机数选一个随机位置，这个操作所对应的就是$1/N_b$，这样的话，我们插入一个特定类型的算符的动作的概率实际上是：

$$P = \frac{\beta(Nh + 2JN_b)}{M - n} \cdot \frac{2JN_b}{Nh + 2JN_b} = \frac{2JN_b\beta}{M - n} \tag{10}$$

以插入的格点对角算符为例

$$P = \frac{\beta(Nh + 2JN_b)}{M - n} \cdot \frac{hN}{Nh + 2JN_b} = \frac{hN\beta}{M - n} \tag{11}$$

对角更新：

实现的是 [0,0] <-> [i,j]，这里[i,j]表示的是$\sigma_i^z\sigma_j^z$相互作用，然后考虑了一下，如果还是要考虑[0,0] <-> [i,i]这种转换

```
subroutine diagonalupdate()
    use configuration
    implicit none

    !parameters:
    integer :: i,b,op,g
    real(8) :: p
    real(8),external :: ran

  !--!*(1/17)重新写了这部分的程序，主要是更换概率发生先后的逻辑--------------!
```

```
    do i = 0,mm-1
        op = opstring(i)
        if (op==0) then !----I operator
            !now need to insert operator:
            if (apb>=ran()) then! insert bond
                if ( aprob>=dfloat(mm-nh) .or. aprob>=ran()*(mm-nh) ) then!(beta/(M-n)) *(2JN_b
+ hN) 是否要插键的概率
                !sure insert bond
                    b = min( int(ran()*nb)+1,nb ) !-----here insert a diag
                    ! print*,"b=",b
                    if (spin(bsites(1,b))==spin(bsites(2,b))) then !只有平行矩阵元起作用 !反平行自旋
权重为0
                        opstring(i) = 4*b
                        nh = nh+1

                        ! print*,"insert bond diag"
                    ! else
                    ! print*,"site:",spin(:)
                    endif
                endif
            else !else insert diag site
                if ( aprob>=dfloat(mm-nh) .or. aprob>=ran()*(mm-nh) ) then
                    g = min( int(ran()*nn)+1,nn ) !diagonal site
                    opstring(i) = 4*g - 1 !4b+a-1,a=0
                    nh = nh+1
                    ! print*,"insert site diag"
                endif
            endif !不符合插键概率，不插直接结束

        elseif(((op/=0).and.(mod(op,4)==0)) .or. (mod(op,4)==3)) then !----bond-diag or g-diag
!* 无论遇到哪种，删去的概率相同，因此写在一起
            !ran < dprob*(M-n+1), dprob*(M-n+1)>=1
            p = dprob*(mm-nh+1)
            if(p>=1.d0 .or. p>=ran()) then
                opstring(i) = 0
                nh=nh-1
                ! print*,"delete a diag"
            endif
        else
            if(mod(op,4)==2) then
                g=(op+2)/4
                spin(g)=-spin(g)
            endif
        endif
    enddo
    ! print*,"nh=",nh,"mm=",mm

end subroutine
```

(1/17 update) 截止到目前为止有哪些问题

(1/19 update) 对角算符其实还有一些问题：

我忽略了对角格点对于$|\downarrow\rangle$的作用：

$$\langle\downarrow|H_{0,g}|\downarrow\rangle = -h, \tag{12}$$

然而，在这种情况下，非对角格点算符依然：

$$\langle\downarrow|H_{-1,g}|\uparrow\rangle = h, \tag{13}$$

~~这会导致目前的对角更新需要进行什么修改呢？我先自己想想：~~

1. ~~Insert 算符的过程需要判断自旋上下，在 insert drag site 的时候需要分两步，一步是对上自旋一步是对下自旋，分别以$h$,$-h$代入概率公式。~~
2. insert 算符依旧不需要插入非对角算符，而是遇到非对角算符进行相应的翻转
3. ~~还有一种可能是我忽略了，那就是我们在这里也不希望出现负能量的对角元！因此，给对角格点项修正为~~

哦！再回看给定的哈密顿量公式，我发现我本来理解正确的又理解错了，那个对角格点项，严格意义上说不叫对角项$H = h\sigma_i^z$，而是叫做在某个格点上增加的常数项！也就是说，无论我们遇到的是上自旋还是下自旋，产生的矩阵元都是$h$.

> 我看严正的代码中遇到了一个疑问，那就是为何会增加了一个叫做“自旋向上点的算符权重” ——应该是处理的纵场，导致向上和向下有不同能量，这和我们这里的模型不一样
>
> 以及它的bond算符居然有两种类型，还有一个耦合J2，但是没看到J1，大胆猜测他处理的模型是各向异性，就是有J1，J2，但是J1=1所以没写出来。暂时和我处理的模型不造成影响。（他的J2像是在处理x轴方向的耦合强度，而J1是y方向的耦合）

非对角算符的更新我不会写，还不知道是个怎样的思路，目前的说法有：

> 1. 从格点算符出发，遇到对角键算符一进三出，遇到格点算符（不一定是起点那个）就终止
> 2. 据说这样，实现的是终点的格点算符的相互更新 H(0,a) <-> H(-1,a)，而不改变配置的权重，不改变簇内的格点算符类型，不改变对角键算符，因为一个对角键算符都属于同一个簇

> 非对角线更新（13）可以有效地进行，如果首先将 SL 分割为每个站点 i 的独立子序列。子序列 i 仅包含自旋翻转运算符 [i, 0] 和常数 [i, i]。它们在 SL 中的位置也被存储，以便在更新后重新组合这些子序列时使用。由 Ising 运算符 [i, j] 或 [j, i]（对于任何 j）对站点 i 进行的修改的约束可以存储为标志，指示在相邻子序列运算符之间是否存在其中一个或多个这些运算符。更新子序列就是从子序列中随机选择两个非受约束的相邻运算符，如果两个运算符相同，则进行替换（13）。如果它们不

> 同，它们可以被置换。对于每个子序列，进行与子序列长度成比例的这种配对更新的次数，然后将它们重新组合成一个新的 SL。
>
> from Sandvik

从Sandvik中分析，

1. 似乎需要两个列表，对于格点算符也需要一个列表，但是列表需要两个吗？需要分开储存吗？——只需要一个列表！
2. 什么叫从子序列中随机选择两个非受约束的相邻运算符，受约束指啥，相邻是什么相邻？

为了构建和翻转一个量子簇，首先随机选择 n 个顶点中的一个顶点的一个腿，然后翻转相应的自旋。根据顶点的类型，采取不同的操作，如图4所示的示例。指向顶点的箭头表示进入腿。在Ising顶点的情况下，所有四个自旋都被翻转，簇构建过程从所有的腿中分支出，如从顶点指出的箭头所示。

这是什么意思？我知道它可能是为了要保证它不变成其他算符，但是如何实现呢？

我现在猜测关键点在于“从一个格点算符出发，把属于这个格点算符的簇全部找完”

1/19 目前写的vertexlist连接的程序没有问题，有问题在于连接、翻转，似乎产生的更新是无效的。

YZ的代码

构造vertexlist部分

```
frstspinop(:)=-1
lastspinop(:)=-1

 do v0=0,4*mm-1,4
    op=opstring(v0/4)

    if ((op/=0).and.(op/2<=nb0+nb1)) then!有算符存在（或者说非单位算符）,bond-op
       b=op/2
       s1=bsites(1,b)
       s2=bsites(2,b)
       v1=lastspinop(s1)!s1格点连上来最后的虚时间位置的v
       v2=lastspinop(s2)
       if (v1/=-1) then!正常情况下，s1对应连在虚轴最上端的vertex格点
          vertexlist(v1)=v0
          vertexlist(v0)=v1
       else!v1=-1，就是说s1对应虚轴上的点还没有过vertex
          frstspinop(s1)=v0
```

```
        endif
        if (v2/=-1) then!同v1
           vertexlist(v2)=v0+1
           vertexlist(v0+1)=v2
        else
           frstspinop(s2)=v0+1
        endif
        lastspinop(s1)=v0+2
        lastspinop(s2)=v0+3
     elseif(op==0) then!没算符的话，list=0
        vertexlist(v0:v0+3)=-2 !空算符标记为-2
     else!site-op
        s1=op/2-nb0-nb1
        v1=lastspinop(s1)
        if (v1/=-1) then!正常情况下，s1对应连在虚轴最上端的vertex格点
           vertexlist(v1)=v0
           vertexlist(v0)=v1
        else!v1=-1，就是说s1对应虚轴上的点还没有过vertex
           frstspinop(s1)=v0
        endif
        lastspinop(s1)=v0+2

        vertexlist(v0+1)=-2
        vertexlist(v0+3)=-2
     endif
  enddo
```

这不是和我构建vertexlist的逻辑一样吗，只是我标记为0，他标记为-2，重点是，遇到site算符只连接s1，也就是只处理一边的腿，另一边的腿不连接。

其次，可能逻辑上有些许的不一样，我是先查是不是空算符，其次再判断是不是bond或者diag-site，最后都不符合，再在其中寻找符合off-diag的概率

他的是先找符合bond（有两种bond）的概率，然后再查是不是空算符，最后都不符合，只能是site算符

OK，先假设逻辑不影响大问题

我们再来看loop更新的部分。

YZ的代码：

```
top=-1
 do v0=0,4*mm-1,2
    if (vertexlist(v0)<0) cycle
    !这个条件，把空算符和访问过(vertexlist(x)=-2 or -1)的算符给略过
    zz=-2
    if (ran()>0.5) zz=-1
    call  input(v0)
    !i=0
    do
        call makeloop()
        if (top==-1) exit
    enddo
 enddo

 ! For input()
```

```
 subroutine input(x)
 use configuration; implicit none
 integer :: x

 top=top+1
 stack(top)=x !很普通添加到堆栈里的程序

 end subroutine input


 ! For makeloop()
 subroutine makeloop()
!-------------------------!
 use configuration;implicit none
 integer :: v1,v2,v3,v4,v,ir,i,output

 v1=output()
 v=vertexlist(v1)
 if ((v>=0).and.(vertexlist(v)/=zz)) call input(v)

 vertexlist(v1)=zz
 i=v1/4

if(zz==-1) then!!!!!!!!!!!add
 if  ((opstring(i)/2)>nb0+nb1) then
    opstring(v1/4)=ieor(opstring(v1/4),1)
    !return
 endif!主要是说明v1已经是一个site-op，到此截止。
endif
 !print*,'说明v1在一个bond-op上'
     if (((opstring(i)/2)<=nb0+nb1).and.(opstring(i)/=0))  then
         v2=(ieor(v1,1))
         v3=(ir(v1))
         v4=(ieor(v3,1))
         if (vertexlist(v2)>0) call input(v2)
         if (vertexlist(v3)>0) call input(v3)
         if (vertexlist(v4)>0) call input(v4)
     endif


!-------------------------!
 end subroutine makeloop
```

其中重点我还是想看一下，在连接过程如何处理的算符更新

1. 如何处理已经访问过的算符?

   ```
   vertexlist(v1)=zz
   !outside:

   zz=-2
       if (ran()>0.5) zz=-1
       ...
   ```

   这自动为符合1/2概率翻转和没翻转的loop做了区分

2. 重头戏还得是 `makeloop()`
   - 由于这是一个内部函数，因此要先把堆栈中的元素调出来:

```
v1=output()
!output() 也只是一个普通的调出堆栈元素的函数
integer function output()
use configuration; implicit none

output=stack(top)
top=top-1

end function output
```

- 先实现跳跃
- 对该跳跃格点进行判断，满足条件再加入到堆栈：

```
v1=output()
v=vertexlist(v1)
if ((v>=0).and.(vertexlist(v)/=zz)) call input(v)
```

符合这种条件的是1. v不能是-2，也就是不能是空算符和访问过的算符；2. 此时外部`zz=-2`而符合翻转条件的是`zz=-1`，那么对于v0这一层，如论哪种，此条件找到的都是v作为算符的情况

- 由于已经对跳跃的格点进行了处理，因此对跳跃路径标记为访问过（过河拆桥）：

```
vertexlist(v1)=zz
```

- 符合1/2的翻转概率，算符change是怎么处理的：

```
! i get from:
i=v1/4

if(zz==-1) then !以v0为起始点的该层符合1/2翻转概率
 if  ((opstring(i)/2)>nb0+nb1) then
    opstring(v1/4)=ieor(opstring(v1/4),1)
    !return
 endif!主要是说明v1已经是一个site-op，到此截止。
endif
```

这个程序只处理了v1这个腿所在的算符，考虑三种情况：

1. v = vertexlist(v1)是回到了同一个site算符，那么 v1时翻转一次，v时翻转一次，对应一条路径上的单格点对角算符不变
2. v = vertexlist(v1)是同一条路径上的另一个格点算符，那么v1时翻转一次，

- 对于“遇到site算符就终止”这个条件，是如何处理的：
- 遇到bond算符的处理：

```
if (((opstring(i)/2)<=nb0+nb1).and.(opstring(i)/=0))  then
          v2=(ieor(v1,1))
          v3=(ir(v1))
          v4=(ieor(v3,1))
          if (vertexlist(v2)>0) call input(v2)
          if (vertexlist(v3)>0) call input(v3)
          if (vertexlist(v4)>0) call input(v4)
endif
```

和我的思路一样，遇到三条腿检查一下访问过没，没就加进堆栈里去。

这样的话，其实他对于加入堆栈的唯一要求只有检查有没有被访问更改过。

总结一下，

当我们从堆栈top中调出元素之后，要对该元素所在的算符进行判断，因为尽管你知道v0挑出来出发的必然是site算符，但你不能保证后续从stack中调出来的都是site算符。

我之前的思路是，我已知我是从site算符出发，那么初始的site必然翻转一次，在添加进stack之前。而无论从stack中抽出来的元素是什么，遇到site算符，都要改变末尾的site算符一次。（我觉得这样也行啊）

1. 如果正确处理算符更新之后，代码算出来的结果还是不对，该怎么debug?

   (1/22)虽然这个问题遇到了，但还没写，今天调出来一个正确的版本了！！！好耶！！

(1/22) Loopupadate(clusterupdate):

```
subroutine loopupdate()
    use Configuration
    implicit none

    !parameters:
    integer :: i,n,l,b,op,s1,s2,v0,v1,v2,top,a,g
    real(8),external :: ran
    integer,external :: ir
    integer :: stack(0:8*mm-1)

    frstspinop = -1
    lastspinop = -1

    vertexlist(:) = -2 !先进行初始化为-2。之前我选择的是0，但后来想了想，0和update里面一些判断条件可能会产生混淆，最后还是选择了-2

    do v0=0,4*mm-1,4
        op=opstring(v0/4)
        if( op/=0 ) then

            if (mod(op,4)==0) then !-diag bond

                b = op/4

                s1=bsites(1,b) !---get i
                s2=bsites(2,b) !---get j
```

```
            v1=lastspinop(s1) !use imformation of lastspin(i)
            v2=lastspinop(s2) !lastspin(j)

            if(v1/=-1) then

                vertexlist(v1)=v0
                vertexlist(v0)=v1
            else
                frstspinop(s1)=v0
            endif

            if(v2/=-1) then
                vertexlist(v2)=v0+1
                vertexlist(v0+1)=v2
            else
                frstspinop(s2)=v0+1
            endif

            lastspinop(s1)=v0+2
            lastspinop(s2)=v0+3

        elseif (mod(op,4)==3) then !---diag site
            g = (op+1)/4

            s1 = bsites(1,g)
            v1=lastspinop(s1) !use imformation of lastspin(i)
            !don't use lastspin(s2)!

            if(v1/=-1) then
                vertexlist(v1)=v0
                vertexlist(v0)=v1
            else
                frstspinop(s1)=v0
            endif

            vertexlist(v0+1)=-2
            vertexlist(v0+3)=-2

            lastspinop(s1)=v0+2 !only connect s1

        else!-----------------------off-diag site
            if (mod(op,4)==2) then !---off-diag site

                g = (op+2)/4
                s1 = bsites(1,g)
                v1=lastspinop(s1) !use imformation of lastspin(i)

            if(v1/=-1) then
                vertexlist(v1)=v0
                vertexlist(v0)=v1
            else
                frstspinop(s1)=v0
            endif
                vertexlist(v0+1)=-2
                vertexlist(v0+3)=-2
                lastspinop(s1)=v0+2

            endif !for off-diag site
        endif !for check mod(op,4)
    else !----------I
        vertexlist(v0:v0+3)=-2 !----no vertex
    endif
enddo
```

```
    !periodic boundary condition
    do s1=1,nn
        v1=frstspinop(s1)
        if ((v1/=-1)) then
            v2=lastspinop(s1)
            vertexlist(v2)=v1
            vertexlist(v1)=v2
        endif
    enddo

    !----------flip loop----------------!
    do v0=0,4*mm-1,2 !*必须每个leg遍历那样来找

        if((vertexlist(v0)<0)) cycle !空算符or被访问过 则跳过

        stack(:) = 0
        top = 0
        stack(top)=v0

        !对于符合1/2概率翻转的，标记为-1，同时对site算符进行算符翻转；不符合概率翻转的，标记为-2。
        if (ran()<0.5d0) then
            do while(top>=0)
                v1 = stack(top) !从堆栈中调出元素
                top = top-1

                if (vertexlist(v1)<0) cycle !如果访问过，则进入到下一条腿的执行
                v2 = vertexlist(v1)
                if ((v2>=0).and.(vertexlist(v2)>=0)) then !由于处理的是v1,因此，符合条件的v2添加进堆
栈，留作下一次处理
                    top = top + 1
                    stack(top) = v2
                endif

                vertexlist(v1)=-1
                i = opstring(v1/4)

                !以下是对v1进行的一系列判断:

                if (mod(i,4)==3 .or. mod(i,4)==2) then !如果是site算符
                    a=ieor(mod(i,4),1)!
                    opstring(v1/4) = opstring(v1/4)-mod(opstring(v1/4),4)+a
                endif

                if (mod(i,4)==0 .and. i/=0) then !如果是bond算符
                    if(vertexlist(ir(v1))>=0) then
                    !'处理直进'
                    top = top + 1
                    stack(top) = ir(v1)
                    endif
                    if(vertexlist(ieor(v1,1))>=0) then
                    !'处理横腿: '
                    top = top + 1
                    stack(top) = ieor(v1,1)
                    endif
                    if(vertexlist(ieor(ir(v1),1))>=0) then
                    !'处理斜穿'
                    top = top + 1
                    stack(top) = ieor(ir(v1),1)
                    endif

                endif
```

```
            enddo

        else
            do while(top>=0)
                v1 = stack(top)
                top = top-1

                if (vertexlist(v1)<0) cycle
                v2 = vertexlist(v1)
                if ((v2>=0).and.(vertexlist(v2)>=0)) then
                    top = top + 1
                    stack(top) = v2
                endif

                vertexlist(v1)=-2
                i = opstring(v1/4)

                if (mod(i,4)==0 .and. i/=0) then
                    if(vertexlist(ir(v1))>=0) then
                    !'处理直进'
                    top = top + 1
                    stack(top) = ir(v1)
                    endif
                    if(vertexlist(ieor(v1,1))>=0) then
                    !'处理横腿: '
                    top = top + 1
                    stack(top) = ieor(v1,1)
                    endif
                    if(vertexlist(ieor(ir(v1),1))>=0) then
                    !'处理斜穿'
                    top = top + 1
                    stack(top) = ieor(ir(v1),1)
                    endif

                endif
            enddo
        endif
    enddo

  !对标记为-1的路径上的自旋翻转，对标记为不是-1的自旋实现自由1/2概率翻转
    do i=1,nn
        if (frstspinop(i)/=-1) then
            if (vertexlist(frstspinop(i))==-1) spin(i)=-spin(i)
        else
            if (ran()<0.5) spin(i)=-spin(i)
        endif
    enddo

end subroutine
```