2．7　イジング・モデル

個体（原子，動物）が近隣の他の個体の挙動と順応

⇒　振舞を模倣

物理学：　強磁性体，2元合金の相分離，スピン・グラス

生物学：　ニューラル・ネットワーク，群れをなす鳥，魚の群れ

蛍の閃光，心室の鼓動，伝染病

ファッションの流行

２．７．１　確率論的イジング・モデル

カノニカル集団の形成　⇒　温度一定の条件

２次元，n×n正方格子

格子サイトの値　＝　1（上向きスピン），-1（下向きスピン）

近接サイトと強さJの２体間の相互作用

J：交換エネルギーの定数，　２体間：step-wise manner

J > 0：　スピンが同方向でエネルギー減

J < 0：　スピンが逆方向でエネルギー減

磁化：　上向きと下向きのスピン数の差

プログラム

```
In[1] := IsingMetropolis[n_, m_, B_, J_, p_] :=
          Module[{energydiff, initconfig, flip, flipList},
```

反転した結果のエネルギー変化

2×（サイトの値）×（B +（近接サイトの全スピン））

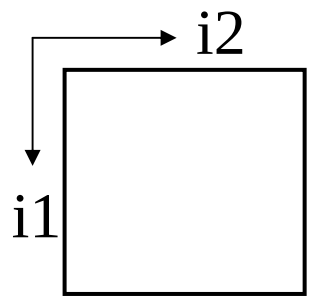
```
          = 2 lat[[i1, i2]](B + J nnvalsum)
          lat[[i1, i2]]     = 1 or –1
          nnvalsum = 4, 2, 0, -2, -4
          energydiff [ 1,  4] =  2 (B + 4J);
          energydiff [ 1,  2] =  2 (B + 2J);
          energydiff [ 1,  0] =  2 (B + 0J);
          energydiff [ 1, -2] =  2 (B - 2J);
          energydiff [ 1, -4] =  2 (B - 4J);
          energydiff [-1,  4] = -2 (B + 4J);
          energydiff [-1,  2] = -2 (B + 2J);
          energydiff [-1,  0] = -2 (B + 0J);
          energydiff [-1, -2] = -2 (B - 4J);
          energydiff [-1, -4] = -2 (B - 4J);
```

初期配置（n×n正方格子，1 or –1 をランダムに配置，
p: 下向きスピンの確率）

1 or 0

```
initconfig = Table[ 2 Floor[p + Random[ ]] – 1, {n}, {n}];
```

2 or 0



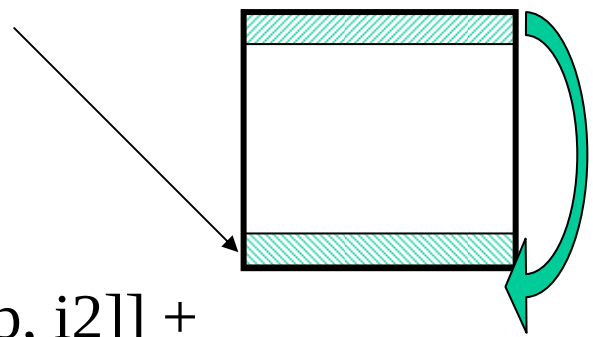

```
flip = (
```

1つの格子サイトを選ぶ

```
    lat = #;
    {i1, i2} =
     {Random[Integer, {1, n}], Random[Integer, {1, n}]};
```

ノイマン近傍，周期的境界条件

```
    If [i1 = = n, dn = 1, dn = i1 + 1];
    If [i2 = = n, rt = 1, rt = i2 + 1];
    If [i1 = = 1, up = n, up = i1 - 1];
    If [i2 = = 1, lt = n, lt = i2 - 1];
    nnvalsum = lat[[dn, i2]] + lat[[up, i2]] +
                    lat[[i1, rt]] + lat[[i1, lt]];
```



スピンを反転させるかどうか決める
（エネルギー変化が負または正でも小さい場合）

```
If [energydiff [lat[[i1, i2]], nnvalsum] < 0 ||
Random[ ] < Exp[-energydiff [lat[[i1, i2]], nnvalsum]],
lat[[i1, i2]] = -lat[[i1, i2]]; lat, lat]
)&;
```

各ステップをm回実行し，monteCarlostepLips（すべての要素を含むサブリストを作る.

```
flipLis = NestList[flip, initconfig, m];
flipLis[[Range[1, m, n^2]]
]]
```

磁化特性

```
In[2] := ShowIsingMagnetization[list_] :=
          Module[{n=Length[List[[1]]],
                  longRangeOrderList},
             longRangeOrderList :=
               Map[Abs[Apply[Plus, Flatten[#]/n^2]&, list];
             ListPlot[longRangeList,
               PlotJoined -> True,
               PlotRange -> {0, 0.6},
               AxesLabel -> {FontForm[“step”, {“Times-Italic”, 8}],
                             FontForm[“M”, {“Times-Italic”, 8}],
               PlotLabel -> FontForm[“Long-range order parameter”,
                                     {“Helvetica”, 10}]]]
```

2．7．2　Q2R イジングモデル

Q2R	確率論的（メトロポリス）
小正準集団（エネルギー一定） 全てのスピンが同時に反転 決定論的	正準集団（温度一定） １つのスピンだけ反転 確率論的

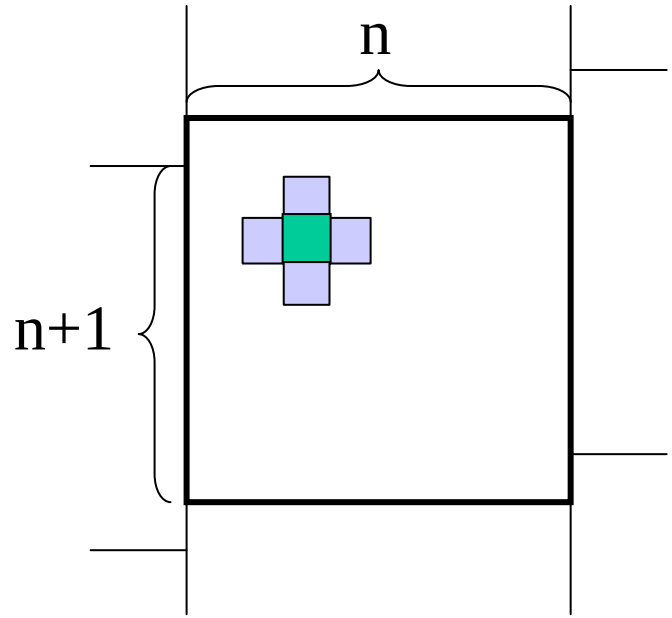
［モデル］

斜行境界条件

２次元，（n + 1）×n 格子

格子サイトの値＝ 1 or –1

ノイマン近傍



プログラム

```
In[1] := IsingCA[n_?OddQ, m_] :=
          Module[{lat, subLatOdd, subLatEven, update, spinFlip},
```

初期格子配置

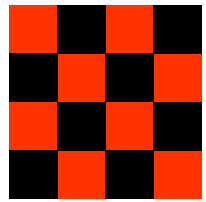
```
            lat = Table[2 Random[Integer] – 1, {n + 1}, {n}];
            subLatOdd = Flatten[Partition[Flatten[lat], 1, 2]];
            subLatEven = Flatten[Partition[Rest[Flatten[lat]], 1, 2]];
```

更新ルール(必要条件:スピン反転で系のエネルギーを不変)

サイトの値　上　左　右　下

反転スピン＝同数

```
            update[x_, -1, -1, 1, 1] := -x;
            update[x_, -1, 1, -1, 1] := -x;
            update[x_, -1, 1, 1, -1] := -x;
            update[x_, 1, -1, -1, 1] := -x;
            update[x_, 1, -1, 1, -1] := -x;
            update[x_, 1, 1, -1, -1] := -x;
            update[x_, _, _, _, _] := x;
            Attributes[update] = Listable
```



サブ格子

■(■)の更新値は最近接サイト■(■)の値で決まる.

新しく更新されたサブ格子のリスト

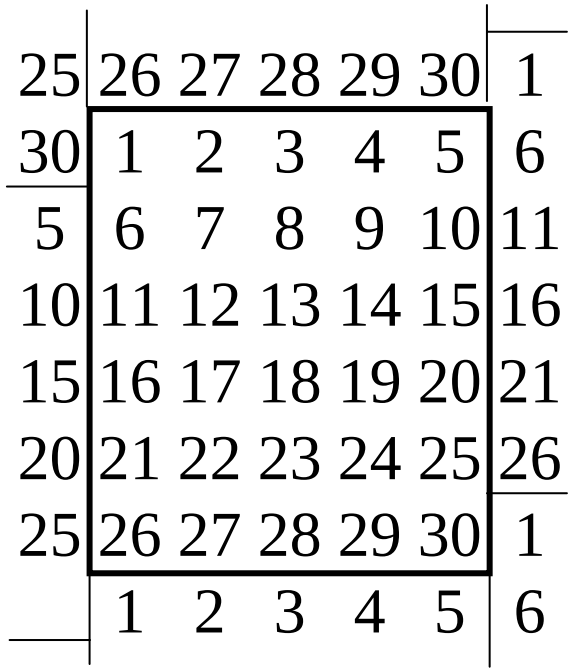
```
spinFlip[{oldOdd_, oldEven_}] :=
        Module[{newOdd, newEven},
            newOdd = update[oldOdd,
                        RotateLeft[oldEven, (-n-1)/2],
                        RotateLeft[oldEven, -1],
                        RotateLeft[oldEven, 0],
                        RotateLeft[oldEven, (n-1)/2]];
            newEven = update[oldEven,
                        RotateLeft[oldOdd, (-n+1)/2],
            {newOdd, newEven}];
        Map[Partition[Flatten[Transpose[#]], n]&,
            NestList[spinFlip, {subLatOdd, subLatEven}, m]]
]
```

（例）

In[1] := (mat = Partition[Range[30], 5])//MatrixForm

Out[1]//MatrixForm =

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

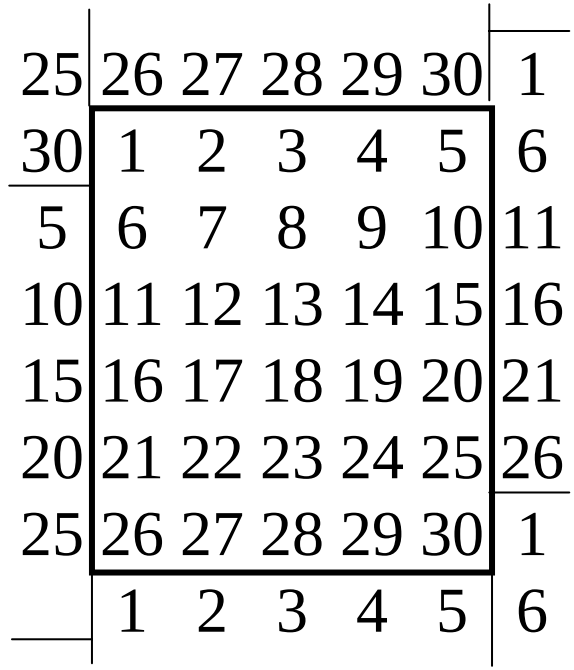


In[2] := oddSites = Flatten[Partition[Flatten[mat], 1, 2]]

Out[2] = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29}

In[3] := evenSites = Flatten[Partition[Rest[Flatten[mat]], 1, 2]]

Out[3] = {2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30}




```
In[4] := ngbrsNorthOfOddSites = RotateLeft[evenSites, (-5-1)/2]
Out[4] = {26, 28, 30, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24}
In[5] := ngbrsWestOfOddSites = RotateLeft[evenSites, -1]
In[6] := ngbrsEastOfOddSites = RotateLeft[evenSites, 0]
In[7] := ngbrsSouthOfOddSites = RotateLeft[evenSites, (5-1)/2]
In[8] := ngbrsNorthOfOddSites = RotateLeft[evenSites, (-5+1)/2]
In[9] := ngbrsWestOfOddSites = RotateLeft[evenSites, 0]
In[10] := ngbrsEastOfOddSites = RotateLeft[evenSites, 1]
In[11] := ngbrsSouthOfOddSites = RotateLeft[evenSites, (5+1)/2]
```

２．８　交通

非対称（不調和）排他性を持つ駆動力のある拡散系（driven diffusive system）での層流-乱流転移

硬い斥力（hard core repulsion）：　１度に1つの格子サイトに１つの粒子しか占有できない

異なる確率で左右に動く　⇒　全体としてある方向への粒子の正味の流れ，移動

［1次元セルラオートマトン］

周期的境界条件，長さ s の1次元格子

サイトの値 ＝ 0 ~ vmax, e

更新：　その値と右の最近接サイトの値に基づく

２．８．１　一車線プログラム

```
In[1] := OneLine[s_, p_, vmax_, t_] :=
          Module[{emptyRoad, road, carDensity, followTheLeader},
```

初期の road 配置，位置，速度のランダム分布

p: 道路の位置が1台の車で占有される確率

e: 道路の空き空間，s: 格子のサイズ

```
              emptyRoad = Table[e, {s}];
              road = Table[Floor[p +Random[ ], {s}]*
                        Table[Random[Integer, {1, vmax+1}], {s}/.
                   {0 -> e, x_Integer -> (x-1)};
```

車の密度，　s → 大　で　密度 → p

```
              carDensity = N[Count[road, _Integer]/s];
```

車の速度 (a) 車間が詰まりすぎれば，速度を下げる
(b) (a) でなければ，vmax まで速度を上げる

vels: 車の速度のリスト，　distance: 車間距離，
locs: 現在の車の位置のリスト，　y: road 配置

```
    followTheLeader =
      Function[y,
        vels = DeleteCases[y, e];
        locs = Complement[Range[s], Flatten[Position[y, e]]];
        distances = Join[Rest[locs – RotateRight[locs]],
                         {s – Last[locs] + First[locs]}];
        newVels = Map[(Min[#[[1]], #[[2]], vmax])&,
                  Transpose[{vels + 1, distances –1}]];
        newLocs = Map[(Mod[#-1, s] + 1)&,
                  (newVels + locs)];
        Fold[ReplacePart[#1, #2[[1]], #2[[2]]]&,
             emptyRoad, Transpose[{newVels, newLocs}]]];
    NestList[followTheLeader, road, t]
  ]
```

プログラムの実行

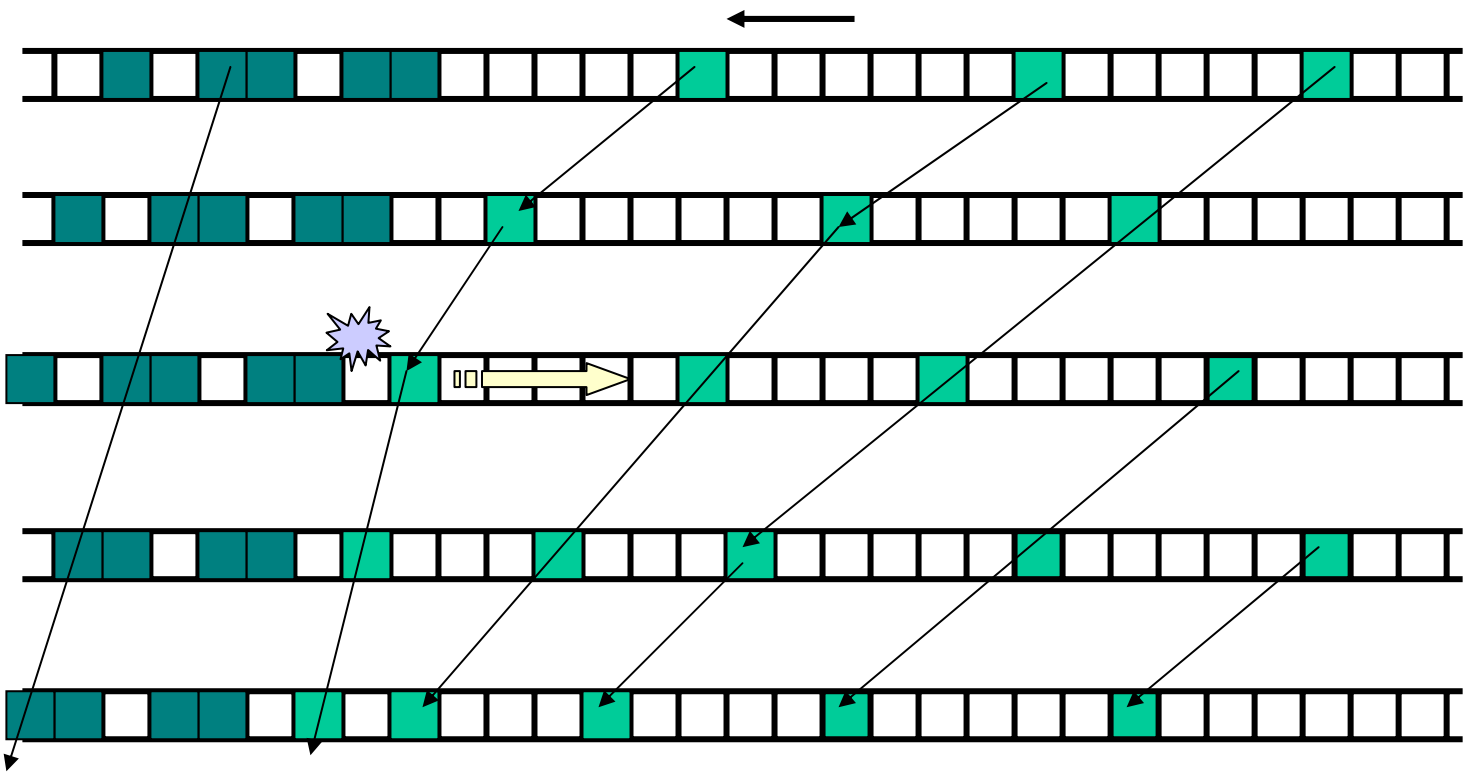
道路: 黒(Hue[0, 0, 0]), 車: 異なる速度を異なる色合

RasterArray

```
In[2] := ShowTraffic[list_List, opts_ _ _ ] :=
            Module[{vmax = Max[List/. e -> -1]},
              Show[{Graphics[RasterArray[Reverse[list]/.
                Join[{e -> Hue[0, 0, 0]},
                  Thread[Range[0, vmax] ->
                    (Map[Hue, Table[Random[ ], {vmax + 1}]])]
              ]]],
              opts, AspectRatio -> Automatic]]
```

衝撃波を生成する動粘度波（kinematic waves）

- 前の車の速度に合わせるとき
- 交通が動くと同方向か，逆方向に進む一定の流れの波
  ＝　高速で広がった群と低速で接近した車束
- 衝撃波：　高速車が低速車と衝突するのを避けるために，
  突然ブレーキをかけるとき作られる．



2．8．2　追い越し車線のある二車線一方通行

2つの相互作用している車線からなる2状態CA

相互作用：　走行車線でブロックされ追い越し車線の隣接位置にずれる．

2状態：　空の空間，車

周期的境界条件

長さ s の2つの1次元格子

サイトの値　＝　0（空空間），　1（車）

更新：　決まった数の時間ステップの間

格子サイトの値，右，上，下の最近接サイトの値に基づく

車の速度の分布はない

二車線プログラム

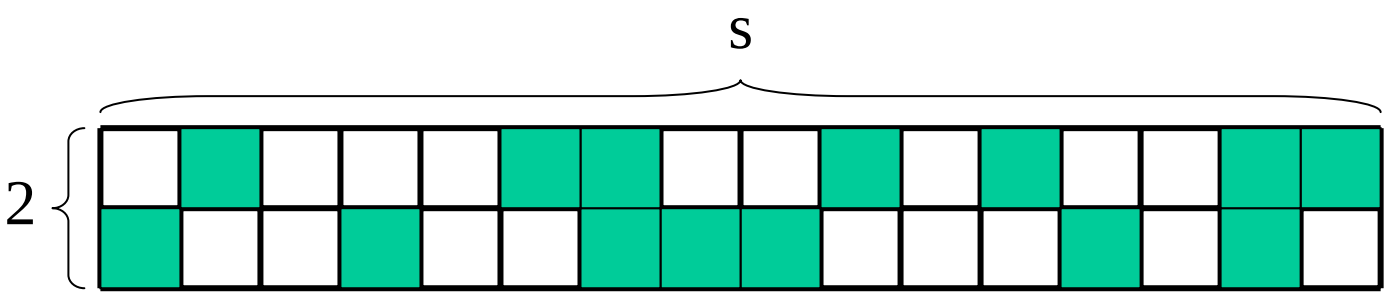
```
In[1] := TwoLane[s_, t_, p_ ] :=
          Module[{highway, totalcars, carDensity, ruleR,
                    count = 0, velLis = {}, ruleL, LaneRupdate,
                    laneLupdate, stopAndGo},
```

初期の高速道路の配置（p: 車で占有されている確率）

```
      highway = Table[Floor[p + Random[ ]], {2}, {s}
```

車の密度

```
      totalcars = Apply[Plus, Flatten[highway]];
      carDensity = N[totalcars/(2s)];
```

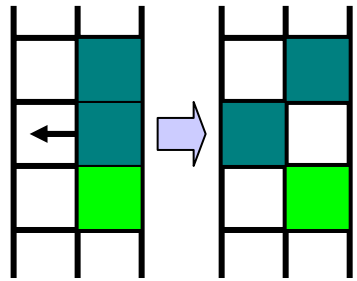


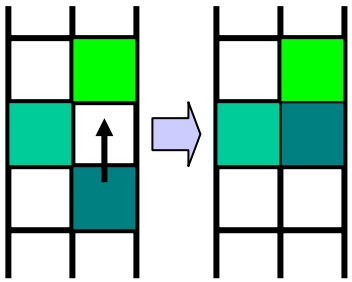
更新ルール（一斉）

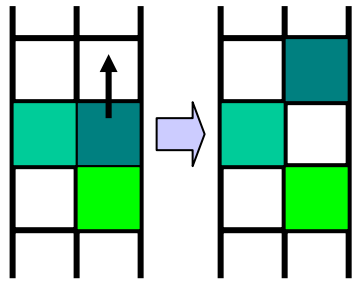
t + 0.5 で，右車線の車は，前に進むか，左車線に移るか，止まる

後　右　前　左　　左　右

```
ruleR[1, 0, _, x_] := (count + + ;{x, 1});
ruleR[_, 1, 1, 0]  := {1, 0};
ruleR[_, 1, 0, x_] :={x, 0};
ruleR[_, y_, _, x_] :={x, y};
Attributes[ruleR] = Listable;
```







t + 1 で, 左車線の車


```
ruleL[1, 0, _, x_] := (count + + ;{1, x});
ruleL[_, 1, 1, 0]  := {0, 1};
ruleL[_, 1, 0, x_] :={0, x};
ruleL[_, y_, _, x_] :={y, x};
Attributes[ruleL] = Listable;
```

更新

```
laneRupdate[lis_ ] :=
 Transpose[rykeR[RotateLeft[lis[[2]], -1], lis[[2]],
                      RotateLeft[lis[[2]], 1], lis[[1]] ];

laneLupdate[lis_ ] :=
 Transpose[rykeR[RotateLeft[lis[[1]], -1], lis[[1]],
                      RotateLeft[lis[[1]], 1], lis[[2]] ];
```

時間ステップの更新，車の数は保存

```
stopAndGo[lane_ ] := Module[{ },
                     AppendTo[velLis, count];
                     count = 0 ;
                     laneLupdate[laneRupdate[lane]]];
```

t 回の時間ステップで発展，MeanVelocityLis: 平均速度

```
NestList[stopAndGo, hghwy, t];
velLis = Join[velLis, {count}];
MeanVelocityLis = N[velLis/totalcars]
]
```