卒業論文

物理現象視覚化ソフトのライブラリ構築

関西学院大学理工学部

情報科学科 西谷研究室

1536 榊原　健

2015年3月

概 要

物理現象を数式だけから理解するのは非常に困難である．例えば，粒子の運動を解析する分子動力学法 (MD) では，ニュートンの運動方程式に従って粒子の運動を数値的に決定していく．しかし，数値を直接追いかけるだけでは，直感的な理解が得られない．この際,実際に粒子の運動を視覚化することによって直感的な理解を深め，学習を促進することができると考えられる．本研究では Web 上で Interactive な操作が可能で，MD の深い理解の助けとなるツールの作成を目的とする．今回は，Processing 言語を用いて，MD における粒子の運動を操作・視覚化するプログラムを作成した．また，Web 上で動作させるために，作成したプログラムを JavaScript 言語に変換した．JavaScript は動的な Web サイト構築に有効な言語の一つである．視覚表示を容易に実現する Processing 言語においては，JavaScript 言語への自動変換が組み込まれおり，Processing 言語で作成したプログラムを，ブラウザ上で動作させることができる．この自動変換には問題点があり，それらの洗い出しと解決策を検討した．プログラムを作成した結果，クラスタや凝固による粒子の振る舞いを確認することができるようになった．また，ブラウザ上で Interactive な操作が可能なため手軽に扱うことができ，今後の学習教材としても有効なものとなった．そして，ライブラリとしてプログラムの解説を行う事によって今後の継続的な発展に繋がるようにした．

目 次

第 1 章　序論 **3**

第 2 章　物理的背景 **4**

2.1 分子動力学法 4

2.1.1 Verlet 法 4

2.1.2 導出 4

2.2 Lennard-Jones ポテンシャル 6

第 3 章　視覚化 **8**

3.1 Processing 8

3.2 JavaScript 8

3.3 変換の問題点 8

3.3.1 ライブラリの使用 9

3.3.2 関数名と変数名が同じ 9

3.3.3 JavaScript の変数型 9

3.3.4 translate による座標の移動 10

第 4 章　結果 **12**

4.1 作成したプログラム 12

第 5 章　プログラム解説 **17**

5.1 アルゴリズム 17

5.2 粒子の初期状態の設定 17

5.3 粒子に加わる力 18

5.3.1 粒子間に加わる力の計算 18

5.3.2 合力の計算 20

5.4 粒子の動作 20
5.4.1 Verlet 法による計算 20
5.4.2 座標の更新 20
5.5 粒子の描画 21
5.5.1 描画 21
5.5.2 粒子の色 22
5.6 壁との衝突 23
5.7 選択した粒子に力を加える 25
5.7.1 粒子の選択 26
5.7.2 ドラッグ後の処理 27
5.8 キー入力 28
5.9 凝固状態 28
5.10 力を視覚化した状態 29
5.11 スライダー 31

第 6 章 総括 **33**

付 録 A プログラムのソースコード **35**

第1章　序論

原子レベルの物理現象を数式だけから理解するのは非常に困難である．例えば，粒子の動きを解析する分子動力学法では，ニュートンの運動方程式に従い粒子の運動を決定する．それらは，数値の羅列で表され直感的な理解が得られない．学習において自分の頭の中でイメージを作ることは重要であり，理解を助長することができると考えられる．そこで，本研究ではグラフィック機能に特化したProcessingを用いて分子動力学法での粒子の振る舞いの視覚化を行い，Interactiveな操作が可能なプログラムを作成する．また，それらを手軽に扱えるようにWebブラウザ上で動作可能なJavaScriptへの変換を行う．JavaScriptへの変換はProcessing上で行う事ができるがそれには問題点があり，それらの洗い出しと解決策を検討する．さらに，作成したプログラムをライブラリ化し，解説を行うことで今後の継続的な発展に繋がると考えられる．

第2章　物理的背景

2.1　分子動力学法

分子動力学法は運動方程式を解く事によって，粒子の振る舞いを解析する手法である[1]．ニュートンの運動方程式はエネルギー保存則を満たすため，エネルギーが保存される集団において用いられる．

2.1.1　Verlet 法

Verlet 法は分子動力学法における粒子の座標を逐次的に求める方法であり，式 (2.1) のように表される．

$$r(t+h) = 2r(t) - r(t-h) + \frac{h^2}{m}f(t) \tag{2.1}$$

ここで，$r(t)$ は時刻 t における粒子の座標，$f(t)$ は時刻 t における粒子に加わっている力，h は微小時間，m は粒子の質量を表している．式 (2.1) は，時刻 t における粒子の座標，時刻 $t-h$ における粒子の座標，時刻 t における粒子に加わっている力，粒子の質量の 4 つの要素から，時刻 $t+h$ における粒子の座標が求まることを表している．粒子に加わっている力を常に決定することができれば，Verlet 法を継続的に用いることが可能であり，逐次的に粒子の座標を決定し続けることができる．また，粒子を動かすために速度が必要そうだが，この手法では速度を必要としないという特徴がある．この手法はニュートンの運動方程式から導出されるため，エネルギーが保存される系で有効である．

2.1.2　導出

時刻 $t+h$ における粒子の座標 $r(t+h)$ にテイラー展開を行うと式 (2.2) ができる．

$$r(t+h) = r(t) + h\frac{dr(t)}{dt} + \frac{h^2}{2!}\frac{d^2r(t)}{dt^2} + \frac{h^3}{3!}\frac{d^3r(t)}{dt^3} + ... \tag{2.2}$$

h は微小時間のため h^3 以上の項を無視すると

$$r(t+h) = r(t) + h\frac{dr(t)}{dt} + \frac{h^2}{2!}\frac{d^2r(t)}{dt^2} \tag{2.3}$$

h を $-h$ に置き換えると

$$r(t-h) = r(t) - h\frac{dr(t)}{dt} + \frac{h^2}{2!}\frac{d^2r(t)}{dt^2} \tag{2.4}$$

式 (2.3) と式 (2.4) を足し合わせ $r(t+h)$ について移項させると式 (2.5) ができる．

$$r(t+h) = 2r(t) - r(t-h) - h^2\frac{d^2r(t)}{dt^2} \tag{2.5}$$

ここでニュートンの運動方程式について考える．$v(t)$ を時刻 t の速度，$r(t)$ を時刻 t の位置とすると式 (2.6) ができる．

$$v(t) = \frac{dr(t)}{dt} \tag{2.6}$$

また，式 (2.6) より時刻 t における加速度 $a(t)$ は

$$a(t) = \frac{d^2r(t)}{dt^2} \tag{2.7}$$

ここで $f(t)$ を時刻 t に作用する力，m を質量とするとニュートンの運動方程式は式 (2.8) となる．

$$f(t) = m\frac{d^2r(t)}{dt^2} \tag{2.8}$$

移項させると

$$\frac{f(t)}{m} = \frac{d^2 r(t)}{dt^2} \tag{2.9}$$

式 (2.9) を式 (2.5) にを代入すると式 (2.10) となり Verlet 法が導出される．

$$r(t+h) = 2r(t) - r(t-h) + \frac{h^2}{m} f(t) \tag{2.10}$$

2.2　Lennard-Jones ポテンシャル

Lennard-Jones ポテンシャルとは 2 体間での相互作用ポテンシャルエネルギーを経験則的に表したモデルである [2]．ψ をポテンシャルエネルギー，R を原子間距離，A，B は任意の定数とすると式 (2.11) のように表される．

$$\psi(R) = A\left(\frac{1}{R}\right)^{12} + B\left(\frac{1}{R}\right)^{6} \tag{2.11}$$

式 (2.11) は縦軸をポテンシャルエネルギー，横軸を粒子間距離とすると図 2.1 のような概形になる．

ポテンシャルエネルギーの理解には，安定した状態からずれるとエネルギーが生じるという考え方が適切である．図 2.1 では極小点が平衡原子間距離であり，双方の粒子が安定した状態であると言える．原子間距離が安定状態より近くなれば，急激にエネルギーが上がる．これは近づくことで原子同士の影響力が増大するからである．逆に，安定状態より遠くなると，エネルギーは緩やかに上がっていき，ある高さで上昇が止まる．これは原子同士の影響力が減少するからである．これに加えて，原子に作用する力について考えていく．ポテンシャルエネルギーを距離で微分することにより作用する力が求まる．図 2.1 の傾きに注目すると，平衡原子間距離は極小点であり傾きは 0 のため力が作用しない．また，距離が近くなれば傾きは急激に負の値をとり斥力が生まれる．逆に，遠くなれば傾きは正の値をとり引力が生まれるが，傾きが徐々に 0 に近づくため力が弱まっていく．このように図 2.1 のような概形のポテンシャルエネルギーは，バネのような振る舞いをすることがわかる．

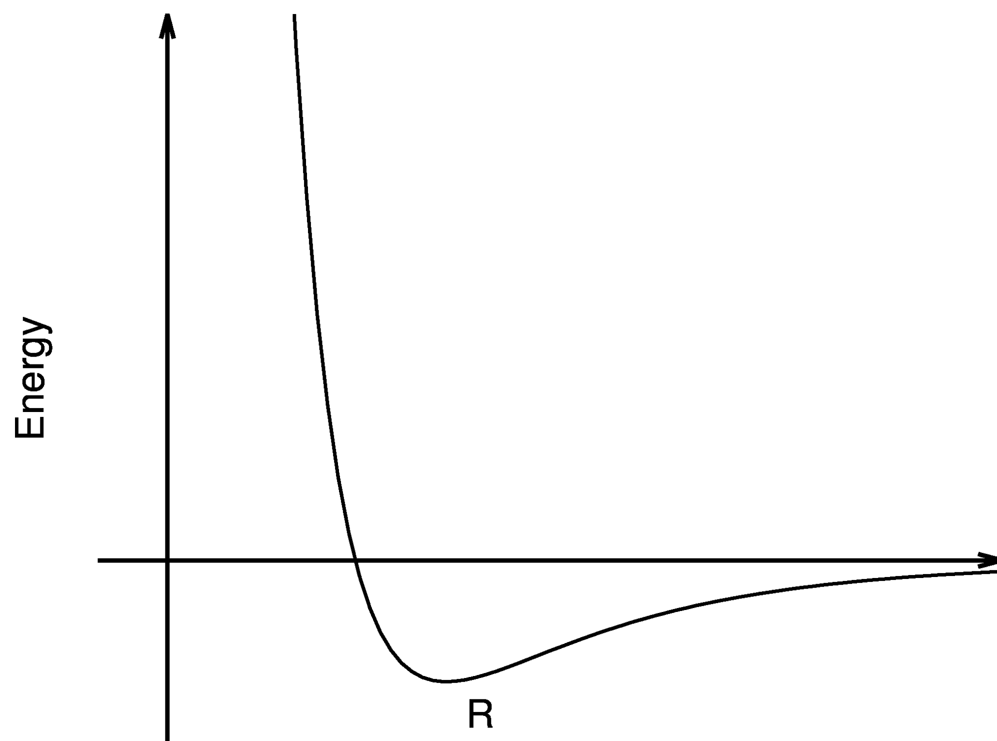

図 2.1: Lennard-Jones ポテンシャル.

第3章　視覚化

物理現象を数式から理解するのは困難である．視覚化を行うことによって理解の促進に繋がると考えられる．本章では視覚化に用いるプログラミング言語について記述する．

3.1　Processing

Processing はグラフィック機能に特化したオープンソースのプログラミング言語である．文法は Java によく似ており，デフォルトでスケッチブックが用意されていて容易に図の描画を行う事ができる．また，Processing には JavaScript への自動変換機能が備わっていて，Processing で作成したプログラムを Web 上で動作させる事が可能になる．この自動変換には問題点があり本研究で明らかになったものは 3.3 節に記す.

3.2　JavaScript

JavaScript は動的な Web ページを作成するために開発されたプログラミング言語である．Java と名前が似ているが両者は全く別のものである．HTML 内に JavaScript のプログラムを埋め込むことにより，Web 上でそのプログラムを動かすことが可能である.

3.3　変換の問題点

本研究で明らかになった Processing から JavaScript への変換により生じる問題点を以下に記す.

3.3.1　ライブラリの使用

ライブラリとは便利なパーツのようなものであり，Processing には最初から備わっている公式のライブラリと，インストールすることで利用することができるライブラリがある．それらはプログラム中で宣言することで容易に利用することができる．本研究のプログラムではスライダー部分に ControlP5 というスライダーやボタンなどの GUI パーツのライブラリを利用していた．しかし，ライブラリを利用したプログラムでは JavaScript への変換後正しく動作しない事が明らかになった．解決策として今回はスライダー部分を自分で実装することにした．スライダーの実装については 5.11 節に記す．

3.3.2　関数名と変数名が同じ

関数名と変数名が同じ場合，JavaScript への変換後，正しく動作しない．例えば以下のようなプログラムがある．

```
void same_name(){
  println(0);
}

void setup(){
 int same_name;
 same_name();
 println(1);
}
```

このプログラムには same_name という名前の変数と関数が使われている．これを実行すると Processing では 01 と出力され，JavaScript では出力なしという結果になる．

3.3.3　JavaScript の変数型

Processing は整数の int 型変数, 浮動小数点の float 型変数と分けているが，JavaScript の変数にはそれらの型の概念がなくどちらも同じ変数として扱われる．例えば以下のようなプログラムがある．

```
void setup(){
  int integer;
  integer = 3/2;
  println(integer);
}
```

このプログラムはint型変数intergerに1.5が代入され，intergerの中身を出力する．Processingではint型のため1と出力されるが，JavaScriptでは1.5と出力される．これはJavaScriptの変数にint型がないことが原因であり，解決策として代入部分を以下のように書けばよい．

```
integer = (int)(3/2);
```

このように代入する値をint型にすればJavaScriptでも1と出力され，Processingと揃えることができる．このような値のずれは気付くのが困難なため注意が必要である．

3.3.4 translateによる座標の移動

mousePressedやkeyPressedなどの強制的に呼び出される関数内でtranslateを行うと座標のずれが生じる．例えば以下のようなプログラムがある．

```
void setup(){
  size(500,500);
}

void mousePressed(){
  translate(10,10);
  ellipse(130,130,10,10);
}

void draw(){
  background(255);
  fill(0);
  ellipse(100,100,10,10);
}
```

このプログラムは座標 (100,100) に常に黒い円があり，クリックするたびに座標 (140,140) に黒い円が一瞬描かれるというものである．JavaScript への変換後はクリックするたびに常に描かれている黒い円が座標 (10,10) ずつずれていく．解決策として関数 mousePressed 内を以下のように書き換えればよい．

```
void mousePressed(){
  translate(10,10);
  ellipse(130,130,10,10);
  translate(-10,-10);
}
```

このように関数の最後で translate(-10,-10) を実行し関数内で translate で移動させた座標を戻すことによって JavaScript の動作を Processing と一致させることができる．

第4章　結果

本章では，分子動力学法を用い粒子の振る舞いをシミュレーションし視覚化するプログラムの実行結果を記述する．

4.1　作成したプログラム

図4.1は本研究で作成したVerlet法とLennard-Jonesポテンシャルを用いて粒子の振る舞いをシミュレーションし，視覚化を行うプログラムである．Processingで作成しているが，JavaScriptへ変換する事によってWebブラウザ上で動作させる事が可能である．左の枠内に複数の粒子モデルが描画され，それらが枠内を動き回る．また，粒子の色を速度によって青から赤に変化させてエネルギーの推移をわかりやすくし，マウスカーソルで粒子をクリックしドラッグする事によって好きな方向に力を加える事ができる．右側には2本のスライダーがあり，粒子の数とVerlet法の時間刻みを調整できる．キー入力により状態を変更することができ，Sキーで凝固状態，Fキーで粒子に加わる力を視覚化した状態，Rキーで粒子のリセットを行うことができる．

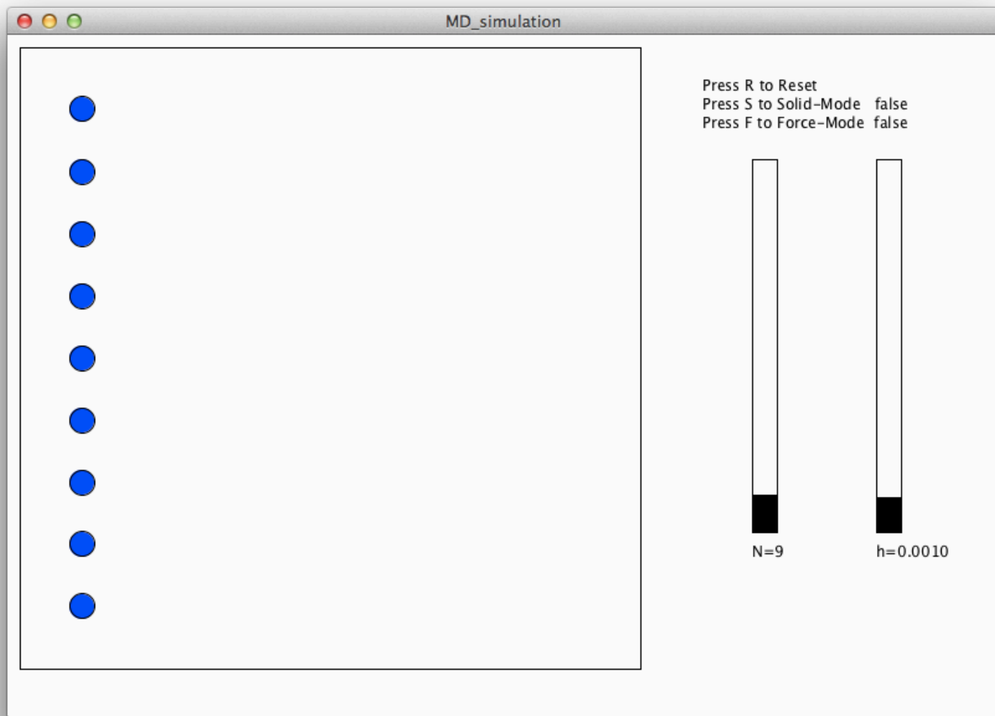

図 4.1: 分子動力学法視覚化プログラム.

図 4.2 は粒子クラスタの移動を表しており，それぞれの粒子のクラスタとしての振る舞いを視認することができる．スライダーで粒子の数を変更しマウスで力を加えることによって，このようなクラスタを好きなようにシミュレーションできる．

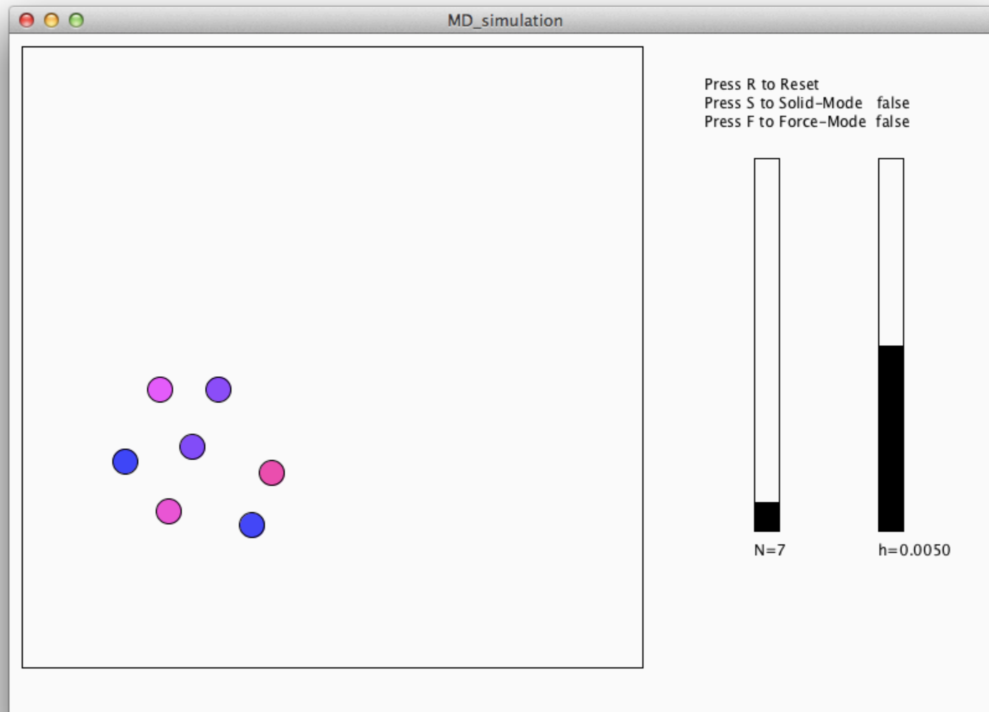

図 4.2: クラスタの移動.

図 4.3 は凝固状態にしてしばらく実行し続けた結果であり，粒子が下の方に集まっていることが確認できる．これは粒子に下向きの力を継続的に加えて擬似的に凝固の現象を表現している．これにより，粒子が自由に動き回る液体や気体の状態から，個体に移り変わる凝固現象の様子を確認できる．

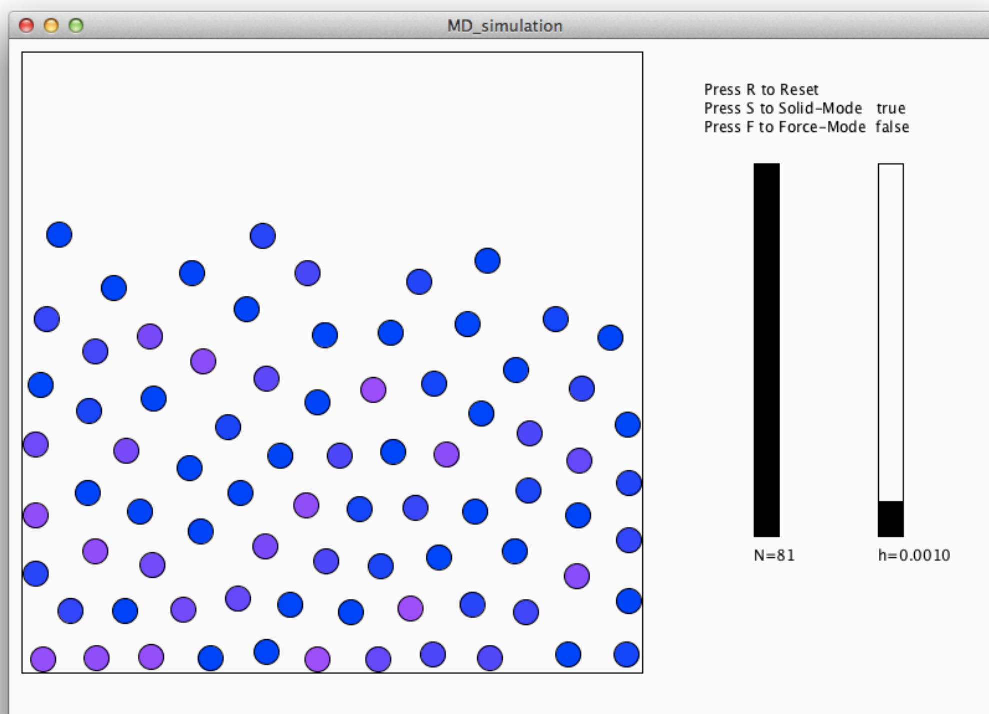

図 4.3: 凝固状態.

図 4.4 は粒子に加わる力を視覚化した状態であり，それぞれの粒子の中心から線が伸びている．この線は長さが力の大きさ，向きが力の向きと対応しており力の加わり方をリアルタイムで把握することができる．それぞれの粒子が力のパラメータを持っており，それらはシュミレーションの中で瞬時に変化していくため，値の確認は困難である．しかし，この視覚化のおかげでそれぞれの粒子にかかる力を直感的に確認できる．

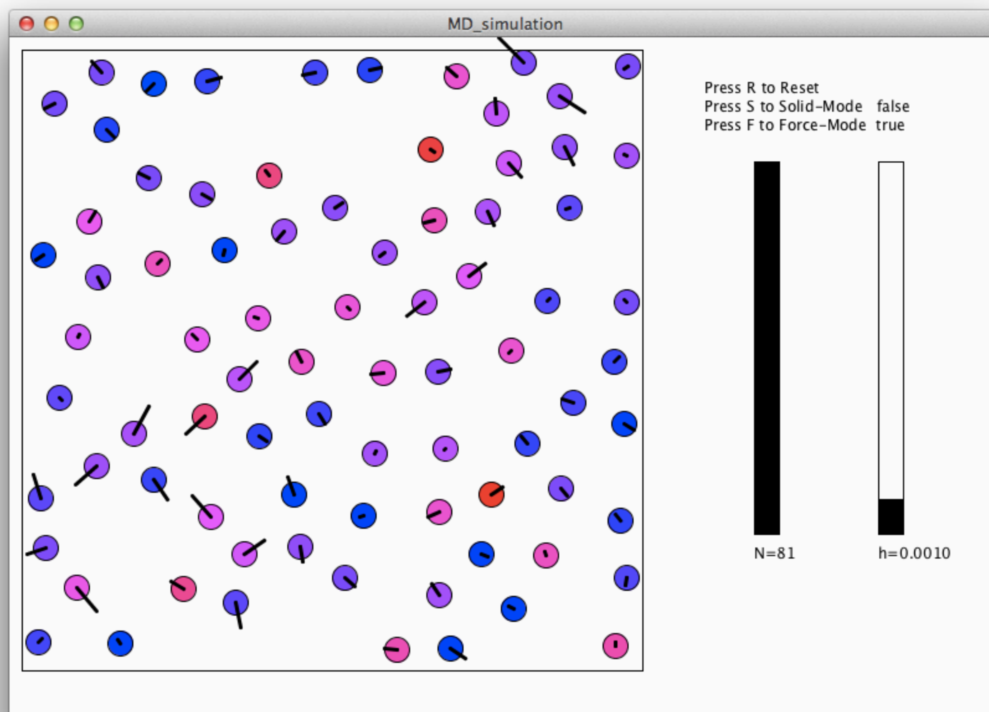

図 4.4: 粒子に加わる力を視覚化した状態.

第5章　プログラム解説

本章では，今回作成したプログラムをライブラリ化し継続的な発展が可能なようにそれぞれの処理の解説を記述する．

5.1　アルゴリズム

分子動力学法を用いて粒子を動かしていくアルゴリズムは以下のようになる．

1. 複数の粒子モデルを用意し，現在の座標、1つ前の座標，粒子に加わる力のパラメータを持たせる．
2. Lennard-Jones ポテンシャルを用いて，粒子モデルに加わる力の合力を計算する．
3. Verlet 法を用いて次の座標を計算する．
4. 2-3 を繰り返して粒子モデルの座標を継続的に決定し続ける．

5.2　粒子の初期状態の設定

粒子の座標，粒子にかかっている力を初期状態にする関数 set_particle は以下のように実装した．

```
void set_particle(){
  int i, j, k=0;
  for(i=1; i<10; i++){
    for(j=1; j<10; j++){
      if(k==n) break;
      ball_p[k][0] = i;
      ball_p[k][1] = j;
      pre_p[k][0] = i;
      pre_p[k][1] = j;
      k++;
    }
  }
  for(i=0; i<n; i++){
    ball_f[i][0] = 0;
    ball_f[i][1] = 0;
  }
}
```

変数 i, j は粒子の座標を等間隔にするための任意の数値であり，k は粒子の番号，n は粒子の個数である．粒子の数だけ，粒子の座標 (ball_p) と過去の座標 (pre_p) の初期値の格納を行い，粒子にかかっている力 (ball_f) を初期状態の 0 に設定している．座標の初期値は左上から縦に 9 個ずつ等間隔に並ぶようになる．この関数はリセットにも用いられており，スライダーによって粒子の数が変更された際も実行される．

5.3 粒子に加わる力

ある一つの粒子に加わる力を決定するためには，他の粒子それぞれに対して力の計算を行い，それらの合力を求める必要がある．

5.3.1 粒子間に加わる力の計算

粒子間距離から Lennard-Jones ポテンシャルを用いて，粒子に加わる力を求める．2 粒子の座標からそれぞれの粒子に加わる力を計算する関数 inter_force は以下ように実装した．

```
float[] lennard(float p1[], float p2[]){
  float d, force;
  float[] force_xy = new float[2];
  d = distance(p1, p2);
  force = -30 * pow(d,-13) + 30 * pow(d,-7);
  force_xy[0] = force * (p2[0] - p1[0])/d;
  force_xy[1] = force * (p2[1] - p1[1])/d;
  return force_xy;
}
```

粒子の座標 p1,p2 を引数とし，p1 に加わる力を返すようになっている．force_xy[0] は x 方向成分， force_xy[1] は y 方向成分である．Lennard-Jones ポテンシャルによる力の計算は粒子間距離が 1 の時に力が 0 になるように定数を決めている．粒子 p2 に加わる力は p1 に加わる力の逆向きになるので以下のような実装を行う．

```
void inter_force(int i, int j){
  float[] tmp= new float[2];
  tmp = lennard(ball_p[i], ball_p[j]);
  ball_f[i][0] += tmp[0];
  ball_f[i][1] += tmp[1];
  ball_f[j][0] += -tmp[0];
  ball_f[j][1] += -tmp[1];
}
```

２つの粒子の番号を引数とし，それらの粒子の座標を用いて関数 lennard を実行し，結果をそれぞれの ball_f に格納している．後に複数の粒子との合力を求めるために ball_f に対して加算代入 (+=) を行う．

5.3.2 合力の計算

```
for( i=0; i<n-1; i++){
  for( j=i+1; j<n; j++){
    inter_force(i, j);
  }
}
```

n 個の粒子に対して重複を持たないように 2 粒子組み合わせを全て選択し，関数 inter_force を実行している．これによりそれぞれの粒子の ball_f に，粒子間に加わる力が何度も加算代入され合力が求まる．

5.4 粒子の動作

粒子の座標の計算を行い，座標を更新することによって粒子を動かしていく．

5.4.1 Verlet 法による計算

Verlet 法を用いて粒子の座標を決定する．Verlet 法の計算は以下のように実装した．

```
float[] verlet(float current[], float previous[], float force[]){
  float[] newpos = new float[2];
  newpos[0]=2*current[0]-previous[0]+h*h/m*force[0];
  newpos[1]=2*current[1]-previous[1]+h*h/m*force[1];
  return newpos;
}
```

Verlet 法に従い，現在の座標 (current),h 時間前の座標 (previous), 粒子にかかっている力 (force) を引数として，h 時間後の座標 (newpos) を返すようになっている．h は時間刻み, m は質量であり，それぞれグローバル変数で定義している．

5.4.2 座標の更新

Verlet 法の計算結果から粒子の座標を更新する関数は以下のように実装した．

```
void calc_verlet(int i){
  float[] tmp= new float[2];
  if(solid_mode) ball_f[i][1]+= 0.00001/(h*h);//solid_mode
  tmp = ball_p[i];
  ball_p[i] = verlet(ball_p[i], pre_p[i],ball_f[i]);
  pre_p[i]= tmp;
}
```

i 番目の粒子に対して Verlet 法の計算を行い，計算結果から現在の座標 (ball_p) の更新を行う．Verlet 法を繰り返し行えるように過去の座標 (pre_p) の更新も行う必要があり，tmp を用いて更新前の ball_p を pre_p に格納している．

5.5 粒子の描画

粒子の座標が決定したので，粒子の描画を行う．また，粒子の速度によって色を変化させる．

```
void draw_particle(int i){
  float v;
  v = distance(ball_p[i], pre_p[i]);
  colorMode(HSB,360,100,100);
  fill(230+5000*v,100,100);
  ellipse(ball_p[i][0]*50,ball_p[i][1]*50, ball_size*100,ball_size*100);
}
```

5.5.1 描画

```
ellipse(ball_p[i][0]*50,ball_p[i][1]*50, ball_size*100,ball_size*100);
```

粒子の描画は ball_p*50 ピクセルの位置に円を描くというものになる．50 倍するのは ball_p の値の範囲が 0 から 10 であり，粒子の動作範囲が 500 ピクセルの正方形であるからである．ball_size は粒子のサイズを決定するグローバル変数である．

5.5.2 粒子の色

粒子の速度によって色を変化させる．色は速度が大きくなればなるほど，青から赤に変化するようにする．

```
float v;
v = distance(ball_p[i], pre_p[i]);
colorMode(HSB,360,100,100);
fill(230+5000*v,100,100);
```

粒子の現在の座標 (ball_p) と過去の座標 (pre_p) の距離を速度と考え，それに応じて色を変化させるようにしている．カラーモードを HSB に変更することによって，H(色相) の調整のみで青から赤に変化させる事が可能になる．

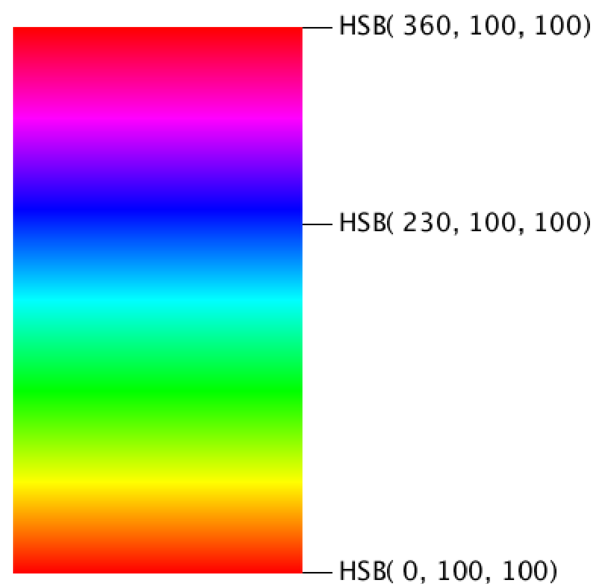

図 5.1: HSB による色の変化.

図 5.1 は S(彩度)=100,B(明度)=100 の状態で H(色相) を変化させたときの色の変化を表している．粒子の色は 230 から 360 までの値をとり，青から赤に徐々に変化することが確認できる．

5.6 壁との衝突

粒子が動作範囲内の枠内を超えた場合，衝突判定を行い反射させる必要がある．判定を行う関数 reflect は以下のように実装した．

```
void reflect(int i){
  if(ball_p[i][0]<ball_size){
  ball_p[i][0] = ball_size*2-ball_p[i][0];
  pre_p[i][0] = ball_size*2-pre_p[i][0];
  }
  else if(ball_p[i][0]>10-ball_size){
    ball_p[i][0] = (10-ball_size)*2-ball_p[i][0];
    pre_p[i][0] = (10-ball_size)*2-pre_p[i][0];
  }

  if(ball_p[i][1]<ball_size){
    ball_p[i][1] = ball_size*2-ball_p[i][1];
    pre_p[i][1] = ball_size*2-pre_p[i][1];
  }
  else if(ball_p[i][1]>10-ball_size && solid_mode)
   ball_p[i][1]=10-ball_size-0.01;
  else if(ball_p[i][1]>10-ball_size){
    ball_p[i][1] = (10-ball_size)*2-ball_p[i][1];
    pre_p[i][1] = (10-ball_size)*2-pre_p[i][1];
  }
}
```

粒子 i について壁との衝突判定を行い，反射させている．衝突判定は上下左右の壁と行う必要があり，if 文でそれぞれ行っている．粒子の動きは Verlet 法で定めているので，現在の座標 (ball_p) だけを更新しても，それ以降の Verlet 法の計算に支障がでてしまう．そのため反射角が合うように過去の座標 (pre_p) の更新も行う．

図 5.2 は衝突前後の粒子の座標を表しており，赤い粒子が衝突判定前，緑の粒子が衝突判定後を示しており，黄と青の線はそれぞれの粒子が壁と同じ距離であることを示している．このような座標の変換をすることによって自然な衝突を表現できる．

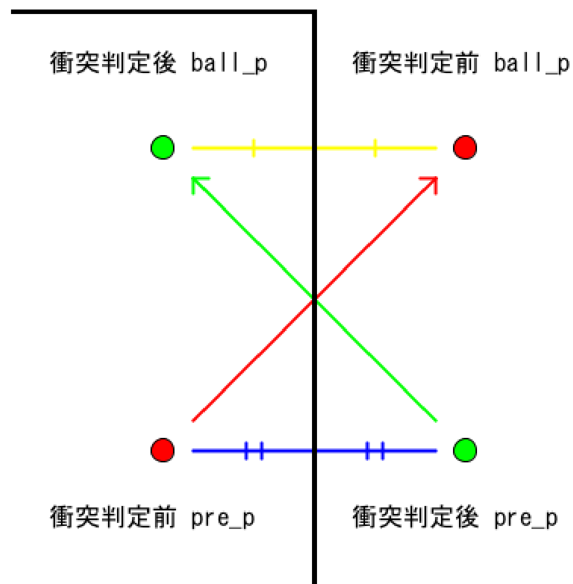

図 5.2: 衝突判定時の座標の変換.

5.7　選択した粒子に力を加える

クリックされた粒子にドラッグした方向の力を加えるようにし，ドラッグの距離によって力の大きさを変えるようにする．以下のように実装した．

```
int click_ball=0;

void mousePressed() {
  int i;
  for(i=0; i<n; i++){
    if(mouseX-10-20<=ball_p[i][0]*50 & mouseX-10+20>= ball_p[i][0]*50 &
    mouseY-10-20<=ball_p[i][1]*50 & mouseY-10+20>= ball_p[i][1]*50){
      click_ball = i;
      fill(0);
      //Ver Processing
      ellipse(pre_p[i][0]*50,pre_p[i][1]*50,ball_size*100,ball_size*100);
      /*Ver JavaScript
      ellipse(10+pre_p[i][0]*50,10+pre_p[i][1]*50,
       ball_size*100,ball_size*100);
       */
      mouse_p[0] = mouseX;
      mouse_p[1] = mouseY;
      return;
    }
  }
  click_ball = -1;
}
void mouseReleased(){
  if(click_ball != -1){
    ball_f[click_ball][0] +=( mouseX - mouse_p[0])*0.0001/(h*h);
    ball_f[click_ball][1] += (mouseY - mouse_p[1])*0.0001/(h*h);
  }
}
```

5.7.1 粒子の選択

粒子がクリックされているかの判定を行い，クリックされていた場合，何番目の粒子か確認する必要がある．また，選択された粒子がわかりやすいように黒く描画する．

```
void mousePressed() {
  int i;
  for(i=0; i<n; i++){
    if(mouseX-10-20<=ball_p[i][0]*50 & mouseX-10+20>= ball_p[i][0]*50 &
    mouseY-10-20<=ball_p[i][1]*50 & mouseY-10+20>= ball_p[i][1]*50){
      click_ball = i;
      fill(0);
      //Ver Processing
      ellipse(pre_p[i][0]*50,pre_p[i][1]*50,ball_size*100,ball_size*100);
      /*Ver JavaScript
      ellipse(10+pre_p[i][0]*50,10+pre_p[i][1]*50,
       ball_size*100,ball_size*100);
       */
      mouse_p[0] = mouseX;
      mouse_p[1] = mouseY;
      return;
    }
  }
  click_ball = -1;
}
```

マウスボタンが押下された際に，i 番目の粒子の上にマウスポインタが重なっているかの判定を行う．これを n 個の粒子全てに対して行っていき

- 重なりを確認した場合
  1. click_ball に粒子の番号を格納する．
  2. その粒子を黒く描画する．
  3. マウスポインタの座標を mouse_p に格納する．
  4. 関数 mouse_Pressed を終了する．

- 粒子全てに対して重なりを確認できなかった場合
    - click_ball に-1 を格納する．

この処理により click_ball には選択された粒子の番号か-1 が格納されている．また粒子が選択された場合，mouse_p にはマウスポインタの座標が格納されている．これらの値はマウスが離された際の処理で利用する事になる．また Processing と JavaScript での動作に異なりが生じるため変換を行う際は，プログラム内のコメントアウト部分の書き換えを行う必要がある．

5.7.2 ドラッグ後の処理

```
void mouseReleased(){
  if(click_ball != -1){
    ball_f[click_ball][0] +=( mouseX - mouse_p[0])*0.0001/(h*h);
    ball_f[click_ball][1] += (mouseY - mouse_p[1])*0.0001/(h*h);
  }
}
```

マウスボタンが押された際に格納しておいた click_ball,mouse_p を利用する．マウスボタンが離された際に click_ball が-1 以外 (粒子が選択されている状態) であれば，ドラッグされた距離に応じて ball_f[click_ball] に力を加える．このような実装を行う事でマウスボタンを用いて粒子を扱うことが可能になる．また，格納する力の値を h^2 で割っているのは時間刻み h の大きさに合わせて力の大きさを調整する必要があるからである．

5.8　キー入力

```
boolean solid_mode = false;
boolean force_mode = false;

void keyPressed() {
  if (key == 's'||key == 'S') {
    if(solid_mode) solid_mode = false;
    else solid_mode = true;
  }
      if (key == 'f'||key == 'F') {
    if(force_mode) force_mode = false;
    else force_mode = true;
  }
  if (key == 'r'||key == 'R') set_particle();
}
```

boolean 変数がキー入力に対して true,false と切り替わり，モード変更が可能になる．今回はS キーで凝固状態 (solid_mode),F キーで力を視覚化した状態 (force_mode) の切り替えを行う．また R キーはリセットに対応していて，粒子の状態を初期化する関数 set_particle を実行する．

5.9　凝固状態

凝固の現象を擬似的に表すために粒子に常に下向きの力を加え下層部に溜まるようにする．また，枠の下面部に粒子が衝突した際の反発力を減らす処理を行う．

関数 calc_verlet 内　下向きの力を加える処理

```
if(solid_mode) ball_f[i][1]+= 0.00001/(h*h);
```

関数 reflect 内　下面部の枠の反発力を減らす処理

```
else if(ball_p[i][1]>10-ball_size && solid_mode)
 ball_p[i][1]=10-ball_size-0.01;
```

これらの処理は solid_mode が true の場合実行される．下面部の反発力を減らす処理では,

粒子が下面部を通り越した際に，粒子の y 座標に下面部と接触するような y 座標を格納している．これにより衝突後の反発力を減らすことができる．この処理は粒子が壁と接触するように座標を決定するため反発力を 0 にしているように思えるが，継続的に下向きの力が加えられた状態で verlet 法での計算が行われるため粒子に多少の上向きの動きが生じるようになる．また，これを以下のように書き換えると反発力を 0 にすることができる．

```
else if(ball_p[i][1]>10-ball_size && solid_mode) {
  ball_p[i][1]=10-ball_size-0.01;
  pre_p[i][1]=10-ball_size-0.01;
}
```

反発力を 0 にしてしまうと粒子が最下層に詰まってしまい粒子間距離が近すぎてしまう．この振る舞いは凝固の表現には不適切である．そのため，反発力を 0 にせずに少し減らす処理を選択した．

5.10 力を視覚化した状態

粒子に加わる力を線を用いて表す関数 draw_force は以下のように実装した．

```
void draw_force(int i){
  float force_sqrt_x, force_sqrt_y;
  strokeWeight(3);
  if(ball_f[i][0]>0) force_sqrt_x = sqrt(abs(ball_f[i][0]));
  else force_sqrt_x = -sqrt(abs(ball_f[i][0]));
  if(ball_f[i][1]>0) force_sqrt_y = sqrt(abs(ball_f[i][1]));
  else force_sqrt_y = -sqrt(abs(ball_f[i][1]));
  line(ball_p[i][0]*50, ball_p[i][1]*50,
   ball_p[i][0]*50+force_sqrt_x, ball_p[i][1]*50+force_sqrt_y);
  strokeWeight(1);
}
```

粒子 i の座標 (ball_p[i][0]*50, ball_p[i][1]*50) を始点とし，始点+粒子に加わっている力の根号 (ball_p[i][0]*50+force_sqrt_x ,ball_p[i][1]*50+force_sqrt_y) まで太めの線 (strokeWeight(3)) の描画を行う．粒子に加わっている力は振れ幅が大きいため根号を用いている．これによ

り 1 以下の小さな値の変化が確認しやすくなる．粒子に加わる力には負の値も存在するが，根号の計算では負の値を扱うことができない．そのため，粒子に加わる力の絶対値から根号を求め，その値の符号を元の値と同一にする必要がある．sqrt(abs(ball_f[i][0])) は x 方向に加わる力の絶対値の根号であり，粒子に加わる力の符号と同じになるように force_sqrt_x に格納している．今回は線の長さに根号を用いているが，値をそのまま利用する方法もある．この方法には力の大きさが長さに比例するという利点があり，以下のように書き換えればよい．

```
line(ball_p[i][0]*50,ball_p[i][1]*50,
 ball_p[i][0]*50+ball_f[i][0],ball_p[i][1]*50+ball_f[i][1]);
```

5.11 スライダー

```
float slider_value = 1;
boolean slider_dragged = false;

void slider(int position_x, int position_y, int slider_height,
  int slider_width, int min, int max, int NumberOfTickMarks){

  int num_separator =
    (int)map(slider_value, min, max, 0, NumberOfTickMarks-1);
  float cordinate_y = position_y + slider_height -
    ((float)slider_height / (NumberOfTickMarks-1))*num_separator;
  fill(255);
  rect(position_x, position_y, slider_width, slider_height);
  fill(0);
  rect(position_x, cordinate_y, slider_width, position_y +
    slider_height - cordinate_y);

  text( slider_value, position_x, position_y + slider_height + 20);



  if(mouseX >= position_x & mouseX <= position_x + slider_width &
    mouseY<=cordinate_y+10 & mouseY>= cordinate_y-10){
    if (mousePressed){
      slider_dragged= true;
    }
   }
  if(mousePressed==false) slider_dragged=false;

   if(slider_dragged){
     num_separator+= (int)((cordinate_y - mouseY)/
       ((float)slider_height / (NumberOfTickMarks-1)));
     num_separator = constrain(num_separator, 0, NumberOfTickMarks-1);
     slider_value=map(num_separator, 0, NumberOfTickMarks-1, min, max);
   }
}
```

このプログラムを利用するためにはスライダーの値を格納する変数 slider_value とスライダーのクリック判定を行う変数 slider_dragged をグローバル変数で定義する必要がある．関数 slider の引数はスライダーの左上の座標を決定する position_x,position_y, 縦と横の幅を決定する slider_height, slider_width, 値の最小値，最大値を決定する min, max, 区切りの数を決定する NumberOfTickMarks となり，それらを宣言することでスライダーを利用できる．変数 num_separator にはスライダーの一番下を 0 番目として区切りが下から何番目なのかを格納している．この整数の値を利用することによってマウスをドラッグした際のスライダーの区切りを表現することができる．以下にプログラムの処理の流れを記す．

1. スライダーの値 (slider_value) から，区切りが下から何番目か (num_separator) を計算する．
2. 区切りの番号から，スライダー部分の y 座標 (cordinate_y) を計算する．
3. スライダーを描画する．
4. スライダーの掴み判定 (slider_dragged) を行う．
5. 掴み判定が true の場合
    1. マウスの移動距離に合わせて区切りの番号に値を加える．
    2. 区切りの番号からスライダーの値を計算し格納する．

第6章　総括

本研究では，Web ブラウザ上で Interactive に操作可能な，分子動力学法による粒子の振る舞いを視覚化するプログラムを作成した．このプログラムによる成果を以下に記す．

1. 分子動力学法による粒子の動きをシミュレーションし視覚化することによって，クラスタや凝固などの現象を視認することができ直感的な理解が可能となった．また，粒子をマウスで操作できるため，自分の好きなようにシミュレーションでき理解の向上に繋がると考えられる．

2. プログラムの JavaScript 化を行い Web ブラウザ上で動作が可能になり容易に公開，利用することができる．そのため，分子動力学法を学習する者が手軽に扱うことができ，学習意欲，効率の向上に繋がると考えられる．

3. 作成したプログラムをライブラリとして保存しプログラムの解説を行うことで，プログラムの継続的な発展を可能にした．また，今後シミュレーションプログラムを作成する際に今回のプログラムをライブラリとして利用し効率よく作成する事が可能である．

謝辞

本研究を行うにあたり，終始多大なるご指導，御鞭撻をいただいた西谷滋人教授に対し，深く御礼申し上げます．また，本研究の進行に伴い，様々な助力，知識の供給を頂きました西谷研究室の同輩，先輩方に心から感謝の意を示します．本当にありがとうございました．

付 録A　プログラムのソースコード

```
float[][] ball_p = new float[81][2];
float[][] pre_p = new float[81][2];
float[][] ball_f  = new float[81][2];
float[] mouse_p =new float[2];
float ball_size = 0.2;
float m=0.2;
float h=0.001;
int n = 9;
float slider_n = n;
boolean slider_dragged = false;
boolean slider2_dragged = false;

int click_ball=0;
boolean solid_mode = false;
boolean force_mode = false;

void setup(){
  size(800, 550);
  set_particle();
}

void set_particle(){
  int i, j, k=0;
  for(i=1; i<10; i++){
    for(j=1; j<10; j++){
      if(k==n) break;
      ball_p[k][0] = i;
```

```
      ball_p[k][1] = j;
      pre_p[k][0] = i;
      pre_p[k][1] = j;
      k++;
    }
  }

  for(i=0; i<n; i++){
    ball_f[i][0] = 0;
    ball_f[i][1] = 0;
  }
}

float distance(float[] a, float[] b){
  return sqrt( sq(a[0]-b[0]) + sq(a[1]-b[1]) );
}

float[] lennard(float p1[], float p2[]){
  float d, force;
  float[] force_xy = new float[2];
  d = distance(p1, p2);
  force = -30 * pow(d,-13) + 30 * pow(d,-7);
  force_xy[0] = force * (p2[0] - p1[0])/d;
  force_xy[1] = force * (p2[1] - p1[1])/d;
  return force_xy;
}

float[] verlet(float current[], float previous[], float force[]){
  float[] newpos = new float[2];
  newpos[0]=2*current[0]-previous[0]+h*h/m*force[0];
  newpos[1]=2*current[1]-previous[1]+h*h/m*force[1];
  return newpos;
}
```

```
void reflect(int i){
  if(ball_p[i][0]<ball_size){
  ball_p[i][0] = ball_size*2-ball_p[i][0];
  pre_p[i][0] = ball_size*2-pre_p[i][0];
  }
  else if(ball_p[i][0]>10-ball_size){
    ball_p[i][0] = (10-ball_size)*2-ball_p[i][0];
    pre_p[i][0] = (10-ball_size)*2-pre_p[i][0];
  }

  if(ball_p[i][1]<ball_size){
    ball_p[i][1] = ball_size*2-ball_p[i][1];
    pre_p[i][1] = ball_size*2-pre_p[i][1];
  }
  else if(ball_p[i][1]>10-ball_size && solid_mode)
    ball_p[i][1]=10-ball_size-0.01;
  else if(ball_p[i][1]>10-ball_size){
    ball_p[i][1] = (10-ball_size)*2-ball_p[i][1];
    pre_p[i][1] = (10-ball_size)*2-pre_p[i][1];
  }
}

void draw_particle(int i){
  float v;
  v = distance(ball_p[i], pre_p[i]);
  colorMode(HSB,360,100,100);
  fill(230+5000*v,100,100);
  ellipse(ball_p[i][0]*50,ball_p[i][1]*50, ball_size*100,ball_size*100);
}

void inter_force(int i, int j){
  float[] tmp= new float[2];
  tmp = lennard(ball_p[i], ball_p[j]);
  ball_f[i][0] += tmp[0];
```

```
  ball_f[i][1] += tmp[1];
  ball_f[j][0] += -tmp[0];
  ball_f[j][1] += -tmp[1];
}

void calc_verlet(int i){
  float[] tmp= new float[2];
  if(solid_mode) ball_f[i][1]+= 0.00001/(h*h);
  tmp = ball_p[i];
  ball_p[i] = verlet(ball_p[i], pre_p[i],ball_f[i]);
  pre_p[i] = tmp;
}

void force_reset(int i){
  ball_f[i][0] = 0;
  ball_f[i][1] = 0;
}

void draw_force(int i){
  float force_sqrt_x, force_sqrt_y;
  strokeWeight(3);
  if(ball_f[i][0]>0) force_sqrt_x = sqrt(abs(ball_f[i][0]));
  else force_sqrt_x = -sqrt(abs(ball_f[i][0]));
  if(ball_f[i][1]>0) force_sqrt_y = sqrt(abs(ball_f[i][1]));
  else force_sqrt_y = -sqrt(abs(ball_f[i][1]));
  line(ball_p[i][0]*50, ball_p[i][1]*50,
    ball_p[i][0]*50+force_sqrt_x, ball_p[i][1]*50+force_sqrt_y);
  strokeWeight(1);
}

void slider(int position_x, int position_y, int slider_height,
  int slider_width, int min, int max, int number_tickmarks){

  int num_separator = (int)map(slider_n, min, max, 0, number_tickmarks-1);
```

```
  float cordinate_y = position_y + slider_height -
    ((float)slider_height / (number_tickmarks-1))*num_separator;
  fill(255);
  noStroke();
  rect(position_x, position_y + slider_height, slider_width+50, 30);
  stroke(1);
  rect(position_x, position_y, slider_width, slider_height);
  fill(0);
  rect(position_x, cordinate_y, slider_width, position_y +
    slider_height - cordinate_y);

  text( "N="+nf(slider_n,1,0), position_x, position_y + slider_height + 20);
  if(mousePressed==false) slider_dragged=false;
  if(mouseX >= position_x & mouseX <= position_x + slider_width &
    mouseY<=cordinate_y+10 & mouseY>= cordinate_y-10){
    if(mousePressed){
      slider_dragged= true;
    }
   }

   if(slider_dragged){
     num_separator+= (int)((cordinate_y - mouseY)/
       ((float)slider_height / (number_tickmarks-1)));
     num_separator = constrain(num_separator, 0, number_tickmarks-1);
     slider_n=map(num_separator, 0, number_tickmarks-1, min, max);
   }
}

void slider2(int position_x, int position_y, int slider_height,
  int slider_width, float min, float max, int number_tickmarks){

  int num_separator =
    (int)map(h, min, max, 0, number_tickmarks-1);
  float cordinate_y = position_y + slider_height -
```

```
    ((float)slider_height / (number_tickmarks-1))*num_separator;
  fill(255);
  noStroke();
  rect(position_x, position_y + slider_height, slider_width+50, 30);
  stroke(1);
  rect(position_x, position_y, slider_width, slider_height);
  fill(0);
  rect(position_x, cordinate_y, slider_width, position_y +
    slider_height - cordinate_y);

  text( "h="+nf(h,1,4), position_x, position_y + slider_height + 20);

  if(mousePressed==false) slider2_dragged=false;
  if(mouseX >= position_x & mouseX <= position_x + slider_width &
    mouseY<=cordinate_y+10 & mouseY>= cordinate_y-10){
    if(mousePressed){
      slider2_dragged= true;
    }
   }

   if(slider2_dragged){
     num_separator+= (int)((cordinate_y - mouseY)/
       ((float)slider_height / (number_tickmarks-1)));
     num_separator = constrain(num_separator, 0, number_tickmarks-1);
     h=map(num_separator, 0, number_tickmarks-1, min, max);
   }
}

void mousePressed() {
  int i;
  for(i=0; i<n; i++){
    if(mouseX-10-20<=ball_p[i][0]*50 & mouseX-10+20>= ball_p[i][0]*50 &
    mouseY-10-20<=ball_p[i][1]*50 & mouseY-10+20>= ball_p[i][1]*50){
      click_ball = i;
```

```
      fill(0);
      //Processing
      ellipse(pre_p[i][0]*50,pre_p[i][1]*50,ball_size*100,ball_size*100);

      /*JavaScript
      ellipse(10+pre_p[i][0]*50,10+pre_p[i][1]*50,
        ball_size*100,ball_size*100);
      */
      mouse_p[0] = mouseX;
      mouse_p[1] = mouseY;
      return;
    }
  }
  click_ball = -1;
}

void mouseReleased(){
  if(click_ball != -1){
    ball_f[click_ball][0] +=( mouseX - mouse_p[0])*0.0001/(h*h);
    ball_f[click_ball][1] += (mouseY - mouse_p[1])*0.0001/(h*h);
  }
}

void keyPressed() {
  if (key == 's'||key == 'S') {
    if(solid_mode) solid_mode = false;
    else solid_mode = true;
  }
      if (key == 'f'||key == 'F') {
    if(force_mode) force_mode = false;
    else force_mode = true;
  }
  if (key == 'r'||key == 'R') set_particle();
}
```

```
void draw(){
  colorMode(RGB,256);
  int i;
  int j;
  translate(10,10);
  if (mousePressed == false){
    if(n!=slider_n) {
      n=(int)slider_n;
      set_particle();
    }
    background(255);
    text("Press R to Reset  ",550,35);
    text("Press S to Solid-Mode   "+solid_mode,550,50);
    text("Press F to Force-Mode  "+force_mode,550,65);
    fill(255,255,255);
    rect(0, 0, 500, 500);



    for( i=0; i<n; i++){
      reflect(i);
      draw_particle(i);
    }

    for( i=0; i<n-1; i++){
      for( j=i+1; j<n; j++){
        inter_force(i, j);
      }
    }

    for( i=0; i<n; i++){
      if(force_mode){
        draw_force(i);
      }
      calc_verlet(i);
```

```
    }

    for(i=0; i<n; i++){
      force_reset(i);
    }

  }

  translate(-10, -10);
  colorMode(RGB,256);
  slider(600, 100, 300, 20, 1, 81, 81);
  slider2(700, 100, 300, 20, 0.0001, 0.01, 100);
  translate(10, 10);
}
```

参考文献

[1] 神山新一著, 佐藤明,「分子動力学シミュレーション」(朝倉書店,1997).

[2] 西谷滋人著,「固体物理の基礎」(森北出版,2006).