青山学院大学 理工学部
物理学科 卒業論文

表面成長のモデルの定量的解析

青木佑介 著

表面成長のモデルの定量的解析

青木佑介

羽田野研究室

平成 14 年 2 月 20 日

概 要

表面の成長過程をスケーリングを用いて定量的に解析し、そこから得られた指数と理論値を比較した。その結果、両者はほぼ一致した。また本研究では新しいモデルの解析も行った。異なった 2 つのモデルのスケーリング指数を比較した結果、それらは同じユニヴァーサリティクラスであった。

目 次

1　はじめに

ブラウン運動で知られるコロイド粒子には、粘土などの疎水コロイドとたんぱく質やでんぷんなどで知られる親水コロイドの２種類がある。これらのコロイド粒子は正または負に帯電しており、電気的に引き合っている。このような性質を持った粒子はどのように沈着するだろうか？
その現象のモデルとして導入されたのが、Ballistic deposition と呼ばれる表面成長モデルである。

本研究ではこのモデルをシミュレーションすることにより、このモデルにおいて粒子が作る表面の成長過程を観察する。また、表面のゆらぎ、飽和時間からスケーリング指数を求め、モデルを定量的に解析する。また新しいモデルとして、非等方的な粒子を想定した Domino model を提案し、２つのモデルを定量的に比較する。異なった２つのモデルのスケーリング指数を比較して、ユニヴァーサリティが保たれているかを調べた。

その結果、シミュレーションから得られた Ballistic deposition のスケーリング指数と KPZ 方程式から導かれる Ballistic deposition の理論値 [1] はほぼ一致した。また Ballistic deposition と Domino model のスケーリング指数を比較した結果、これらは同じユニヴァーサリティクラスであった。

第２章では表面成長モデルの説明をする。第３章ではスケーリングとそれを用いて得られるスケーリング指数の説明をし、第４章でシミュレーション結果を述べる。

2　モデルの説明

この章ではまず表面成長のモデルを説明をし、今回シミュレーションを行った Ballistic deposition と Domino model を定義する。

2.1　表面成長のモデルとは？

表面成長のモデルとは、何もない表面の上に粒子を垂直に落としていくモデルのことである。その時粒子を落とす位置 x はランダムに選ぶものとする（図 1）。横軸 x は粒子を落とす位置を表し、縦軸 h は粒子の堆積した高さを表す。また扱っている系の大きさを L とする。例として、$L = 5$ のときと $L = 7$ のときを図 2 に示す。

次に本研究で行った Ballistic deposition と Domino model について説

明する。

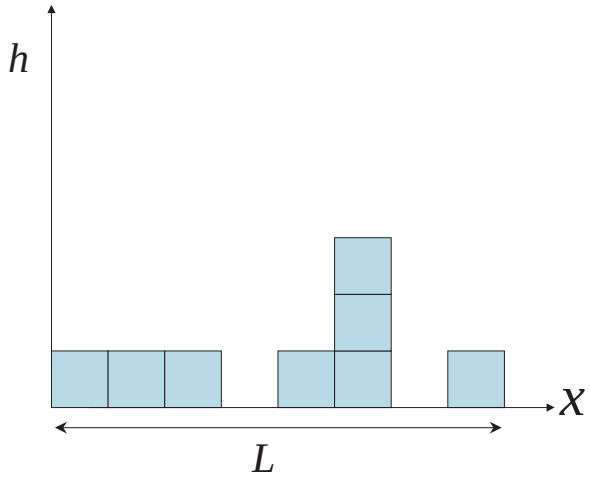


図 1: 表面成長のモデルの例。

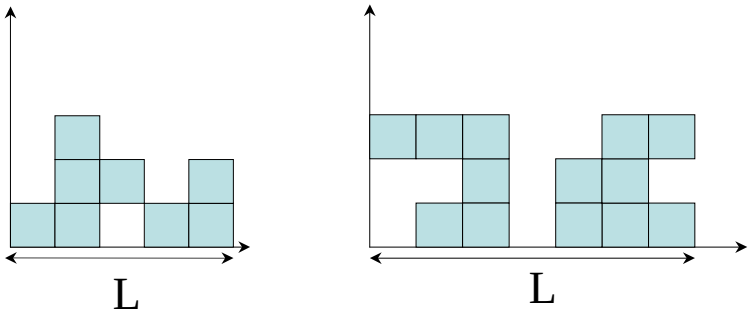


図 2: 左図が $L = 5$ のとき、右図が $L = 7$ のときの例。

2.2 Ballistic deposition

コロイド粒子の堆積の仕方を表した Ballistic deposition は、コロイド粒子の持つ性質、つまりお互い電気的に引き合っているという性質をうまく再現している。つまり落ちていく粒子が堆積する際に、次のようなルール (rule1) [2] を与えている（図 3)。

rule1:

(1) ある粒子 A を垂直に表面の上に落としていく途中に、もし両隣りに粒子がなければ、粒子 A はただ垂直に落ちていく。

(2) ある粒子 B を垂直に表面の上に落としていく途中に、もしその両隣りのどちらか一方にでも粒子が存在した場合、粒子 B は隣の粒子と最初に接触した地点で止まってしまう。

このルールに基づいて粒子を落としていくのが、Ballistic deposition である。

2.3 Domino model

前節では、コロイド粒子のモデルとして導入された Ballistic deposition について紹介した。一方、これから紹介する Domino model というモデ

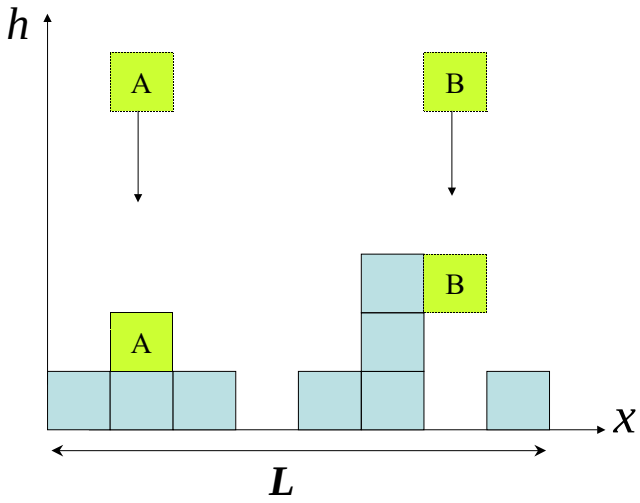


図 3: Ballistic deposition の堆積ルール。

ルは本研究で考えた新しいモデルである。

この Domino model と前節で説明した Ballistic deposition の違う点は落とす粒子の形にある。先程の Ballistic deposition では落としていく粒子、つまりコロイド粒子の形はすべて正方形だが、Domino model ではその正方形を 2 つくっつけたブロックを落としていく（図 4）。粒子の堆積の仕方は前節の Ballistic deposition のルール（rule1）と同じものを採用し、ランダムに落としていく。

ではここでなぜ今回このモデルについて研究し始めたのか、その意義を述べる。前節の Ballistic deposition はコロイド粒子のモデルとして導入されているが、すべてのコロイド粒子が Ballistic deposition のように等方的な丸い粒子ばかりではないと思われる。そこで、非等方的な細長い粒子を想定して Domino model を考案した。

3　スケーリング指数の決定

この章では、まず表面成長モデルを定量的に解析する上で必要な物理量を定義し、そこから本研究の目的であるスケーリング指数を決定するに至るまでのプロセスを示す。

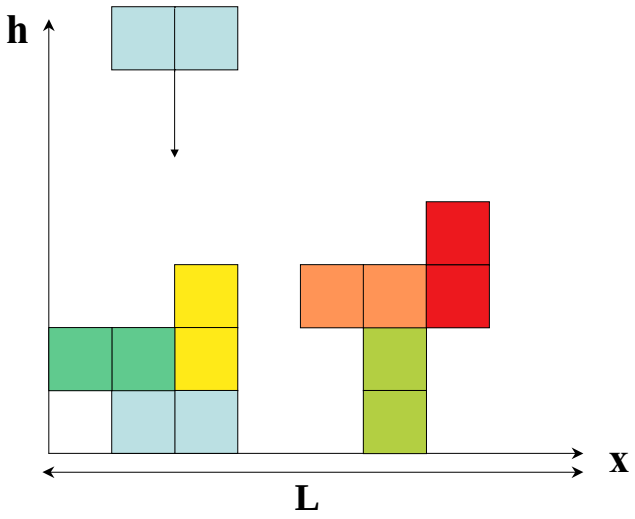


図 4: Domino model の堆積ルール。

3.1 表面のゆらぎ

まず最初に表面を特徴づける量である w の説明をする。この w は表面の普遍的な性質を調べるために必要な量で、

$$w = \sqrt{\frac{1}{L}\sum_{x=1}^{L}[h(x,t) - \bar{h}(t)]} \tag{1}$$

と定義される。但し、$h(x,t)$ は時刻 t の位置 x における堆積した粒子の高さを表し、$\bar{h}(t)$ は時刻 t における平均的な高さを表している。

この式からもわかるように、表面を特徴づける量 w は表面のゆらぎそのものを表している。またこの式の右辺では $h(x,t)$ と $\bar{h}(t)$ の差をとっているので、表面が粗くでこぼこな場合ほど w はより大きくなるということがわかる。

3.2 飽和時間

前節で説明した w と t を両対数グラフにプロットすると、多くの場合図 5 のような飽和曲線が得られることがわかっている。このグラフには大きく分けて 2 つの状態が存在する。まず、何もない平面に堆積するこ

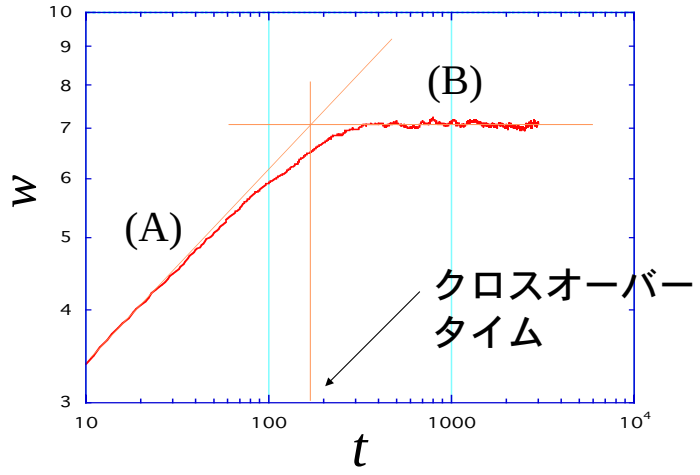


図 5: 飽和曲線。式 (1) の量 w の時間依存性を対数プロットしたもの。

とによって次第に表面が粗くなり、w は t と共に増大する（状態 A)。しかし、やがて w は飽和し、一定値に収束する（状態 B)。飽和する前の状態 A から飽和した状態 B に移り変わる時間のことを飽和時間またはクロスオーバータイム $t_\times$ と呼ぶ。具体的には、飽和する前の状態を近似した直線と飽和した状態を近似した直線が交わったところを、飽和し始める時間 $t_\times$ とする。

3.3 スケーリング指数

ここでモデルを定量的に解析するためのスケーリング指数についての説明をする。

3.3.1 growth exponent : β

図 5 の飽和曲線では、飽和する前の状態の w は時間 t のベキ乗で増加している。そこでこの指数を β と定義する。つまり、

$$w(L,t) \sim t^{\beta} \qquad (t \ll t_\times \text{のとき}) \tag{2}$$

とする。この β は growth exponent と呼ばれ、表面が粗くなっていく過程の時間依存性を特徴づけている。

3.3.2 roughness exponent : α

一方、飽和した状態における w は時刻 t ではなく、システムサイズ L のベキ乗で増加していることがわかっている [2]。そこでこの指数を α と定義する。つまり、

$$w_{\mathrm{sat}}(L) \equiv \lim_{t\to\infty} w(L,t) \tag{3}$$

$$\sim L^{\alpha} \qquad (t \gg t_{\times} \text{のとき}) \tag{4}$$

とする。この α は roughness exponent と呼ばれ、これは飽和した表面の粗さを特徴づけている。

3.3.3 dynamic exponent : z

またクロスオーバータイム $t_{\times}$ とシステムサイズ L の間には、

$$t_{\times} \sim L^{z} \tag{5}$$

の関係式が成り立っていることがわかっている [2]。この指数 z は dynamic exponent と呼ばれている。

以上の 3 つのスケーリング指数 (α, β, z) を求めることで、表面成長のモデルを定量的に解析することができる。

3.4 スケーリング則

前節で紹介した 3 つのスケーリング指数は互いに独立ではなく、スケーリング則でつながっている。この節ではまずスケーリング関係式を導出し、そこからスケーリング則を導き出す。

スケーリングの手順は以下の通りである [2]。

(i) まず例として Ballistic deposition のシステムサイズが $L = 200$, 400, 600 のときの w の時間依存性を対数プロットする (図 3.4(a))。

(ii) 次に図 3.4(a) のグラフの縦軸を w から w/w_{sat} に変換する (図 6(b))。

この操作により log-log スケールにおいてグラフは垂直にシフトする。つまり $w/w_{\rm sat}$ の時間依存性を対数プロットすると、系の大きさに依らず同じ値で飽和した曲線が得られる。

(iii) 図 6(b) のグラフの横軸を t から $t/t_\times$ に変換する (図 6(c))。

この操作により log-log スケールにおいてグラフは平行にシフトする。つまり $t/t_\times$ と $w/w_{\rm sat}$ の関係を両対数プロットすると、系の大きさに依らず同じ値かつ同じ時間で飽和状態に達する。システムサイズ L の値により最初はそれぞれ異なっていた飽和曲線が、スケーリングを行なうことで一致した。これはスケーリング仮説が正しいということが数値的に証明できたことになる。

図 6(c) より $w/w_{\rm sat}$ は $t/t_\times$ の関数であることがわかる。よって

$$\frac{w(L,t)}{w_{\rm sat}(L)} \sim f\left(\frac{t}{t_\times}\right) = f(u) \tag{6}$$

となる。但し、$\mathrm{u} \equiv t/t_\times$ である。この $f(u)$ をスケーリング関数という。

更に式 (4)、(5) を代入すると、Family-Vicsek のスケーリング則 [3]、

$$w(L,t) \sim L^\alpha f\left(\frac{t}{L^z}\right) \tag{7}$$

が得られる。

このスケーリング関数 $f(u)$ についても、前節と同じことが言える。まず u が小さいとき、つまり飽和する前の状態のとき、スケーリング関数 $f(u)$ は u のベキ乗で増加している。この傾きは β である。なぜならこのスケーリングが、曲線を回転させる操作ではなく並進操作のみで実行されるからであり、傾きはスケーリング前後で変化しない。よって、

$$f(u) \sim u^\beta \qquad (u \ll 1 \text{ のとき}) \tag{8}$$

がいえる。一方、u が大きいとき、スケーリング関数 $f(u)$ は飽和状態なので

$$f(u) = \text{const} \qquad (u \gg 1 \text{ のとき}) \tag{9}$$

がいえる。

次にスケーリング則からスケーリング関係式を導出する。$t{=}t_\times$ のとき、w は式 (2) より、

$$w(t_\times) \sim t_\times^\beta \tag{10}$$

となる。一方、式 (4) より、

$$w(t_\times) \sim L^\alpha \tag{11}$$

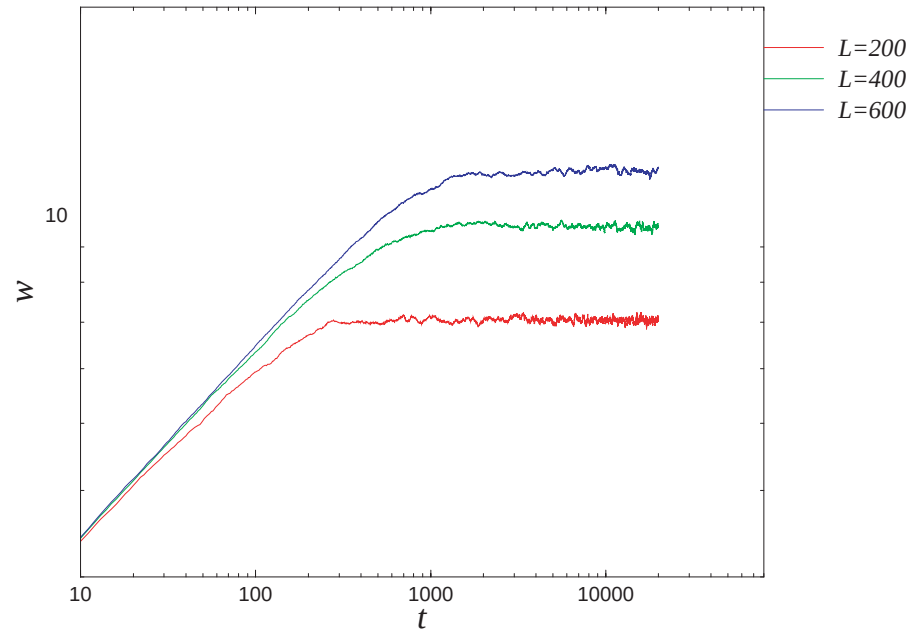
となる。すなわち、

$$t_\times^\beta \sim L^\alpha \tag{12}$$

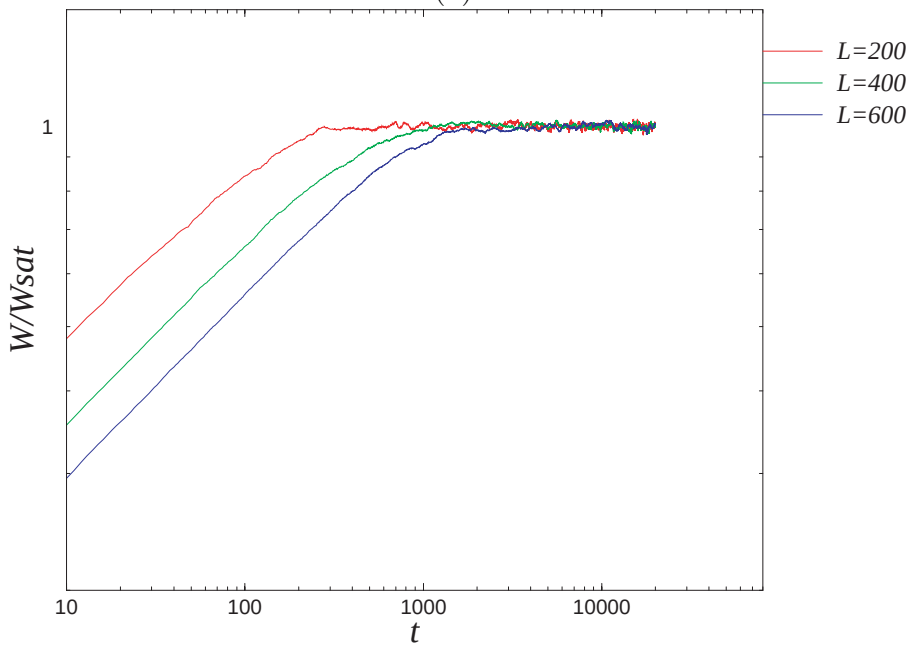
がいえ、式 (5) から 3 つのスケーリング指数には

$$z = \frac{\alpha}{\beta} \tag{13}$$

が成り立つ。これをスケーリング関係式という。これは式 (7) が成り立つ成長過程において役立つ [2]。

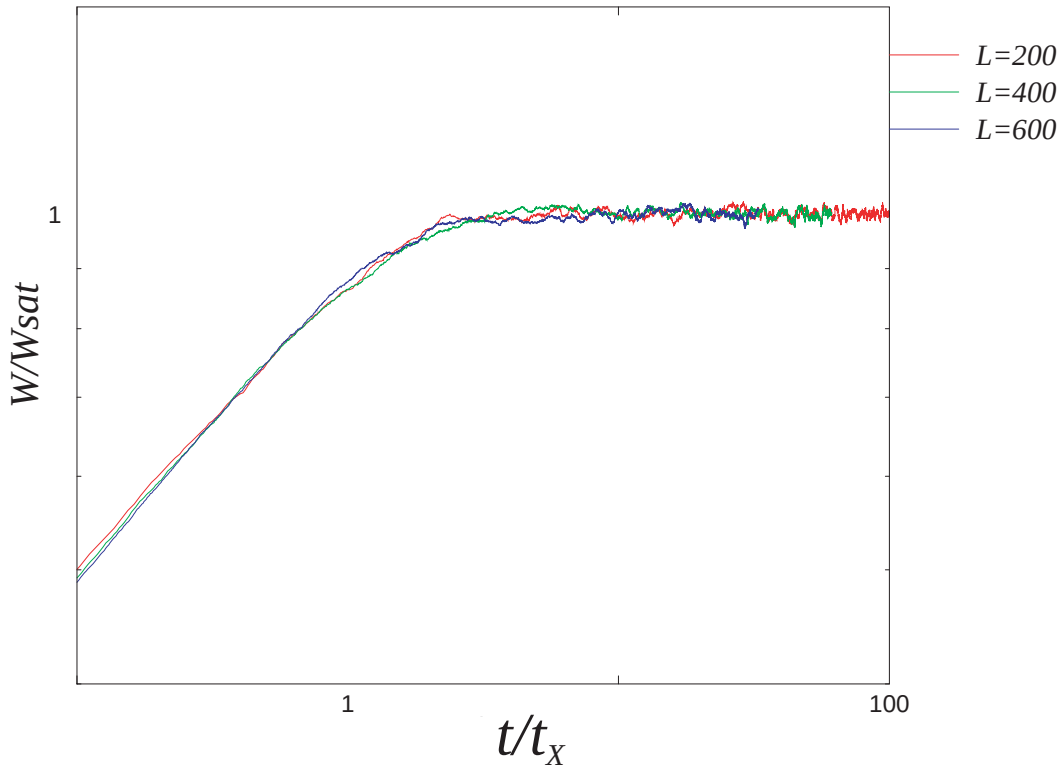


(a)



(b)

図 6: (a)：w の時間依存性を対数プロットしたもの。(b)：w/w_{sat} の時間依存性を対数プロットしたもの。(c)：$t/t_{\times}$ と w/w_{sat} の関係を両対数グラフにしたもの。



(c)

4　シミュレーションの結果と考察

２つのモデルについてのシュミレーション結果を次に示す。まず粒子を落として出来た表面を示す。その次に、スケーリング指数を示す。

4.1　表面

図 7 は 20,000 個の粒子を落として出来た Ballistic deposition の表面である。なお、系の両端には周期的境界条件を採用している。５つに色分けされた層は、20,000 個の粒子を５等分し、4,000 個の粒子が堆積する毎に層の色を変えていったものである。５つの表面を見比べると徐々にではあるが、時間が経つにつれて表面が粗くなっているのが見てとれる。これは 2.2 節で説明した粒子の堆積の仕方 (rule1) によるものと考えられる。

図 8 は先程の Ballistic deposition と同様、20,000 個の粒子を落として出来た Domino model の表面である。系の両端には周期的境界条件を採用した。図 7 の Ballistic deposition と比べると、表面も粗く、隙間の部分も大きくなっているのがわかる。このように、異なった２つのモデルの表面は定性的に異なることがわかった。

では２つのモデルは具体的にどこが違うのか？　Ballistic deposition よりも Domino model のほうが隙間が大きかった。このことより、Domino model のほうが KPZ 方程式（付録 A 参照）の λ の値が大きいと考えられる。言い換えるならば、このモデルは表面を引っ張り上げる力 F が強い。

表面を引っ張る力 F と本研究で採用した堆積ルール (rule1) は密接に繋がっている。rule1 は隣りの粒子との相互作用を考慮したものである。新しく落ちてきた粒子は両隣りにある粒子と同じ高さになるか、それよりも高くなる。このルールに基づいて、粒子はお互い相関しながら成長する。最初は局所的に堆積していた粒子だが、時間が経つにつれてその粒子の高さに関する情報が隣りの粒子を介して伝わっていく。この相関が結果として、表面を引っ張り上げ、最終的には飽和状態に到達させる。飽和状態は相関長が系の大きさと等しくなった状態を表していて、粒子の持つ情報が系全体に行き届いた結果である。

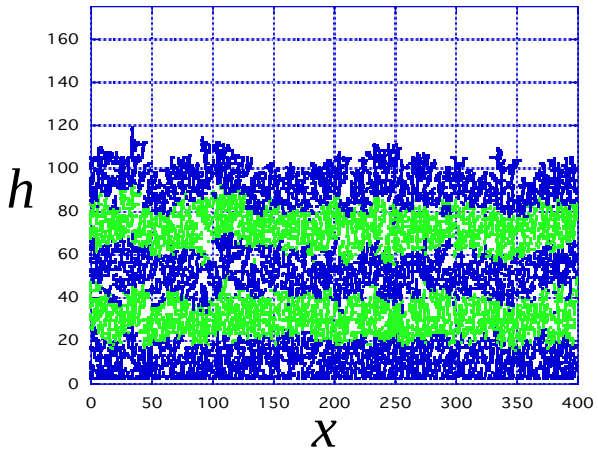


図 7: Ballistic deposition の表面。システムサイズ L=400 で、落とした粒子数が 20,000 個のとき。

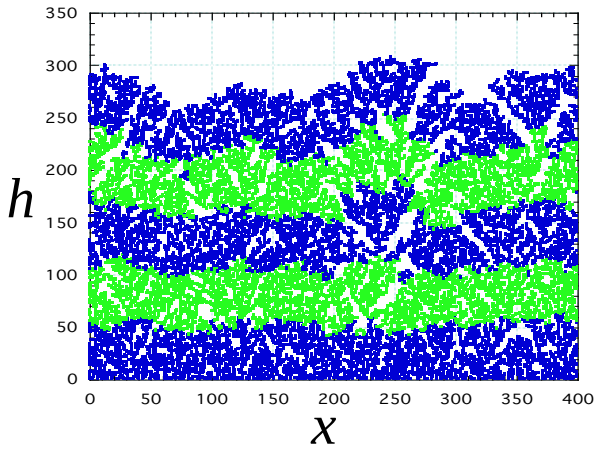


図 8: Domino model の表面。システムサイズ L=400 で、落とした粒子数が 20,000 個のとき。

4.2 スケーリング指数

まず最小二乗法を用いて指数 α を評価した（図 9）。式 (4) より、

$$\log w_{\mathrm{sat}} \approx \alpha \log L + \mathrm{const} \tag{14}$$

となる。この式でデータをフィットし、傾き α を求めた。尚、指数 α を求めるために用いたシステムサイズ L の範囲は Ballistic deposition、Domino model ともに $L = 300$〜1000 である。それぞれ間隔が 100 の計 8 個から各データ点の誤差を評価した。

次に最小二乗法を用いて指数 β を評価した（図 10）。式 (2) より

$$\log w(L,t) \approx \beta \log t + \mathrm{const} \tag{15}$$

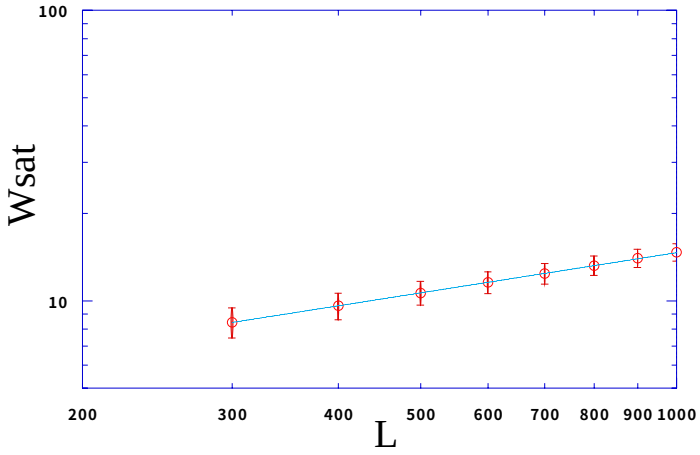
となる。この式でデータをフィットし、傾き β を求めた。尚、指数 β を求めるために用いた時刻 t の範囲は Ballistic deposition が $t = 90$〜180、Domino model が $t = 40$〜130 でそれぞれ間隔が 10 の計 10 個から各データ点の誤差を評価した。

同様にして最小二乗法を用いて指数 z を評価した（図 11）。式 (5) より、

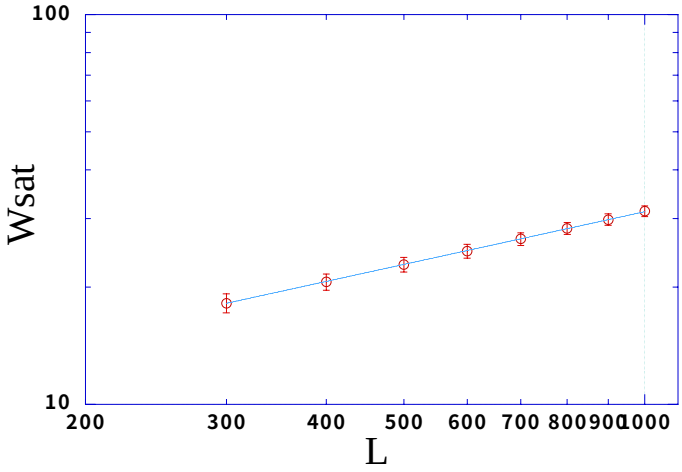
$$\log t_{\times} \approx z \log L + \mathrm{const} \tag{16}$$

となる。この式でデータをフィットし、傾き z を求めた。尚、指数 z を求めるために用いたシステムサイズ L の範囲は Ballistic deposition が $L = 100$〜1000 の計 10 個、Domino model が $L = 300$〜1000 の計 8 個から各データの誤差を評価した。データの間隔はともに 100 である。

以上の評価から表 1 の値を得た。表 1 から、今回のシミュレーションで得られた Ballistic deposition のスケーリング指数と Ballistic deposition の理論値 [1] はほぼ同じ値ととっていることがわかる。更に、誤差の範囲で Ballistic deposition と Domino model は一致している。つまり 2 つのモデルは同じユニヴァーサリティクラスに属していることがわかった。

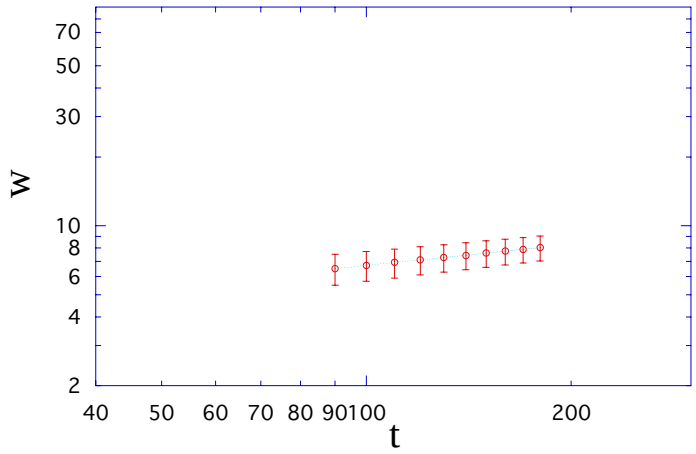


(a)

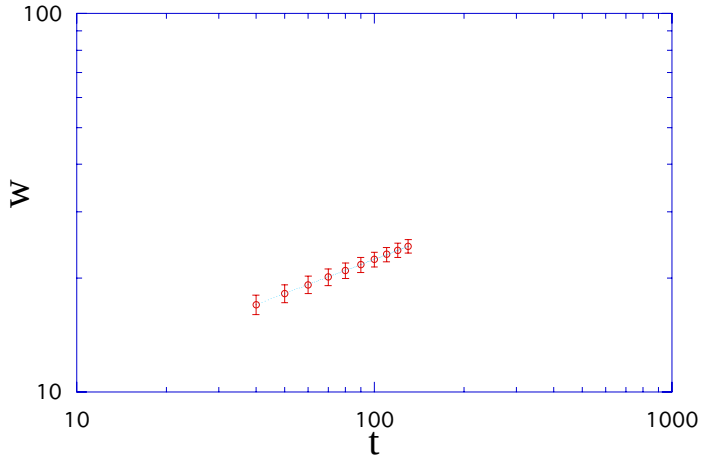


(b)

図 9: 指数 α の評価 : (a) Ballistic deposition ; (b) Domino model。

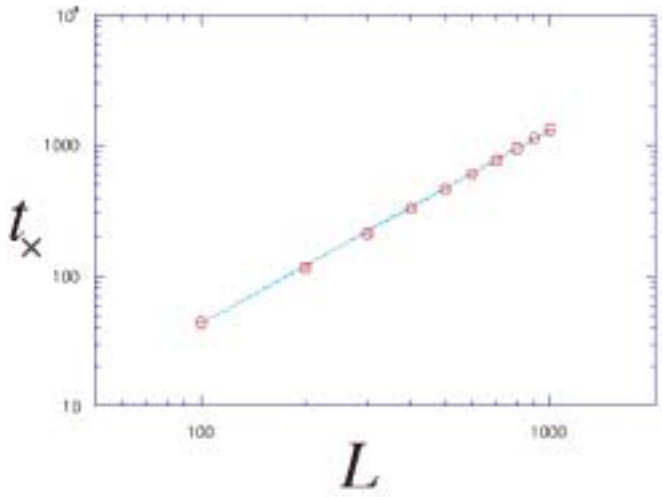


(a)

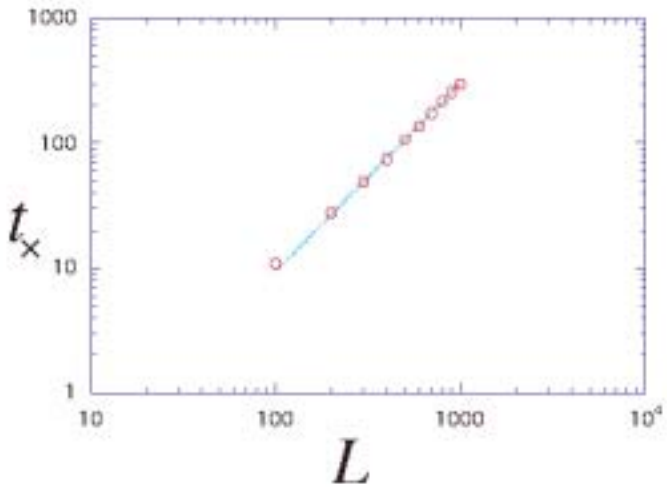


(b)

図 10: 指数 β の評価 : (a) Ballistic deposition ; (b) Domino model。



(a)



(b)

図 11: 指数 z の評価 : (a) Ballistic deposition ; (b) Domino model。

	α	β	z
Ballistic deposition	0.46±0.04	0.30±0.03	1.49±0.01
Ballistic deposition 理論値 [1]	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{3}{2}$
Domino model	0.45±0.04	0.30±0.05	1.51±0.01

表 1: スケーリング指数の比較。

5　まとめと考察

本研究では表面成長のモデルを定性的と定量的の両面から解析した。Ballistic deposition と Domino model の表面を見比べると、それぞれの時刻における表面の粗さや隙間の面積に違いが見られた。従って、これら２つのモデルは定性的に異なることがわかる。それにもかかわらず、スケーリング指数を求めて定量的にモデルを解析した結果、２つのモデルは同じ値を示した。つまりこれら２つのモデルは同じユニヴァーサリティクラスであるといえる。Ballistic deposition と Domino model の違う点は落とす粒子の形であり、Ballistic deposition の粒子に異方性を加えたのが Domino model である。２次元イジングモデルにおいて、正方格子モデルとそれに異方的相互作用を加えたモデルが同じユニヴァーサリティクラスであるように、表面成長モデルにおいてもやはり同じ事が言えるのではないかと考えられる。

本研究では両隣りにあるブロックとの相互作用しか考えていなかったが、更にその隣りにあるブロックとの２次的な相互作用までのルールを取り入れたモデルを考えて、スケーリング指数を調べるのもまた興味深い研究だろう。スケーリング指数のメリットはそこにあって、異方性や次近接以降の相互作用を加えたモデルでも値は変化しない。逆に、単純に見えるモデルでもスケーリング指数を調べた結果、明らかに指数の値が他のものと違うのであれば、それは特殊なモデルと言え、そこには何か複雑なメカニズムがあるかもしれない。

Ballistic deposition の理論値との比較では、誤差の範囲内ではあるが、結果はほぼ一致した。図 9〜11 において、誤差棒が短いのはそれぞれのシステムサイズについて 1000 回〜2000 回のサンプルをとったためである。α と β が理論値から多少ずれた原因として、扱った系のサイズや最小二乗法でフィットしたときのデータ数など考えられるが、傾き β を求める際の時間 t の選び方にも問題があったかもしれない。本研究で行った２つのモデルは共に、粒子が堆積し始めてからわずかな時間は式 (2) に従わないようなふるまいが見られた。一番最初に指数の計算をした時は、条件 $t \ll t_\times$ に注意するあまりこの時間を含めて算出してしまったためによい値は得られなかった。理論的にはクロスオーバータイム $t_\times$ に到達するまでは w は t のベキ乗で増加するので、それほど条件 $t \ll t_\times$ に気を配る必要はない。ただやはり時間が十分経つにつれて β より傾きは小さくなっていくので、システムサイズ L を大きくするなどの工夫が必要となる。そうすることで有限サイズ効果の緩和につながり、飽和するまでに多くの

時間を費やすことができる。

参考文献

[1] M. Kardar, G. Parisi and Y.-C. Zhang, 'Dynamic scaling of growing interfaces,'Phys. Rev. Lett. **56**, 889-892 (1986)

[2] A.-L. Barabási & H.E. Stanley, 'Fractal Concepts in Surface Growth,'(Cambridge University Press,1995)

[3] F. Family and T. Vicsek, 'Scaling of the active zone in the Eden process on percolation networks and the ballistic deposition model,'J. Phys. A **18**, L75-L81 (1985)

[4] J.M. Burgers, 'The Nonlinear Diffusion Equation,'(Riedel, Boston, 1974)

[5] N.G. van Kampen, 'Stochastic Processes in Physics and Chemistry,'(North-Holland, Amsterdasm, 1981)

[6] C.W. Gardiner, 'Handbook of Stochastic Methods,'(Springer-Verlag, Berlin, 1985)

[7] T. Halpin-Healy and Y.-C. Zhang, 'Surface growth ,directed polymers and all that,'Phys. Rep. **254**, 215-362 (1995)

A KPZ方程式から導くBallistic deposition理論値

ここではKPZ方程式をスケーリングすることによってスケーリング関係式を導き出し、そこからBallistic depositionの理論値 [1] を導出する。KPZ方程式 [1] とはBallistic depostionを理論的に再現した微分方程式で、

$$\frac{\partial h}{\partial t} = \nu \nabla^2 h + \frac{\lambda}{2}(\nabla h)^2 + \eta(x,t) \tag{17}$$

で与えられる。右辺の第1項は表面の拡散を表す線形項で、ν は表面張力を表している。また非線形項である第2項は、Ballistic depositionの堆積ルールに含まれる横への成長を再現した項である。つまり表面がある力 F によって引っ張られていると考えている。λ はその引っ張られる速度 v に関する量である。第3項は位置 x の時刻 t におけるランダムノイズで、これは表面成長モデルの堆積過程におけるランダムな変動を反映している。

このBallistic depositionを再現したKPZ方程式をそのままスケール変換したのではスケーリング指数は求められない。なぜならKPZ方程式をスケール変換するとき、それぞれ異なった項の係数（ν, λ, D）が独立に繰り込まれないからである [2]。D はノイズに含まれる係数で、ノイズの相関を考慮する際に出てくる次元を表した量である。

スケーリング指数を求めるために、まずガリレイ変換に対して不変であるBurger's equationからスケーリング関係式を導出し、更にブラウン運動している表面を考えることにより、スケーリング指数を求める。

Burger's equation [4] とはランダムにかきまぜられた過なしの流体を記述した方程式で、

$$\frac{\partial \mathbf{v}}{\partial t} + \lambda(\mathbf{v}\cdot\nabla)\mathbf{v} = \nu\nabla^2\mathbf{v} - \nabla\eta(x,t) \tag{18}$$

で与えられる。$\mathbf{v}(x,t)$ は流体の速度を表すベクトルである。右辺の ν は粘性率を表し、$\nabla\eta(x,t)$ は外力などのランダムな力を表している。この式の $\mathbf{v}$ を

$$\mathbf{v} = -\nabla h \tag{19}$$

と変数変換することでKPZ方程式へ写像できる。

ここで速度 $\mathbf{v}$ を全微分すると、

$$\frac{D\mathbf{v}}{Dt} = \frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v}\cdot\nabla)\mathbf{v} \tag{20}$$

が得られる。これは式 (18) の左辺と同じ形をしている。全微分することで表面の高さと横への成長部分を分けて考えることができる。但し、$\lambda = 1$ とおいた。なぜなら、式 (18) をスケール変換したとき、係数 λ と ν が同時に繰り込まれるとスケール不変性は保たれず、そのためにスケーリング指数が求まらないからである。よって全微分はスケール変換の前後で不変であるとして、$\lambda = 1$ とした。

ここで式 (20) に対して、

$$x \longrightarrow x' \equiv bx \tag{21}$$

$$h \longrightarrow h' \equiv b^{\alpha} h \tag{22}$$

$$t \longrightarrow t' \equiv b^{z} t \tag{23}$$

のようにスケール変換する。但し、表面は自己アフィンであるものと仮定して、上のような異方的なスケール変換を行なった。また表面は時間に関しても変化するので、時刻 t についてもスケール変換しなければならない。

式 (19) と (21)〜(23) を用いて、式 (20) をスケール変換すると、

$$\frac{D\mathbf{v}}{Dt} = b^{\alpha - z - 1} \frac{\partial \mathbf{v}}{\partial t} + b^{2\alpha - 3} (\mathbf{v} \cdot \nabla)\mathbf{v} \tag{24}$$

となる。よって、

$$\alpha + z = 2 \tag{25}$$

が得られる。これはスケーリング指数 α と z を関係づけた式で、あらゆる次元で成り立つことがわかっている [2]。上の変換を用いれば、係数 ν と λ が独立に繰り込まれ、スケール不変性を保ったまま KPZ 方程式をスケール変換することができる。

次に、今求まったスケーリング関係式 (25) からスケーリング指数を求める。ここで、一般的な Langevin 方程式

$$\frac{\partial h}{\partial t} = G(h) + \eta(t) \tag{26}$$

を考える。Langevin 方程式とはブラウン運動を確率的に記述した運動方程式で、ノイズ η の相関と拡散係数 D には

$$\langle \eta(t)\eta(t') \rangle = 2D\delta(t - t') \tag{27}$$

の関係がある。表面成長モデルを考える上では、式 (26) の $G(h)$ は時刻 t の位置 x における平均的な粒子の高さであると解釈してもよいだろう。この Langevin 方程式 (26) と Fokker-Planck 方程式 [5]、

$$\frac{\partial \Pi}{\partial t} = -\frac{\partial}{\partial h}[G(h)\Pi] + D\frac{\partial^2 \Pi}{\partial h^2} \tag{28}$$

を結びつけて考えると、指数 α が求められる。ここで $\Pi(h,t)$ は、時刻 t において粒子の高さが h であるという確率を表したものである。しかしこの h は KPZ 方程式とは違って、位置 x に依存しない量である。よってこの式を連続変数 x で積分すると [6, 7]、

$$\frac{\partial \Pi}{\partial t} = -\int d^d x \frac{\delta}{\delta h}\left[\left(\nu\nabla^2 h + \frac{\lambda}{2}(\nabla h)^2\right)\Pi\right] + D\int d^d x \frac{\delta^2 \Pi}{\delta h^2} \tag{29}$$

が得られる。但し、$G(h) \equiv \nu\nabla^2 h + \frac{\lambda}{2}(\nabla h)^2$ とおいた。この式から $d = 1$ のときの確率分布 Π を求めると [2]、

$$\Pi = \exp\left\{-\int dx \left[\frac{\nu}{2D}(\partial_x h)^2\right]\right\} \tag{30}$$

が求められる。これは $d = 1$ の時の式 (29) の解だが、$\lambda = 0$ の時のみ、次元によらず式 (29) の解となる。

式 (30) からわかることは、局所的な傾き ∇h がガウス分布をしていることであり、無相関ということである。局所的な傾き ∇h の集まりを全体的に見ると、表面はブラウン運動をしているわけである。

ここでブラウン運動することで知られる 1 次元ランダムウォークについて考える。もちろん 1 次元ランダムウォークの確率分布もガウス分布に従っている。その確率分布から標準偏差を考えると、

$$\sigma \sim |t_2 - t_1|^{1/2} \tag{31}$$

となり、時刻 t の $1/2$ 乗に比例する。ここで表面が自己アフィンであると仮定すると、ある 2 点間の距離とその高さとの関係は、

$$\Delta \sim l^{\alpha} \tag{32}$$

で与えられる [2]。但し、$\Delta(l) \equiv |h(x_1) - h(x_2)|$、$l \equiv |x_1 - x_2|$ とする。式 (31) と式 (32) から、roughness exponent α は

$$\alpha = \frac{1}{2} \tag{33}$$

であることがわかる。これが KPZ 方程式から導かれるスケーリング指数 α である。更に式 (13) と式 (25) より、

$$z = \frac{3}{2} \quad , \quad \beta = \frac{1}{3} \tag{34}$$

が求められる。これが $d = 1$ のときの KPZ 方程式から導かれる理論値でこれは Ballistic deposition のスケーリング指数と同じユニバーサリティクラスに属している [2]。

B Ballistic depositionの表面のプログラム

```
#include <stdio.h>
#define SIMPLE_SPRNG
#include "sprng.h"
#define SEED 985456376
#define  _L  400
#define  _HEIGHT  300
#define  _PARTICLES  20000

main(){
  FILE  *datafile;
  int  x,y,t,h1,h2;
  int  h[_L],particle[_L][_HEIGHT];

  datafile=fopen("BDMODEL.dat","w");
  init_sprng(DEFAULT_RNG_TYPE,SEED,SPRNG_DEFAULT);

  h1=0;
  h2=0;
  for(x=0;x<_L;x++){
      for(y=0;y<_HEIGHT;y++){
          particle[x][y]=0;
      }
  }
  for(t=0;t<=35000;t++){
      h[x]=0;
  }

  for(t=0;t<_PARTICLES;t++){
   /*    まず粒子の落とす位置をランダムに決める。    */
      x=(int)(sprng()*_L);

   /*      系の両端に境界条件を与える。    */
      if(x==0){
           h1=h[_L-1];
```

```
        }
        else{
             h1=h[x-1];
        }
        if(x==_L-1){
             h2=h[0];
        }
        else{
             h2=h[x+1];
        }

    /*   両隣りの粒子の高さを判別する。   */
        if(h1>h[x]  &&  h1>h2){
              h[x]=h1;
              particle[x][h1]=1;
        }
        else  if(h[x]>=h1  &&  h[x]>=h2){
              h[x]++;
              particle[x][h[x]]=1;
         }
        else  if(h2>=h1  &&  h2>h[x]){
              h[x]=h2;
              particle[x][h2]=1;
         }
    }

    for(x=0;x<_L;x++){
        for(y=0;y<_HEIGHT;y++){
             if(particle[x][y]==1){
                   fprintf(datafile,"%d  %d  %d\n",x,y,particle[x][y]);
             }
        }
    }
    fclose(datafile);
}
```

C　Domino modelの表面のプログラム

```
#include <stdio.h>
#define SIMPLE_SPRNG
#include "sprng.h"
#define SEED 985456376
#define  _L  400
#define  _HEIGHT  400
#define  _PARTICLES  20000

main(){
  FILE  *datafile;
  int  x,y,t,i,p;
  int  x1,x2,x4;
  static  int  h[_L],particle[_L][_HEIGHT];

  datafile=fopen("DOMINO.dat","w");
  init_sprng(DEFAULT_RNG_TYPE,SEED,SPRNG_DEFAULT);

  for(x=0;x<_L;x++){
      for(y=0;y<_HEIGHT;y++){
          particle[x][y]=0;
      }
  }
  for(x=0;x<_L;x++){
      h[x]=0;
  }

  for(t=0;t<_PARTICLES;t++){
  /*   粒子の落とす位置をランダムに決める    */
      x=(int)(sprng()*_L);
  /*   落とす粒子の形をランダムに決める    */
      p=(int)(sprng()*2);

  /*   境界条件を与える    */
```

```
    x1=((x-1)+_L)%_L;
    x2=((x+1)%_L);
    x4=((x+2)%_L);

/*   両隣りの粒子の高さを判別する    */
    switch(p){
        case  0:

        if(h[x1]>h[x]  &&  h[x1]>h[x2]){
             h[x]=h[x1];
             particle[x][h[x1]]=1;
             h[x]++;
             particle[x][h[x]]=1;
        }
        else if(h[x]>=h[x1]  &&  h[x]>=h[x2]){
             for(i=1;i<=2;i++){
                  h[x]++;
                  particle[x][h[x]]=1;
             }
        }
        else if(h[x2]>=h[x1]  &&  h[x2]>h[x]){
             h[x]=h[x2];
             particle[x][h[x2]]=1;
             h[x]++;
             particle[x][h[x]]=1;
        }
        else{
             printf("ERROR1\n");
        }
        break;

        case 1:

        if(h[x4]<=h[x1]  ||  h[x4]<=h[x]  ||  h[x4]<=h[x2]){
              if(h[x1]>h[x]  &&  h[x1]>h[x2]){
```

```
                h[x]=h[x1];
                particle[x][h[x1]]=1;
                h[x2]=h[x];
                particle[x2][h[x]]=1;
            }
            else if(h[x]>=h[x1]  &&  h[x]>=h[x2]){
                h[x]++;
                particle[x][h[x]]=1;
                h[x2]=h[x];
                particle[x2][h[x]]=1;
            }
            else if(h[x2]>=h[x1]  &&  h[x2]>h[x]){
                h[x2]++;
                particle[x2][h[x2]]=1;
                h[x]=h[x2];
                particle[x][h[x2]]=1;
            }
            else{
                printf("ERROR2\n");
            }
        }
        else if(h[x4]>h[x1]  &&  h[x4]>h[x]  &&  h[x4]>h[x2]){
            h[x2]=h[x4];
            particle[x2][h[x4]]=1;
            h[x]=h[x2];
            particle[x][h[x2]]=1;
        }
        else{
            printf("ERROR3\n");
        }
        break;

    default:
         printf("ERROR4\n");
         }
```

```
        }

  for(x=0;x<_L;x++){
      for(y=0;y<_HEIGHT;y++){
          if(particle[x][y]==1){
               fprintf(datafile,"%d  %d  %d\n",x,y,particle[x][y]) ;
          }
      }
  }
  fclose(datafile);
}
```

D Ballistic depositionのw_{sat}を求めるプログラム（L=400のとき）

```
#include <stdio.h>
#include <math.h>
#define SIMPLE_SPRNG
#include "sprng.h"
#define SEED 985456376
#define  _L  400
#define  _MAXT  20000
#define  _SAMPLE  1000
#define  _A  10000
#define  _INTERVAL  1000
#define  _N  10

main(){
  FILE  *datafile;
  int  x,y,t,i,j,a;
  double  h1,h2,sum1,sum2,h1_av,h2_av,bunsan,delta;
  double  w_bar,w1,w2,w3,w4,w5,w6,w7,w8,w9,w10,w11,w12,w13;
  static  double  h[_L],width[_MAXT],w_av[_N]

  datafile=fopen("width.dat","w");
  init_sprng(DEFAULT_RNG_TYPE,SEED,SPRNG_DEFAULT);
  for(j=0;j<_MAXT;j++){
     width[j]=0.0;
  }
  for(a=0;a<_N;a++){
     w_av[a]=0.0;
  }

  for(i=0;i<_SAMPLE;i++){
     j=0;
     h1=0;
     h2=0;
```

```
for(x=0;x<_L;x++){
   h[x]=0.0;
}

for(t=0;t<=_MAXT*_L;t++){
   x=(int)(sprng()*_L);

   if(x==0){
        h1=h[_L-1];
   }
   else{
        h1=h[x-1];
   }
   if(x==_L-1){
        h2=h[0];
   }
   else{
        h2=h[x+1];
   }
   if(h1>h[x]  &&  h1>h2){
        h[x]=h1;
   }
   else  if(h[x]>=h1  &&  h[x]>=h2){
        h[x]+=1.0;
   }
   else  if(h2>=h1  &&  h2>h[x]){
        h[x]=h2;
   }
   else{
        printf("ERROR\n");
   }
   w1=0,0;
   h1_av=0.0;
   h2_av=0.0;
   if(t!=0 && t%(_L)==0){
```

```
            sum1=0.0;
            sum2=0.0;
            for(x=0;x<_L;x++){
               sum1+=h[x];
               sum2+=h[x]*h[x];
            }
            h1_av=sum1/_L;
            h2_av=sum2/_L;
            w1=h2_av-h1_av*h1_av;
            width[j]+=sqrt(w1);
            j++;
      }
   }
}

/*    飽和した状態の w のデータを 10 個選ぶ    */
w2=w3=w4=w5=w6=w7=w8=w9=w10=w11=0.0;
for(j=_A;j<_A+_INTERVAL;j++){
   w2+=width[j];
}
w_av[0]=(double)(w2/_INTERVAL);
for(j=_A+_INTERVAL;j<_A+_INTERVAL*2;j++){
   w3+=width[j];
}
w_av[1]=(double)(w3/_INTERVAL);
for(j=_A+_INTERVAL*2;j<_A+_INTERVAL*3;j++){
   w4+=width[j];
}
w_av[2]=(double)(w4/_INTERVAL);
for(j=_A+_INTERVAL*3;j<_A+_INTERVAL*4;j++){
   w5+=width[j];
}
w_av[3]=(double)(w5/_INTERVAL);
for(j=_A+_INTERVAL*4;j<_A+_INTERVAL*5;j++){
   w6+=width[j];
```

```
}
w_av[4]=(double)(w6/_INTERVAL);
for(j=_A+_INTERVAL*5;j<_A+_INTERVAL*6;j++){
   w7+=width[j];
}
w_av[5]=(double)(w7/_INTERVAL);
for(j=_A+_INTERVAL*6;j<_A+_INTERVAL*7;j++){
   w8+=width[j];
}
w_av[6]=(double)(w8/_INTERVAL);
for(j=_A+_INTERVAL*7;j<_A+_INTERVAL*8;j++){
   w9+=width[j];
}
w_av[7]=(double)(w9/_INTERVAL);
for(j=_A+_INTERVAL*8;j<_A+_INTERVAL*9;j++){
   w10+=width[j];
}
w_av[8]=(double)(w10/_INTERVAL);
for(j=_A+_INTERVAL*9;j<_A+_INTERVAL*10;j++){
   w11+=width[j];
}
w_av[9]=(double)(w11/_INTERVAL);

/*    飽和した状態の w の平均、分散とその誤差を求める    */
w12=w13=0.0;
bunsan=0.0;
delta=0.0;
for(a=0;a<_N;a++){
   w12+=w_av[a];
}
w_bar=0.0;
w_bar=(double)(w12/_N);                    /*    平均    */
for(a=0;a<_N;a++){
   w13+=(w_av[a]-w_bar)*(w_av[a]-w_bar);
}
```

```
  bunsan=(double)(w13/(_N-1));          /*   分散   */
  delta=(double)(sqrt(bunsan/_N));      /*   誤差   */

  printf("w_av=%lf\n",w_bar);
  printf("bunsan=%lf\n",bunsan);
  printf("delta=%lf\n",delta);

  fclose(datafile);
}
```

E Ballistic depositionの指数βを求めるプログラムーその１－

```
/* Ballistic deposition */
#include <stdio.h>
#include <math.h>

#define SIMPLE_SPRNG            /* simple interface */
#include "sprng.h"              /* SPRNG header file */
#define SEED 985456376

#define _L 10000
#define _MAXT 181
#define _SAMPLE 1000
#define _N 11

main(){
  FILE *fout1,*fout2,*fout3,*fout4,*fout5,*fout6,*fout7,*fout8,*fout9,*fout10;
  int x,y,t,i,j,a;
  double h1,h2,sum1,sum2,h1_av,h2_av,w0;
  double w1,w2,w3,w4,w5,w6,w7,w8,w9,w10;
  static double h[_L],width[_SAMPLE][_MAXT],w_av[_N];

  fout1=fopen("exponent_beta11.dat","w");
  fout2=fopen("exponent_beta12.dat","w");
  fout3=fopen("exponent_beta13.dat","w");
  fout4=fopen("exponent_beta14.dat","w");
  fout5=fopen("exponent_beta15.dat","w");
  fout6=fopen("exponent_beta16.dat","w");
  fout7=fopen("exponent_beta17.dat","w");
  fout8=fopen("exponent_beta18.dat","w");
  fout9=fopen("exponent_beta19.dat","w");
  fout10=fopen("exponent_beta110.dat","w");

  init_sprng(DEFAULT_RNG_TYPE,SEED,SPRNG_DEFAULT);     /* Initialization */
```

```
  for(i=0;i<_SAMPLE;i++){
    for(j=0;j<_MAXT;j++){
      width[i][j]=0.0;
    }
  }
  for(a=0;a<_N;l++){
    w_av[a]=0.0;
  }

  for(i=0;i<_SAMPLE;i++){
    j=0;
    h1=0.0;
    h2=0.0;
    for(x=0;x<_L;x++){
      h[x]=0.0;
    }
    for(t=0;t<=_MAXT*_L;t++){
      x=(int)(sprng()*_L);           /* system size */

      if(x==0){
h1=h[_L-1];
      }
      else{
h1=h[x-1];
      }
      if(x==_L-1){
h2=h[0];
      }
      else{
h2=h[x+1];
      }
      if(h1>h[x] && h1>h2){
h[x]=h1;
      }
```

```
      else if(h[x]>=h1 && h[x]>=h2){
h[x]+=1.0;
      }
      else if(h2>=h1 && h2>h[x]){
h[x]=h2;
      }
      else {
printf("ERROR\n");
      }
      w0=0.0;
      h1_av=0.0;
      h2_av=0.0;
      if(t!=0 && t%(_L)==0){
sum1=0.0;
sum2=0.0;
for(x=0;x<_L;x++){
  sum1+=h[x];
  sum2+=h[x]*h[x];
}
h1_av=sum1/_L;
h2_av=sum2/_L;
w0=h2_av-h1_av*h1_av;
width[i][j]=sqrt(w0);
j++;
      }
    }
  }
  w1=w2=w3=w4=w5=w6=w7=w8=w9=w10=0.0;
  for(i=0;i<_SAMPLE;i++){
    fprintf(fout1,"%lf\n",width[i][90]);
    w1+=width[i][90];
    fprintf(fout2,"%lf\n",width[i][100]);
    w2+=width[i][100];
    fprintf(fout3,"%lf\n",width[i][110]);
    w3+=width[i][110];
```

```
  fprintf(fout4,"%lf\n",width[i][120]);
  w4+=width[i][120];
  fprintf(fout5,"%lf\n",width[i][130]);
  w5+=width[i][130];
  fprintf(fout6,"%lf\n",width[i][140]);
  w6+=width[i][140];
  fprintf(fout7,"%lf\n",width[i][150]);
  w7+=width[i][150];
  fprintf(fout8,"%lf\n",width[i][160]);
  w8+=width[i][160];
  fprintf(fout9,"%lf\n",width[i][170]);
  w9+=width[i][170];
  fprintf(fout10,"%lf\n",width[i][180]);
  w10+=width[i][180];
}

w_av[1]=(double)(w1/_SAMPLE);
printf("w_av[1]=%lf\n",w_av[1]);
w_av[2]=(double)(w2/_SAMPLE);
printf("w_av[2]=%lf\n",w_av[2]);
w_av[3]=(double)(w3/_SAMPLE);
printf("w_av[3]=%lf\n",w_av[3]);
w_av[4]=(double)(w4/_SAMPLE);
printf("w_av[4]=%lf\n",w_av[4]);
w_av[5]=(double)(w5/_SAMPLE);
printf("w_av[5]=%lf\n",w_av[5]);
w_av[6]=(double)(w6/_SAMPLE);
printf("w_av[6]=%lf\n",w_av[6]);
w_av[7]=(double)(w7/_SAMPLE);
printf("w_av[7]=%lf\n",w_av[7]);
w_av[8]=(double)(w8/_SAMPLE);
printf("w_av[8]=%lf\n",w_av[8]);
w_av[9]=(double)(w9/_SAMPLE);
printf("w_av[9]=%lf\n",w_av[9]);
w_av[10]=(double)(w10/_SAMPLE);
```

```
  printf("w_av[10]=%lf\n",w_av[10]);

  fclose(fout1);
  fclose(fout2);
  fclose(fout3);
  fclose(fout4);
  fclose(fout5);
  fclose(fout6);
  fclose(fout7);
  fclose(fout8);
  fclose(fout9);
  fclose(fout10);

}
```

F Ballistic depositionの指数βを求めるプログラムーその２ー

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define _SAMPLE 1000
#define _MAXT 181
#define _N 11

main(int argc, char *argv[])
{
  FILE *fin1,*fin2,*fin3,*fin4,*fin5,*fin6,*fin7,*fin8,*fin9,*fin10,*fout;
  int i,l;
  double w1,w2,w3,w4,w5,w6,w7,w8,w9,w10;
  double b1,b2,b3,b4,b5,b6,b7,b8,b9,b10;
  static double width[_SAMPLE][_MAXT],w_av[_N];

  if(argc!=12){
    printf("hikisu error\n");
    exit(1);
  }
  if((fin1=fopen(argv[1],"r"))==NULL){
    printf("readfile error\n");
    exit(1);
  }
  if((fin2=fopen(argv[2],"r"))==NULL){
    printf("readfile error\n");
    exit(1);
  }
  if((fin3=fopen(argv[3],"r"))==NULL){
    printf("readfile error\n");
    exit(1);
  }
```

```
if((fin4=fopen(argv[4],"r"))==NULL){
  printf("readfile error\n");
  exit(1);
}
if((fin5=fopen(argv[5],"r"))==NULL){
  printf("readfile error\n");
  exit(1);
}
if((fin6=fopen(argv[6],"r"))==NULL){
  printf("readfile error\n");
  exit(1);
}
if((fin7=fopen(argv[7],"r"))==NULL){
  printf("readfile error\n");
  exit(1);
}
if((fin8=fopen(argv[8],"r"))==NULL){
  printf("readfile error\n");
  exit(1);
}
if((fin9=fopen(argv[9],"r"))==NULL){
  printf("readfile error\n");
  exit(1);
}
if((fin10=fopen(argv[10],"r"))==NULL){
  printf("readfile error\n");
  exit(1);
}
if((fout=fopen(argv[11],"w"))==NULL){
  printf("writefile error\n");
  exit(1);
}
for(l=1;l<_N;l++){
  w_av[l]=0.0;
}
```

```
for(i=0;i<_SAMPLE;i++){
  fscanf(fin1,"%lf",&width[i][90]);
  fscanf(fin2,"%lf",&width[i][100]);
  fscanf(fin3,"%lf",&width[i][110]);
  fscanf(fin4,"%lf",&width[i][120]);
  fscanf(fin5,"%lf",&width[i][130]);
  fscanf(fin6,"%lf",&width[i][140]);
  fscanf(fin7,"%lf",&width[i][150]);
  fscanf(fin8,"%lf",&width[i][160]);
  fscanf(fin9,"%lf",&width[i][170]);
  fscanf(fin10,"%lf",&width[i][180]);
}
for(l=1;l<_N;l++){
  printf("w_av[%d]=",l,w_av[l]);
  scanf("%lf",&w_av[l]);
}
for(i=0;i<_SAMPLE;i++){
  w1+=(width[i][90]-w_av[1])*(width[i][90]-w_av[1]);
  w2+=(width[i][100]-w_av[2])*(width[i][100]-w_av[2]);
  w3+=(width[i][110]-w_av[3])*(width[i][110]-w_av[3]);
  w4+=(width[i][120]-w_av[4])*(width[i][120]-w_av[4]);
  w5+=(width[i][130]-w_av[5])*(width[i][130]-w_av[5]);
  w6+=(width[i][140]-w_av[6])*(width[i][140]-w_av[6]);
  w7+=(width[i][150]-w_av[7])*(width[i][150]-w_av[7]);
  w8+=(width[i][160]-w_av[8])*(width[i][160]-w_av[8]);
  w9+=(width[i][170]-w_av[9])*(width[i][170]-w_av[9]);
  w10+=(width[i][180]-w_av[10])*(width[i][180]-w_av[10]);
}

b1=(double)(w1/(_SAMPLE-1));
b2=(double)(w2/(_SAMPLE-1));
b3=(double)(w3/(_SAMPLE-1));
b4=(double)(w4/(_SAMPLE-1));
b5=(double)(w5/(_SAMPLE-1));
```

```
  b6=(double)(w6/(_SAMPLE-1));
  b7=(double)(w7/(_SAMPLE-1));
  b8=(double)(w8/(_SAMPLE-1));
  b9=(double)(w9/(_SAMPLE-1));
  b10=(double)(w10/(_SAMPLE-1));

  printf("bunsan[1]=%lf\n",b1);
  printf("bunsan[2]=%lf\n",b2);
  printf("bunsan[3]=%lf\n",b3);
  printf("bunsan[4]=%lf\n",b4);
  printf("bunsan[5]=%lf\n",b5);
  printf("bunsan[6]=%lf\n",b6);
  printf("bunsan[7]=%lf\n",b7);
  printf("bunsan[8]=%lf\n",b8);
  printf("bunsan[9]=%lf\n",b9);
  printf("bunsan[10]=%lf\n",b10);

  fclose(fin1);
  fclose(fin2);
  fclose(fin3);
  fclose(fin4);
  fclose(fin5);
  fclose(fin6);
  fclose(fin7);
  fclose(fin8);
  fclose(fin9);
  fclose(fin10);
  fclose(fout);
}
```

このプログラムより求まった分散を用いて、最小二乗法で指数βを出す。

G　最小二乗法のプログラム

```
#include <stdio.h>
#include <math.h>
#define  _N  10

main(){
  int  i;
  double  delta,sum1,sum2,sum3,sum4,sum5,a1,a2,s1,s2;
  static  double x[_N], y[_N], w[_N];

  for(i=0;i<_N;i++){
      x[i]=0.0;
      y[i]=0.0;
      w[i]=0.0;                          /*   wは重みを表す   */
  }

  /*   データを入力する   */
  for(i=0;i<_N;i++){
      printf("i=",i);
      scanf("%d",&i);
      printf("x=",x[i]);
      scanf("%lf",&x[i]);
      printf("y=",y[i]);
      scanf("%lf",&y[i]);
      printf("w=",w[i]);
      scanf("%lf",&w[i]);
  }
  printf("\n");

  sum1=0.0;
  sum2=0.0;
  sum3=0.0;
  sum4=0.0;
  sum5=0.0;
  for(i=0;i<10;i++){
```

```
        sum1+=w[i];
        sum2+=x[i]*w[i];
        sum3+=x[i]*x[i]*w[i];
        sum4+=y[i]*w[i];
        sum5+=x[i]*y[i]*w[i];
    }
     delta=sum1*sum3-sum2*sum2;

    /*    直線の切片を求める    */
    a1=(sum4*sum3-sum2*sum5)/delta;
    printf("seppen=%lf\n",a1);

    /*    切片の誤差を求める    */
    s1=sqrt(sum3/delta);
    printf("seppen_gosa=%lf\n",s1);

    /*    直線の傾きを求める    */
    a2=(sum1*sum5-sum2*sum4)/delta;
    printf("katamuki=%lf\n",a2);

    /*    傾きの誤差を求める    */
    s2=sqrt(sum1/delta);
    printf("katamuki_gosa=%lf\n",s2);
}
```

謝辞

羽田野先生には本当にお世話になりました。本研究を始める前から、進路や研究テーマについて相談にのっていただいたり、研究で悩んでいる時に羽田野先生のご指導のおかげで乗り越えることができました。大変感謝しています。また研究室の先輩方の貴重なアドバイスや同級生の人たちと一緒に考える機会があったからこそ、研究もここまで進むことができました。ありがとうございました。