FDTD法によるTDGLシミュレーション

リンク変数法によるシミュレーション，および
オブジェクト指向プログラミングによる込み入った数式の実装

有明高専 電子情報工学科　松野哲也

応用物理学会２０１５秋（９月１３日（日）12:00-13:30）
インフォーマルミーティング 「量子化磁束動力学シミュレーション研究会 」

FDTD法によるTDGLシミュレーション

- FDTD：時間領域有限差分法
  （ 偏微分方程式 → 時間発展差分方程式 ）

- FEM ：有限要素法
  （ 偏微分方程式 → 変分問題 → 線形方程式 ）

TDGL方程式の導出

$$\epsilon_{GL} = \epsilon_K + \epsilon_P + \epsilon_B,$$

$$\begin{aligned}
\epsilon_K &= \frac{1}{2}|\boldsymbol{D}\psi|^2, \quad \boldsymbol{D} \equiv \nabla - \mathrm{i}g\boldsymbol{A}, \\
\epsilon_P &= \frac{\eta}{2}(|\psi|^2 - 1)^2, \\
\epsilon_B &= \frac{1}{2}|\nabla \times \boldsymbol{A}|^2.
\end{aligned}$$

$$
\begin{aligned}
\tau\left(\frac{\partial}{\partial t}+\mathrm{i}g\phi\right)\psi &= -\frac{\delta\epsilon_{GL}}{\delta\overline{\psi}},\\
\frac{\partial \boldsymbol{A}}{\partial t}+\nabla\phi &= -\frac{\delta\epsilon_{GL}}{\delta\boldsymbol{A}},
\end{aligned}
$$

TDGL方程式
時間依存ギンツブルグランダウ方程式
(Time dependent Ginzburg-Landau equation)

$$
\begin{aligned}
\tau\left(\frac{\partial}{\partial t}+\mathrm{i}g\phi\right)\psi &= \frac{1}{2}\boldsymbol{D}^2\psi+\eta(1-|\psi|^2)\psi,\\
\frac{\partial \boldsymbol{A}}{\partial t}+\nabla\phi &= g\mathrm{Im}\left[\overline{\psi}\boldsymbol{D}\psi\right]-\nabla\times\nabla\times\boldsymbol{A}.
\end{aligned}
$$

ゲージ対称性

$$\boldsymbol{A} \longrightarrow \boldsymbol{A} + \nabla\chi, \quad \phi \longrightarrow \phi - \frac{\partial\chi}{\partial t}, \quad \psi \longrightarrow \psi\exp(\mathrm{i}g\chi).$$

ϕ-zero gauge

$$\begin{aligned} \tau\frac{\partial\psi}{\partial t} &= \frac{1}{2}\boldsymbol{D}^2\psi + \eta(1-|\psi|^2)\psi, \\ \frac{\partial\boldsymbol{A}}{\partial t} &= g\mathrm{Im}\left[\overline{\psi}\boldsymbol{D}\psi\right] - \nabla\times\nabla\times\boldsymbol{A}. \end{aligned}$$

なぜ φゼロゲージ，なぜリンク変数法か？

ϕ-zero gauge

FDTD法と相性が良い.

陽的数値積分を組める.

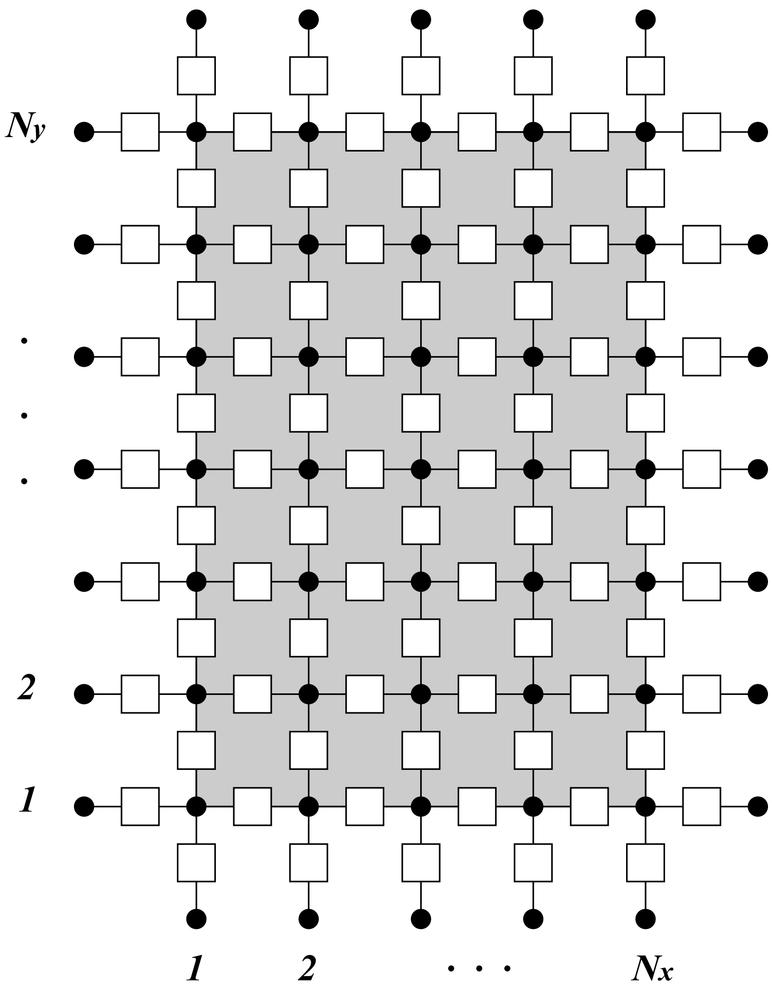
並列化が容易

高速化

flux flow シミュレーションを
行うにはリンク変数法が必須.

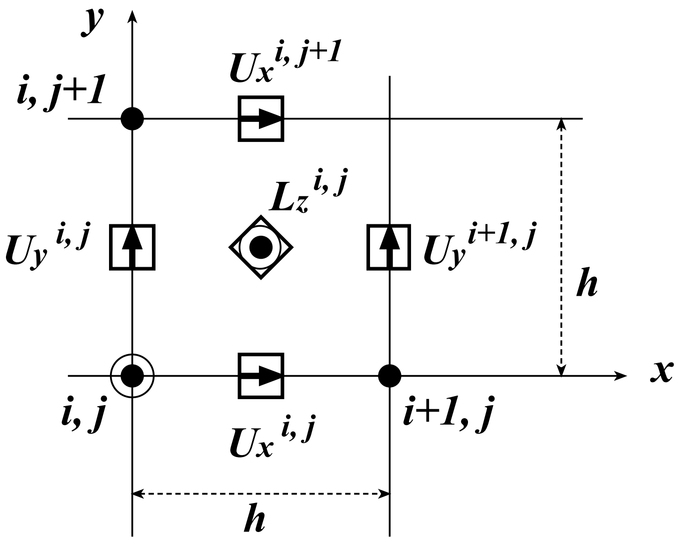
E 一定は A の時間微分が一定
であることを意味するから.
つまり，$t \to \infty$ で A は有界ではない.

リンク変数法



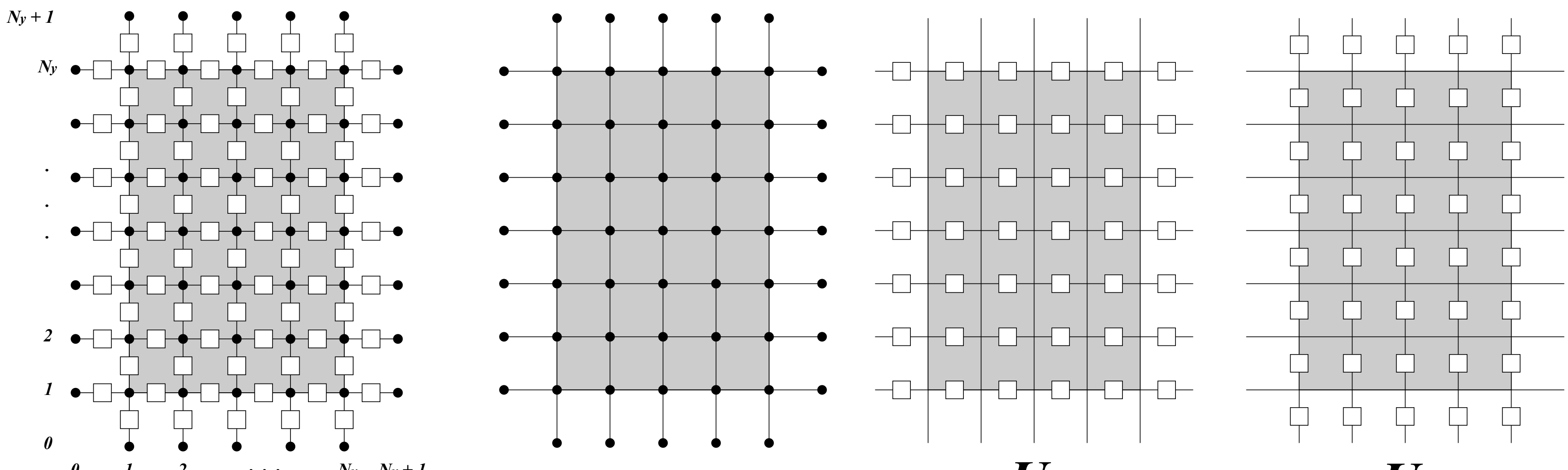
格子点の上で
オーダーパラメータを定義

格子点と格子点の間で
ベクトルポテンシャルを定義



リンク変数:

$$
\begin{aligned}
U_x^{i,j,k} &= \exp(-\mathrm{i}hgA_x^{i,j,k}),\\
U_y^{i,j,k} &= \exp(-\mathrm{i}hgA_y^{i,j,k}),\\
U_z^{i,j,k} &= \exp(-\mathrm{i}hgA_z^{i,j,k}),
\end{aligned}
$$



ψ U_x U_y

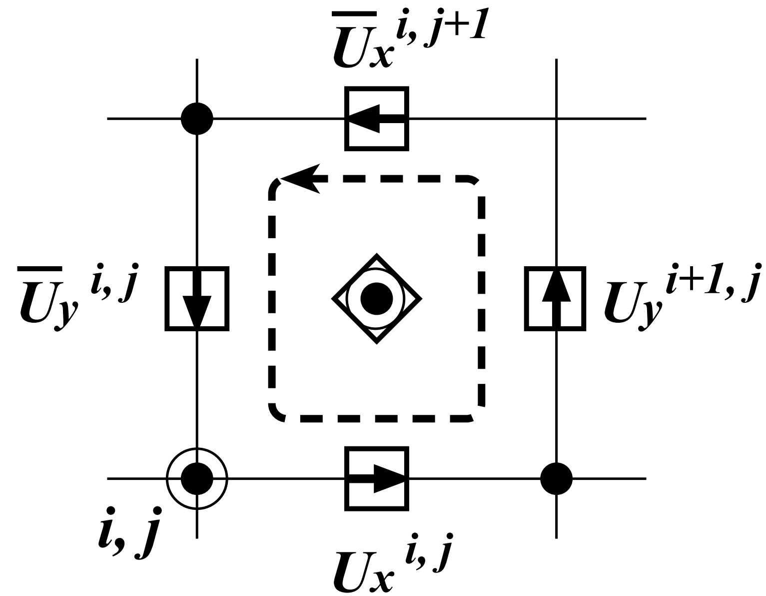
共変微分演算子（２次元）

$$D_x\psi \quad \longrightarrow \quad \frac{1}{h}(U_x^{i,j}\psi^{i+1,j} - \psi^{i,j})$$

$$D_x^2\psi \quad \longrightarrow$$

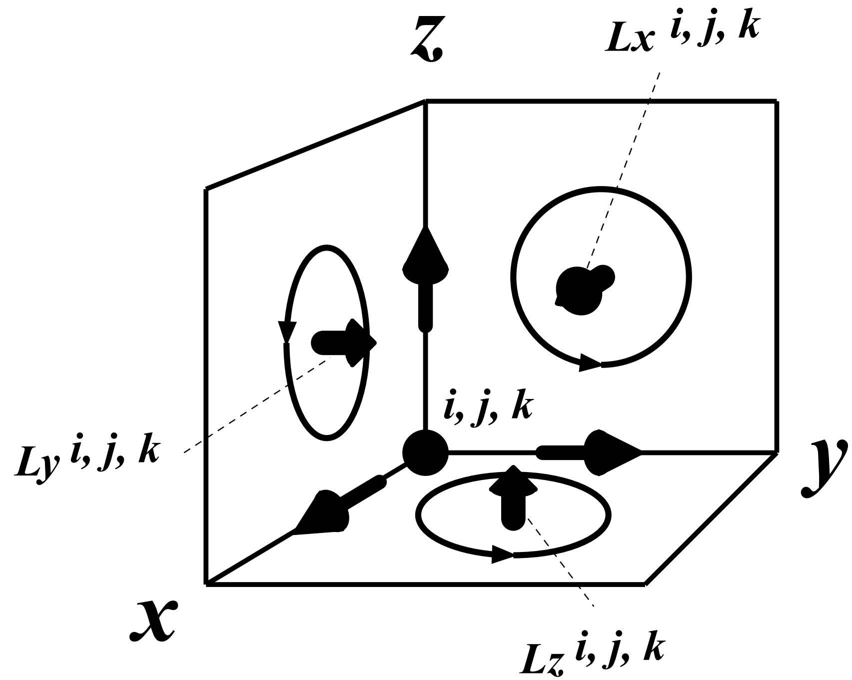
$$\frac{1}{h^2}(U_x^{i,j}\psi^{i+1,j} + \overline{U}_x^{i-1,j}\psi^{i-1,j} - 2\psi^{i,j})$$

リンク変数の積は経路積分に対応する.



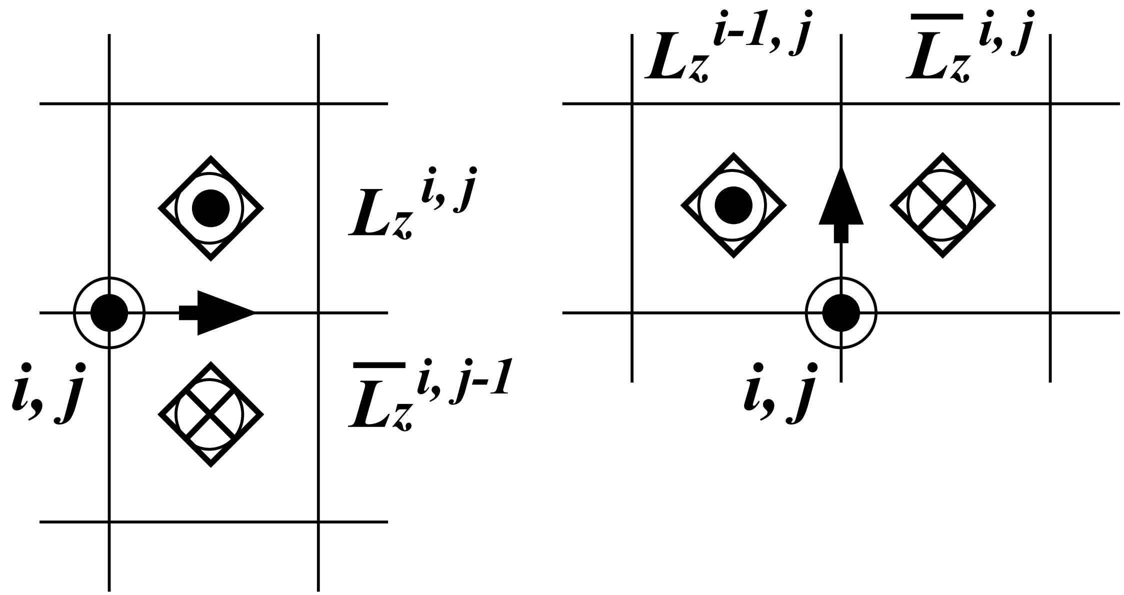
$$
\begin{aligned}
L_z^{i,j} &\equiv U_x^{i,j} U_y^{i+1,j} \overline{U}_x^{i,j+1} \overline{U}_y^{i,j} \\
&= \exp[-\mathrm{i} h g (A_y^{i+1,j} - A_y^{i,j} - A_x^{i,j+1} + A_x^{i,j})] \\
&\simeq \exp[-\mathrm{i} h^2 g (\nabla \times \boldsymbol{A})_z^{i,j}],
\end{aligned}
$$

3次元では



$$
\begin{aligned}
L_x^{i,j,k} &= U_y^{i,j,k} U_z^{i,j+1,k} \overline{U}_y^{i,j,k+1} \overline{U}_z^{i,j,k}, \\
L_y^{i,j,k} &= U_z^{i,j,k} U_x^{i,j,k+1} \overline{U}_z^{i+1,j,k} \overline{U}_x^{i,j,k}, \\
L_z^{i,j,k} &= U_x^{i,j,k} U_y^{i+1,j,k} \overline{U}_x^{i,j+1,k} \overline{U}_y^{i,j,k}.
\end{aligned}
$$

２次元



$$L_z^{i,j}\overline{L}_z^{i,j-1} \simeq \exp[-\mathrm{i}h^3 g(\nabla\times\nabla\times\boldsymbol{A})_x^{i,j}],$$

$$\overline{L}_z^{i,j}L_z^{i-1,j} \simeq \exp[-\mathrm{i}h^3 g(\nabla\times\nabla\times\boldsymbol{A})_y^{i,j}],$$

$$
\begin{aligned}
\tau \frac{\partial \psi}{\partial t} &= \frac{1}{2}\boldsymbol{D}^2\psi + \eta(1-|\psi|^2)\psi, \\
\frac{\partial \boldsymbol{A}}{\partial t} &= g\mathrm{Im}\left[\overline{\psi}\boldsymbol{D}\psi\right] - \nabla\times\nabla\times\boldsymbol{A}.
\end{aligned}
$$

Semi-discretized equation (2次元)

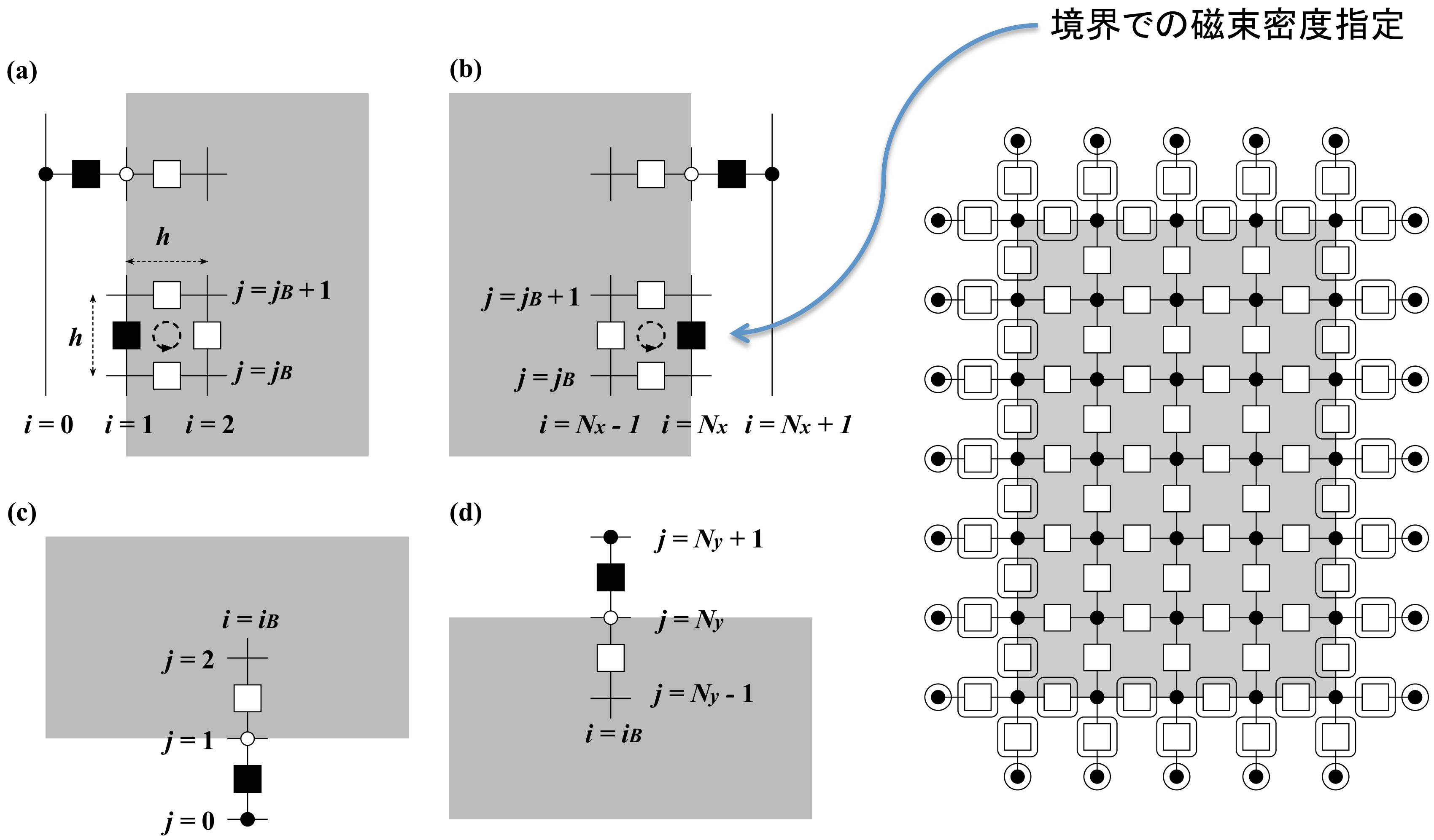
$$
\begin{aligned}
\tau \frac{\partial \psi^{i,j}}{\partial t} &= \frac{1}{2h^2}(U_x^{i,j}\psi^{i+1,j} + \overline{U}_x^{i-1,j}\psi^{i-1,j} + U_y^{i,j}\psi^{i,j+1} + \overline{U}_y^{i,j-1}\psi^{i,j-1} - 4\psi^{i,j}) \\
&\quad + \eta(1-|\psi^{i,j}|^2)\psi^{i,j}, \\
\frac{\partial U_x^{i,j}}{\partial t} &= -\mathrm{i}g\mathrm{Im}[\overline{\psi}^{i,j}U_x^{i,j}\psi^{i+1,j}]U_x^{i,j} - \frac{1}{h^2}(\overline{L}_z^{i,j-1}L_z^{i,j} - 1)U_x^{i,j}, \\
\frac{\partial U_y^{i,j}}{\partial t} &= -\mathrm{i}g\mathrm{Im}[\overline{\psi}^{i,j}U_y^{i,j}\psi^{i,j+1}]U_y^{i,j} - \frac{1}{h^2}(\overline{L}_z^{i,j}L_z^{i-1,j} - 1)U_y^{i,j},
\end{aligned}
$$

Semi-discretized equation (3次元)

$$
\begin{aligned}
\tau \frac{\partial \psi^{i,j,k}}{\partial t} &= \frac{1}{2h^2}\big(U_x^{i,j,k}\psi^{i+1,j,k} + \overline{U}_x^{i-1,j,k}\psi^{i-1,j,k} + U_y^{i,j,k}\psi^{i,j+1,k} + \overline{U}_y^{i,j-1,k}\psi^{i,j-1,k} \\
&\quad + U_z^{i,j,k}\psi^{i,j,k+1} + \overline{U}_z^{i,j,k-1}\psi^{i,j,k-1} - 6\psi^{i,j,k}\big) + \eta(1 - |\psi^{i,j,k}|^2)\psi^{i,j,k}.
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial U_x^{i,j,k}}{\partial t} &= -\mathrm{i}g\mathrm{Im}[\overline{\psi}^{i,j,k}U_x^{i,j,k}\psi^{i+1,j,k}] - \frac{1}{h^2}(\overline{L}_y^{i,j,k}L_y^{i,j,k-1}L_z^{i,j,k}\overline{L}_z^{i,j-1,k} - 1), \\
\frac{\partial U_y^{i,j,k}}{\partial t} &= -\mathrm{i}g\mathrm{Im}[\overline{\psi}^{i,j,k}U_y^{i,j,k}\psi^{i,j+1,k}] - \frac{1}{h^2}(\overline{L}_z^{i,j,k}L_z^{i-1,j,k}L_x^{i,j,k}\overline{L}_z^{i,j,k-1} - 1), \\
\frac{\partial U_z^{i,j,k}}{\partial t} &= -\mathrm{i}g\mathrm{Im}[\overline{\psi}^{i,j,k}U_z^{i,j,k}\psi^{i,j,k+1}] - \frac{1}{h^2}(\overline{L}_x^{i,j,k}L_x^{i,j-1,k}L_y^{i,j,k}\overline{L}_y^{i-1,j,k} - 1),
\end{aligned}
$$

境界条件



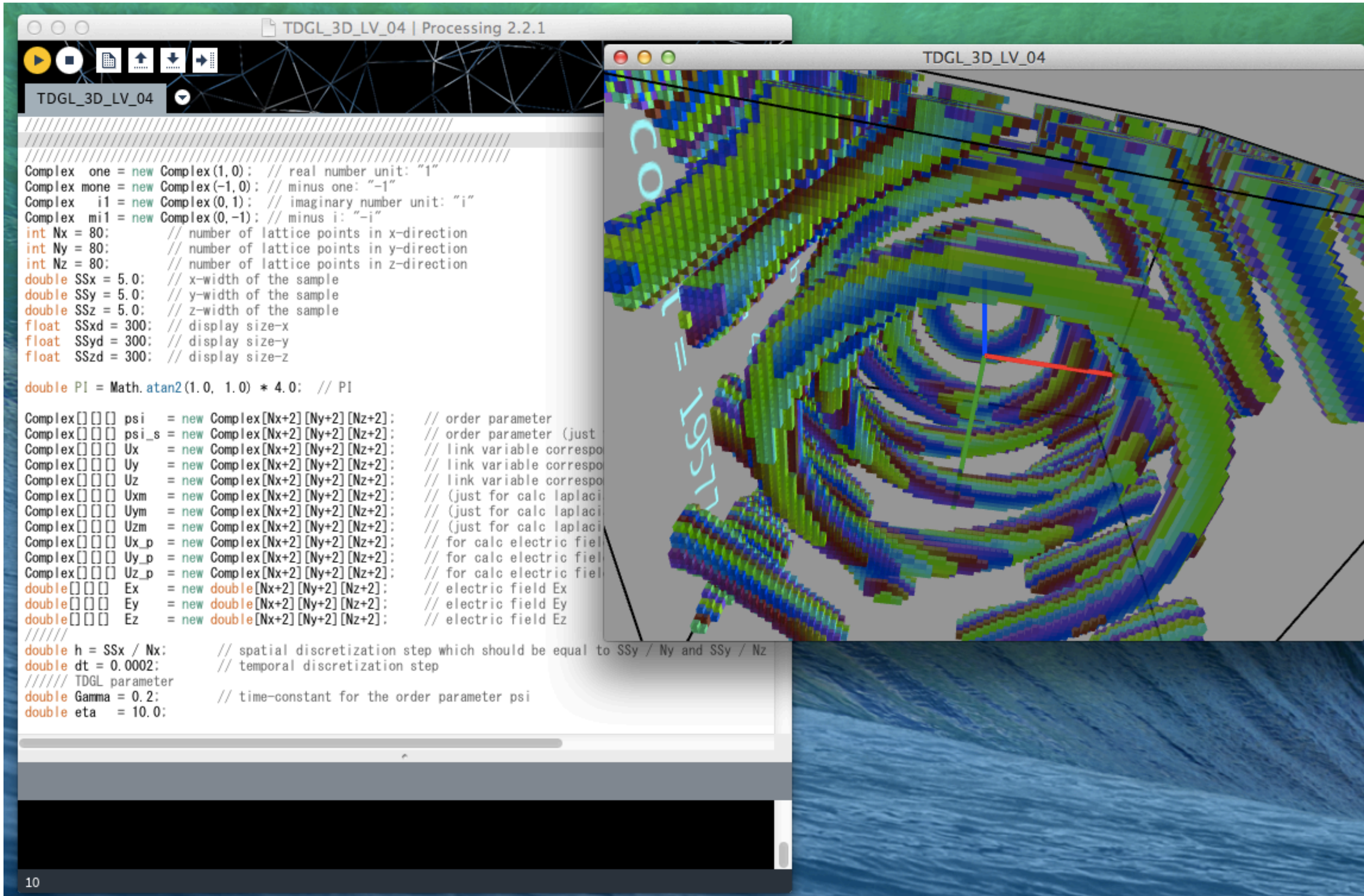
境界での磁束密度指定
(a)
h
h
j = jB + 1
j = jB
i = 0
i = 1
i = 2
(b)
j = jB + 1
j = jB
i = Nx - 1
i = Nx
i = Nx + 1
(c)
i = iB
j = 2
j = 1
j = 0
(d)
j = Ny + 1
j = Ny
j = Ny - 1
i = iB

数値シミュレーション

- 動画：２次元
  横磁界
- 動画：３次元
  横磁界，縦磁界，リング
- リアルタイムデモ
  ２次元(40x80)，３次元(30x30x30)

※ order parameter 値に応じた色で可視化

コーディング（実装）


TDGL_3D_LV_04 | Processing 2.2.1
TDGL_3D_LV_04
TDGL_3D_LV_04
Complex one = new Complex(1,0); // real number unit: "1"
Complex mone = new Complex(-1,0); // minus one: "-1"
Complex   i1 = new Complex(0,1); // imaginary number unit: "i"
Complex mi1 = new Complex(0,-1); // minus i: "-i"
int Nx = 80;   // number of lattice points in x-direction
int Ny = 80;   // number of lattice points in y-direction
int Nz = 80;   // number of lattice points in z-direction
double SSx = 5.0;   // x-width of the sample
double SSy = 5.0;   // y-width of the sample
double SSz = 5.0;   // z-width of the sample
float SSxd = 300;   // display size-x
float SSyd = 300;   // display size-y
float SSzd = 300;   // display size-z
double PI = Math.atan2(1.0, 1.0) * 4.0; // PI
Complex[][][] psi   = new Complex[Nx+2][Ny+2][Nz+2];   // order parameter
Complex[][][] psi_s = new Complex[Nx+2][Ny+2][Nz+2];   // order parameter (just
Complex[][][] Ux   = new Complex[Nx+2][Ny+2][Nz+2];   // link variable correspo
Complex[][][] Uy   = new Complex[Nx+2][Ny+2][Nz+2];   // link variable correspo
Complex[][][] Uz   = new Complex[Nx+2][Ny+2][Nz+2];   // link variable correspo
Complex[][][] Uxm   = new Complex[Nx+2][Ny+2][Nz+2];   // (just for calc laplaci
Complex[][][] Uym   = new Complex[Nx+2][Ny+2][Nz+2];   // (just for calc laplaci
Complex[][][] Uzm   = new Complex[Nx+2][Ny+2][Nz+2];   // (just for calc laplaci
Complex[][][] Ux_p = new Complex[Nx+2][Ny+2][Nz+2];   // for calc electric fiel
Complex[][][] Uy_p = new Complex[Nx+2][Ny+2][Nz+2];   // for calc electric fiel
Complex[][][] Uz_p = new Complex[Nx+2][Ny+2][Nz+2];   // for calc electric fiel
double[][][] Ex   = new double[Nx+2][Ny+2][Nz+2];   // electric field Ex
double[][][] Ey   = new double[Nx+2][Ny+2][Nz+2];   // electric field Ey
double[][][] Ez   = new double[Nx+2][Ny+2][Nz+2];   // electric field Ez
//////
double h = SSx / Nx;   // spatial discretization step which should be equal to SSy / Ny and SSy / Nz
double dt = 0.0002;   // temporal discretization step
////// TDGL parameter
double Gamma = 0.2;   // time-constant for the order parameter psi
double eta   = 10.0;
10

表現に類似性をもたせる

```
a = a + b・c・d + e・f
```

```
a.equal( a.plus( b.mul(c).mul(d) ).plus( e.mul(f) );
```

```
Complex plus( Complex z ){
   return new Complex( x + z.x, y + z.y );
}
```

```
class Complex{
  //// x: real part
  //// y: imaginary part
  double x, y;
  Complex(double x0, double y0){
    x = x0; // real part
    y = y0; // imaginary part
  }
  Complex(Complex z){
    x = z.x;
    y = z.y;
  }
  void setPolar(double R, double theta){
    x = R * Math.cos(theta);
    y = R * Math.sin(theta);
  }
  void equal(Complex z){
    x = z.x;
    y = z.y;
  }
  void equal(double x0, double y0){
    x = x0;
    y = y0;
  }
  Complex plus(Complex z){
    return new Complex(x + z.x, y + z.y);
  }
  Complex minus(Complex z){
    return new Complex(x - z.x, y - z.y);
  }
  Complex mul(double a){
    return new Complex(a*x, a*y);
  }
  Complex mul(Complex z){
    double x1 = z.x;
    double y1 = z.y;
    double x2 = x;
    double y2 = y;
    return new Complex(x1*x2 - y1*y2, x1*y2 + y1*x2);
  }
  Complex div(double a){
    return new Complex(x/a, y/a);
  }
  Complex conj(){
    return new Complex(x, -y);
  }
}
```

「複素数クラス」を定義

```
Complex mul(Complex z){
   double x1 = z.x;
   double y1 = z.y;
   double x2 = x;
   double y2 = y;
   return new Complex(x1*x2 - y1*y2, x1*y2 + y1*x2);
 }
```

$$
\begin{aligned}
L_x^{i,j,k} &= U_y^{i,j,k} U_z^{i,j+1,k} \overline{U}_y^{i,j,k+1} \overline{U}_z^{i,j,k}, \\
L_y^{i,j,k} &= U_z^{i,j,k} U_x^{i,j,k+1} \overline{U}_z^{i+1,j,k} \overline{U}_x^{i,j,k}, \\
L_z^{i,j,k} &= U_x^{i,j,k} U_y^{i+1,j,k} \overline{U}_x^{i,j+1,k} \overline{U}_y^{i,j,k}.
\end{aligned}
$$

```
Complex Lz(Complex[][][] Ux, Complex[][][] Uy, Complex[][][] Uz, int i, int j, int k){
  return Ux[i][j][k].mul(Uy[i+1][j][k]).mul(Ux[i][j+1][k].conj()).mul(Uy[i][j][k].conj());
}
```

$$
\begin{aligned}
\tau\frac{\partial\psi^{i,j,k}}{\partial t} \;=\;& \frac{1}{2h^2}(U_x^{i,j,k}\psi^{i+1,j,k}+\overline{U}_x^{i-1,j,k}\psi^{i-1,j,k}+U_y^{i,j,k}\psi^{i,j+1,k}+\overline{U}_y^{i,j-1,k}\psi^{i,j-1,k} \\
&+U_z^{i,j,k}\psi^{i,j,k+1}+\overline{U}_z^{i,j,k-1}\psi^{i,j,k-1}-6\psi^{i,j,k})+\eta(1-|\psi^{i,j,k}|^2)\psi^{i,j,k}.
\end{aligned}
$$

```
//////////
void do_Laplacian(Complex[][][] psi, Complex[][][] Ux, Complex[][][] Uy, Complex[][][] Uz, double dt, double D){
  for(int k = 1; k <= Nz; k++){
    for(int j = 1; j <= Ny; j++){
      for(int i = 1; i <= Nx; i++){
        psi_s[i][j][k].equal(psi[i][j][k].plus(
                 ((psi[i+1][j][k].mul(Ux[i][j][k])).plus(psi[i-1][j][k].mul(Ux[i-1][j][k].conj()))
             .plus(psi[i][j+1][k].mul(Uy[i][j][k])).plus(psi[i][j-1][k].mul(Uy[i][j-1][k].conj()))
             .plus(psi[i][j][k+1].mul(Uz[i][j][k])).plus(psi[i][j][k-1].mul(Uz[i][j][k-1].conj()))
             .minus(psi[i][j][k].mul(6.0) ) )
             .mul(dt *  D / (h * h) )
        )
        );
      }
    }
  }
  for(int k = 1; k <= Nz; k++){
    for(int j = 1; j <= Ny; j++){
      for(int i = 1; i <= Nx; i++){
        psi[i][j][k].equal(psi_s[i][j][k]);
      }
    }
  }
}
//////////
```

ラプラシアン演算の実装

境界条件の実装

```
/// constraints by the applied field and current
 /// y-z plane ( Bz-Uy-component ) set Ja = 0 when longitudinal
 for(int k = 1; k <= Nz; k++){
   for(int j = 1; j <= Ny-1; j++){
     ex.setPolar(1.0,  h*h*(Ba + Ja*SSx/2.0));
     Uy[1][j][k].equal(Uy[2][j][k].mul(Ux[1][j+1][k].conj()).mul(Ux[1][j][k]).mul(ex));
     ex.setPolar(1.0, -h*h*(Ba - Ja*SSx/2.0));
     Uy[Nx][j][k].equal(Uy[Nx-1][j][k].mul(Ux[Nx-1][j+1][k]).mul(Ux[Nx-1][j][k].conj()).mul(ex));
   }
 }
 /// z-x plane ( Bz-Ux-component ) omit when transverse
 for(int i = 1; i <= Nx-1; i++){
   for(int k = 1; k <= Nz; k++){
     ex.setPolar(1.0, - h*h*Ba);
     Ux[i][1][k].equal( (Uy[i+1][1][k].conj()).mul(Ux[i][2][k]).mul(Uy[i][1][k]).mul(ex) );
     ex.setPolar(1.0,   h*h*Ba);
     Ux[i][Ny][k].equal( Ux[1][Ny-1][k].mul(Uy[i+1][Ny-1][k]).mul(Uy[i][Ny-1][k].conj()).mul(ex) );
   }
 }
 /// y-z plane ( J=(0,0,Jb)-Uz-component ) longitudinal
 for(int k = 1; k <= Nz-1; k++){
   for(int j = 1; j <= Ny; j++){
     ex.setPolar(1.0, h*h*Jb*SSx/2.0);
     Uz[1][j][k].equal( Ux[1][j][k].mul(Uz[2][j][k]).mul(Ux[1][j][k+1].conj()).mul(ex) );
     Uz[Nx][j][k].equal( (Ux[Nx-1][j][k].conj()).mul(Uz[Nx-1][j][k]).mul(Ux[Nx-1][j][k+1]).mul(ex) );
   }
 }
 /// z-x plane ( J=(0,0,Jb)-Uz-component ) longitudinal
 for(int i = 1; i <= Nx; i++){
   for(int k = 1; k <= Nz-1; k++){
     ex.setPolar(1.0, h*h*Jb*SSy/2.0);
     Uz[i][1][k].equal( Uy[i][1][k].mul(Uz[i][2][k]).mul(Uy[i][1][k+1].conj()).mul(ex) );
     Uz[i][Ny][k].equal( (Uy[i][Ny-1][k].conj()).mul(Uz[i][Ny-1][k]).mul(Uy[i][Ny-1][k]).mul(ex) );
   }
 }
```

今後の課題

- 各種物理量の表示
- アルゴリズムの改良
- 並列化（高速化）

- ピンニング
- Two-component 系