カオスの構造の定量的解析

及川 晃平
羽田野研究室

平成 12 年 2 月 29 日

概 要

環境科学や気象学の分野において、より正確で長期間の予報を行う上で、自然現象のカオス的性質を取り入れた予報モデルが必要とされている。そこで本研究ではカオスの特徴を表す軌道拡大率という物理量を用いて、カオスの構造を定量的に解析した。カオスを示す 2 つのモデル (Lorenz Equations と パイコネ変換) をシミュレーションし、その振る舞いを調べた。その結果、理論とシミュレーションから得られた軌道拡大率の値が非常に良く一致した。

目 次

1　はじめに

我々の住んでいるこの自然界には、予測不可能で複雑な振舞いをするカオス的性質を持つ現象が数多く存在している。例を挙げると、非平衡状態での熱対流、その対流によって起っている様々な気象現象などである。そこで、カオスの特徴を表す物理量を定義して、カオスの構造を定量的に解析すれば、カオスをただ予測不可能として扱うのではなく、別の視点から捉えられる。この研究ではカオスを定量的に表す量として軌道拡大率を使う。まず、カオスであることが解っている系でこの軌道拡大率の振舞いを調べておき、次に、そうでない系に適用して定常状態とカオス状態の相転移を調べた。常にカオス状態にあるのモデルとしてパイこね変換を用い、定常状態とカオス状態の両方をとるモデルとしてローレンツ方程式を用いた。

2　カオスの特徴

この節ではカオスの特徴を列挙しておく。

1 初期値に対して鋭敏な依存性を持つ。

例えば、ある値x_0から時間発展方程式$x_{t+1} = f(x_t)$により次々にx_1,x_2,x_3...を求める場合を考える。x_0から微少距離だけ離れた点をx'_0から始まる、時間発展をx'_1,x'_2,x'_3...としたとき、x_nとx'_nは初めは同じような軌道を通るが時間が経つにつれて全く別の軌道を通るようになる。

2 方程式の時間発展が決定論の方程式で与えられるが、確率論的な取り扱いができる。

ある系を考えてその系が取り得る範囲を100個位の小区間に分けて、その各々にx_nが落る回数のヒストグラムを作る。ここでnを出来る限り長くとると、大抵x_0を何処にとってもいつも同じようなヒストグラムを得ることが出来る。

3 方程式の通る軌道が有界な範囲内で運動するが、軌道が交わることがない。

特に、本研究では2－1で説明した初期値に対する鋭敏な依存性に着目している。

3　カオスの例

この節ではカオス状態を示す２つの例を挙げる。最初のパイこね変換は常にカオス状態をとることがわかっている。次のローレンツ方程式は定常状態とカオス状態の両方をとる。

3.1　一般化されたパイこね変換

一般化されたパイこね変換は単位正方形の閉領域からそれ自身への二次元写像であり [1]

$$(x_{t+1}, y_{t+1}) = \begin{cases} (\xi_a x_t, y_t/a) & (0 \leq y_t \leq a) \\ (0.5 + \xi_b x_t, (y_t - a)/b) & (a < y_t \leq 1) \end{cases} \tag{1}$$

で定義される。ただし ξ_a, ξ_b, a, b は定数で、それぞれ $0 < \xi_a < 0.5$, $0 < \xi_b < 0.5$, $0 < a < 1$, $b = 1 - a$ を満たすとする。この変換はパラメーター ξ_a, ξ_b, a, b の値を変化させても常にカオスの性質を持つことがわかっている。この変換の様子を図に示すと図 1 のようになる。この変換によって図 1(a) の正方形は図 1(b) になり、もう一度変換すると図 1(c) となる。すなわち、領域 a の部分は x 方向の長さが ξ_a になり y 方向では正の方向に伸ばされ正方形の左側に詰められる。また、領域 b の部分は x 方向の長さが ξ_b になり y 方向では負の方向に伸ばされ正方形の右側によせられる。

ここで図 1(a) の正方形内の領域 a と領域 b の境界を狭んでいる２つの近接点に注目する。この点の領域 a のほうにある点を x_{a0}、領域 b のほうにある点を x_{b0} とすると図 1(a) から図 1(b) への変換により点 x_{a0} と点 x_{b0} は図 2 の点 x_{a1} と点 x_{b1} になる。つまり、少しでも初期値が違うと全く別のところに行き着いてしまう。これは、２節で説明したカオスの特徴を良く表している。

3.2　ローレンツ方程式 (Lorenz Equations)

ローレンツ方程式は [2]

$$\begin{cases} \frac{dx}{dt} = -\sigma(x - y) \\ \frac{dy}{dt} = -xy + rx - y \\ \frac{dz}{dt} = xy - bz \end{cases} \tag{2}$$

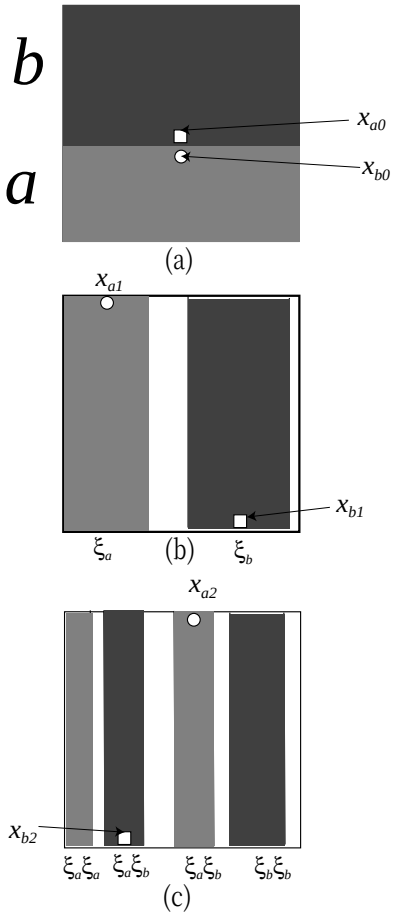

図 1: パイこね変換の様子。y 方向にはいつも伸び、x 方向にはいつも縮み、正方形からはみ出た部分は正方形内に戻し全体として面積はいつも縮小していくような変換である。

という３個の変数 x, y, z を含む非線形常微分方程式である。ただし、σ, r, b は定数とする。この方程式はアメリカの気象学者 Lorenz が大気の対流現象研究のために見出した対流乱流モデル（天気予報のモデル）である。このモデルは微分方程式の解がパラメーターのある値を境に対流（定常状態）から乱流（カオス状態）へと移り変わり、非常に複雑な挙動を示す。本研究では $\sigma = 10$,$b = 8/3$ に固定し、変化させるパラメーターは r ただ１つとする。r の値が $24.74 < r < 145$ のときにこの方程式の解はカオス状態になることが知られている。[2]

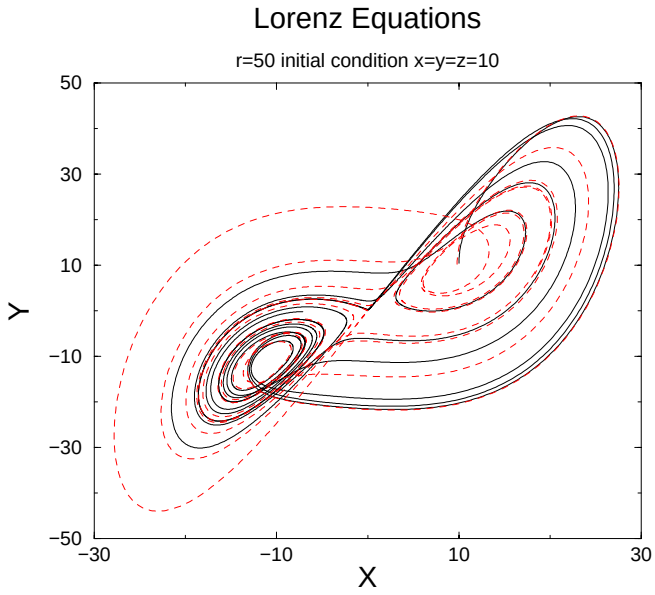

図 2: 初期値のわずかに違う軌道の振舞い。３次元空間内の軌道を X-Y 平面に写影してある。初期値は $x = y = z = 10$ と $x = y = z = 9.8$ でパラメーターは $r = 50$ である。

図 2 は初期値のわずかに違う２点のカオス状態にあるときのローレンツ方程式の解である。図を見ると判るとおり初期値が少しでも違っていると行き着く所は全く別の所である。また、軌道はある有限の範囲内を動き回っている。これは、２節で説明したカオスの特徴を良く表している。

4　軌道拡大率

軌道拡大率とは、今回の研究で一番の要となる物理量であり、定常状態とカオス状態とを定量的に区別することができる物理量である。ある系の任意の軌道の初期点 x_0 とその点から微少距離 δ だけ離れた点 $x_{0\delta}$ を考える。そしてその系を時間発展させる。n 単位時間経過した後に x_n と $x_{n\delta}$ との距離が $A\delta$ となっていたとする。すなわち、n 単位時間後に軌道の差が初期値の差の A 倍となっているときに、この A を

$$A = \exp(n\lambda) \tag{3}$$

で近似したときの λ が軌道拡大率という物理量である。式 (3) を変形して

$$\langle\lambda\rangle = \langle\frac{\ln A}{n}\rangle \tag{4}$$

とする。ここで $\langle\ldots\rangle$ は系全体の平均をとるという意味である。式 (4) を系の軌道拡大率として定義する。図 3 は軌道拡大率の概念図である。

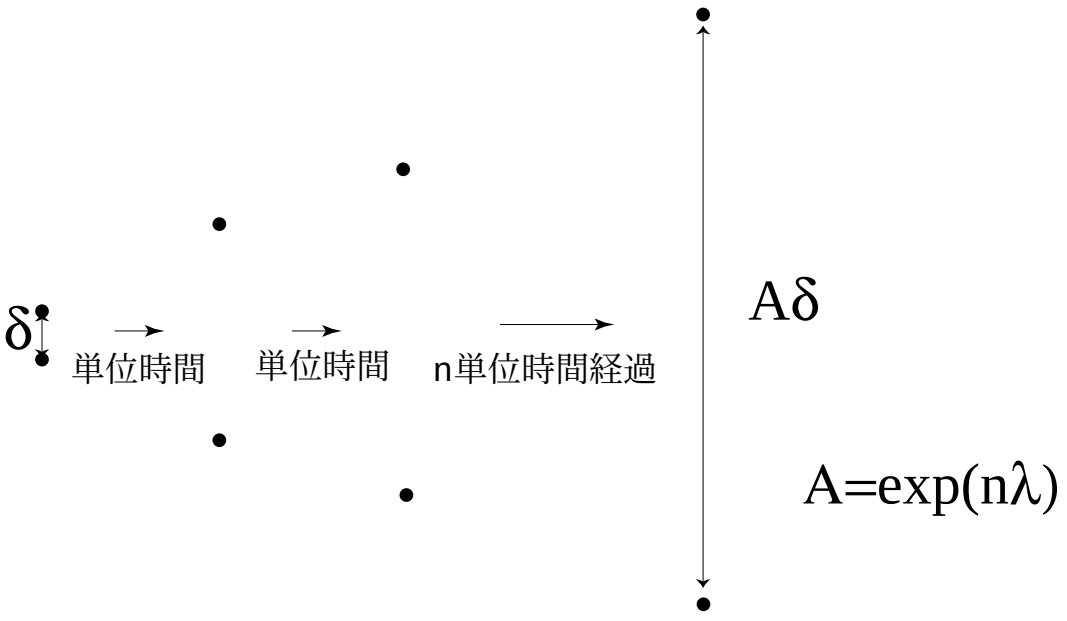

図 3: 軌道拡大率の概念図。n 単位時間経過後に 2 点間の距離が A 倍になっているとき。

次に、この軌道拡大率と定常状態からカオス状態への移り変わりがどのように関係しているかということを説明する。まず、$\lambda < 0$ のときには $A < 1$ となる。すなわち n 時間経過後の x_n と $x_{n\delta}$ との距離が δ より小さくなっているということである。$n \to \infty$ にしたときに $A \to 0$ となり δ がどんなに大きくても軌道が 1 つにまとまってしまう（ある一定の値に収束していく）。これは定常状態であるということが出来る。一方、$\lambda > 0$ のときには $A > 1$ となり n 時間経過後の x_n と $x_{n\delta}$ との距離が δ よりも大きくなっている。これは $n \to \infty$ にしたときに $A \to \infty$ となり、$\delta \ll 1$ でも最終的な軌道に大きな違いを生じる（初期値に対して鋭敏な依存性がある）という意味で、カオス状態を表しているということが出来る。

5 パイこね変換の軌道拡大率とシミュレーション（常にカオス状態をとることがわかっている系）

この節ではまずカオス状態であることが判っている系でこの軌道拡大率という物理量の特徴を調べる。それをもとに次の節で定常状態とカオス状態の両方をとる系についての軌道拡大率の変化を調べる。そこでまず常にカオスと判っているパイこね変換についての軌道拡大率を理論的に求めてみる。そして次にコンピュータでシミュレーションを行いその結果を理論値と比較する。

正方形に一様に点が分布しているとき、n 回パイこね変換すると 2^n 個の細片ができる。この細片のうちでその幅が $\xi_a^{n-r}\xi_b^r$ である細片内にある点を見い出す確率は $a^{n-r}b^n$ である。ただし、r は 0 から n までの整数である。ここで、二項定理

$$(x+y)^n = \sum_{r=0}^{n} {}_nC_r x^{n-r} y^r \tag{5}$$

より、幅が $\xi_a^{n-r}\xi_b^r$ である細片は ${}_nC_r$ 個ある。

$\vec{x}$ を初期点とする軌道を考える。近接軌道と元の軌道の y 座標の差は 1 回変換をするごとに $1/a$ 倍または $1/b$ 倍ずつ伸びていく。n 回変換後では $\xi_a^{n-r}\xi_b^r$ に対して $(1/a)^{n-r}(1/b)^r$ 倍となる。そこで式 (4) に従ってその対数をとり、1 変換当りの量にしたのがこのパイこね変換の場合の伸びる方向に対する局所的な軌道拡大率であり

$$\lambda = -(1 - \frac{r}{n}) \ln a - \frac{r}{n} \ln b \tag{6}$$

となる。

次に式 (4) の平均操作

$$\langle \lambda \rangle = \sum_{r=0}^{n} {}_nC_r a^{n-r} b^r \lambda \tag{7}$$

をする。式 (6) になる確率は、

$${}_nC_r a^{n-r} b^r \tag{8}$$

で与えられる。n が非常に大きくなると、実際には式 (8) の最大値しか平均に寄与しなくなる。そこで、式 (8) が最大値をとるときの r を求める。まず。式 (8) の対数をとりスターリングの公式を用いると

$$n \ln n - (n-r) \ln(n-r) - r \ln r + (n-r) \ln a + r \ln b \tag{9}$$

が得られる。よって、式 (9) が最大値をとるときの r を求めればよい。それには、式 (9) を r で微分したものが 0 になるような r を求める。r で微分して

$$\ln(n-r) - \ln r - \ln a - \ln b = 0 \tag{10}$$

とおくと、$a = 1 - b$ であることから

$$r = bn \tag{11}$$

が求まる。よって、正方形内のほとんどすべての $\vec{x}$ に対して $n \to \infty$ では軌道は $\lim_{n\to\infty} \frac{r}{n} = b$ であるような細片の間に入るようになる。

従って、すべての x に対しては式 (6),(7) より

$$\lim_{n\to\infty} \lambda = \langle \lambda \rangle = -a \ln a - b \ln b \tag{12}$$

となる。これがこの系の軌道拡大率である (図 4)。

次に、コンピュータを用いてパイこね変換のシミュレーションを行った。図 5 は変換によるある点の軌跡である。

そして本研究ではこの変換の軌道拡大率として、

$$\lambda = \frac{\ln L_i - \ln L_0}{i} \tag{13}$$

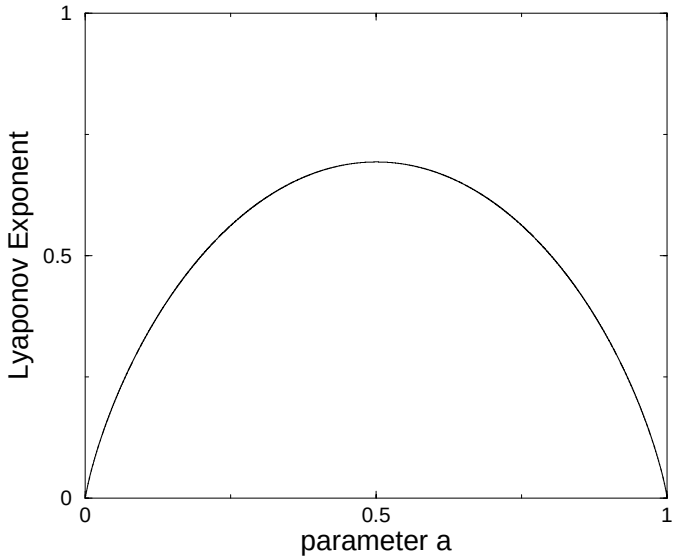

図 4: パイこね変換の軌道拡大率の理論値

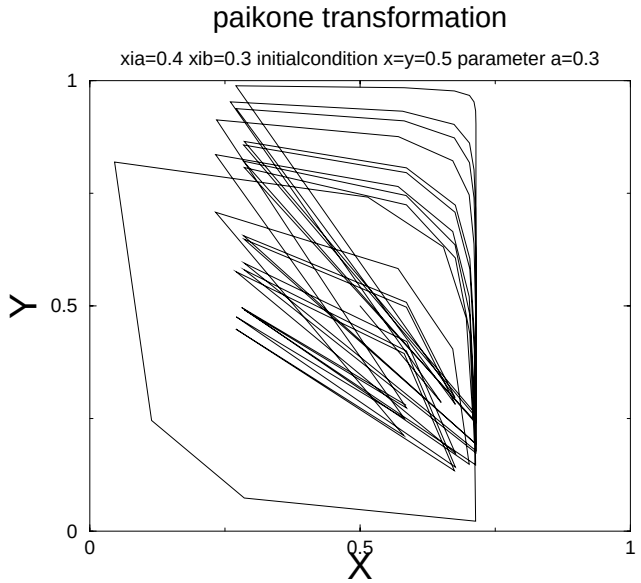

図 5: パイこね変換の軌道の軌跡。初期値は $x = y = 0.5$、$a = 0.3$、$\xi_a = 0.4$、$\xi_b = 0.3$ である。

を計算した。ここで、L_n は初期値の状態で微少距離 $L_0 = \delta$ だけ離れた 2 点間の距離、i は L がある程度の大きさを持つまでの変換の回数である。ここである程度の大きさということについて述べる。カオス状態では近接点の距離は時間発展とともに指数関数的に増大していくことが知られている。しかし、2 節のカオスの特徴でも述べたように軌道は有界な範囲から出ることはない。従ってその系の中では 2 点間の距離の最大値が存在することになる。例えば、このパイこね変換の場合は 1 辺 1 の正方形から出ることはない。すなわちこの系の場合の 2 点間の距離の最大値は $\sqrt{2}$ である。このようにある時間経過してしまうとそれ以上距離が指数関数的に増大することがなくなる。そこが、ある程度の大きさということである。このパイこね変換の場合は、シミュレーションした結果ほとんどの場合 i が 20 前後であることがわかった (図 6〜8)。そこで本研究では $i = 22$ とおいた。そして a, b をパラメーターとして変化させパイこね変換の軌道拡大率をもとめた（図 9)。

ここで先に求めたパイこね変換の軌道拡大率の理論値とシミュレーションにより求めた実験値を比較してみる。すると非常に良く一致していることがわかる（図 10)。また図 10 からもわかるようにパイこね変換の軌道拡大率は常に正の値をとっている。このことからもパイこね変換は常にカオス状態にあるということが出来る。

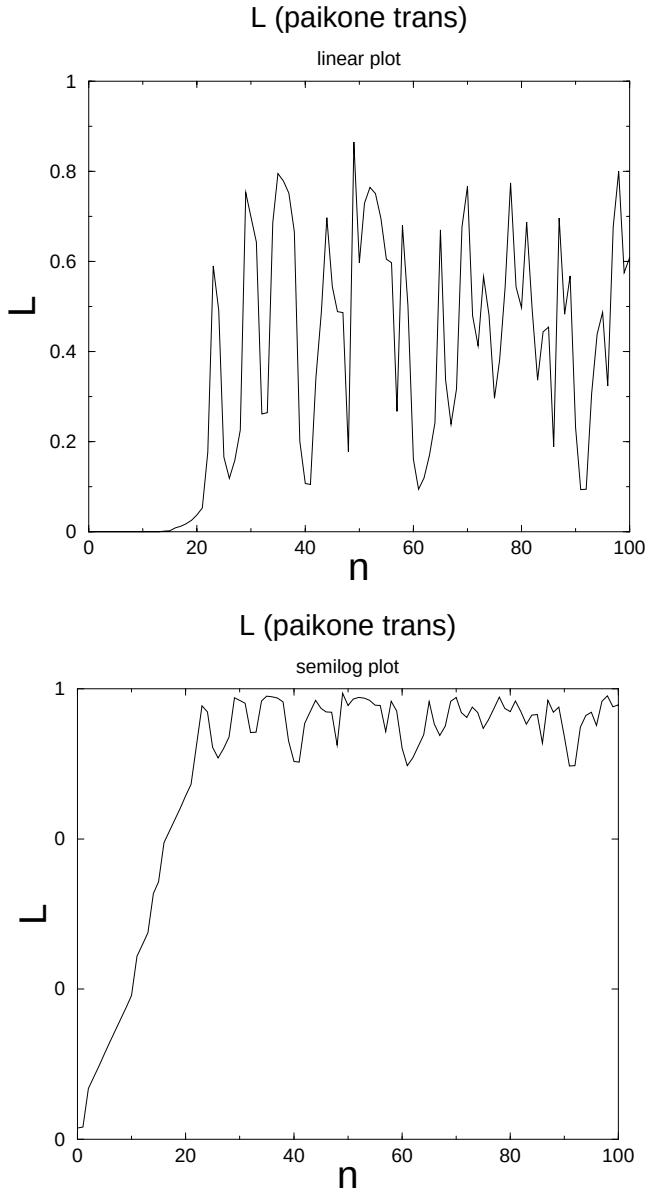

図 6: パイこね変換による 2 点間の距離のグラフ。ただし $\xi_a = 0.4, \xi_b = 0.3, a = 0.3$, 初期値 $x_0 = y_0 = 0.5, x_0' = y_0' = 0.50001$ とした。上は線形プロットで下は片対数プロット。片対数プロットで $0 \leq n \leq 20$ 付近の直線の傾きが軌道拡大率を与える。

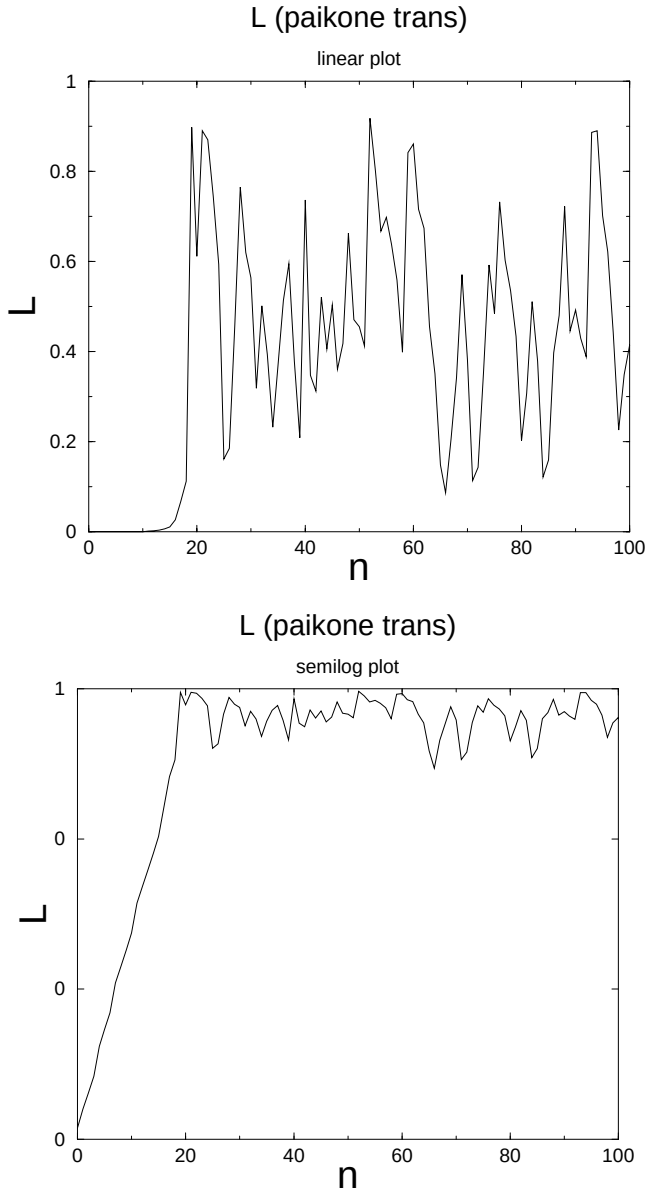

図 7: パイこね変換による 2 点間の距離のグラフ。ただし $\xi_a = 0.4, \xi_b = 0.2, a = 0.4$, 初期値 $x_0 = y_0 = 0.3, x'_0 = y'_0 = 0.30001$ とした。上は線形プロットで下は片対数プロット。片対数プロットで $0 \leq n \leq 20$ 付近の直線の傾きが軌道拡大率を与える。

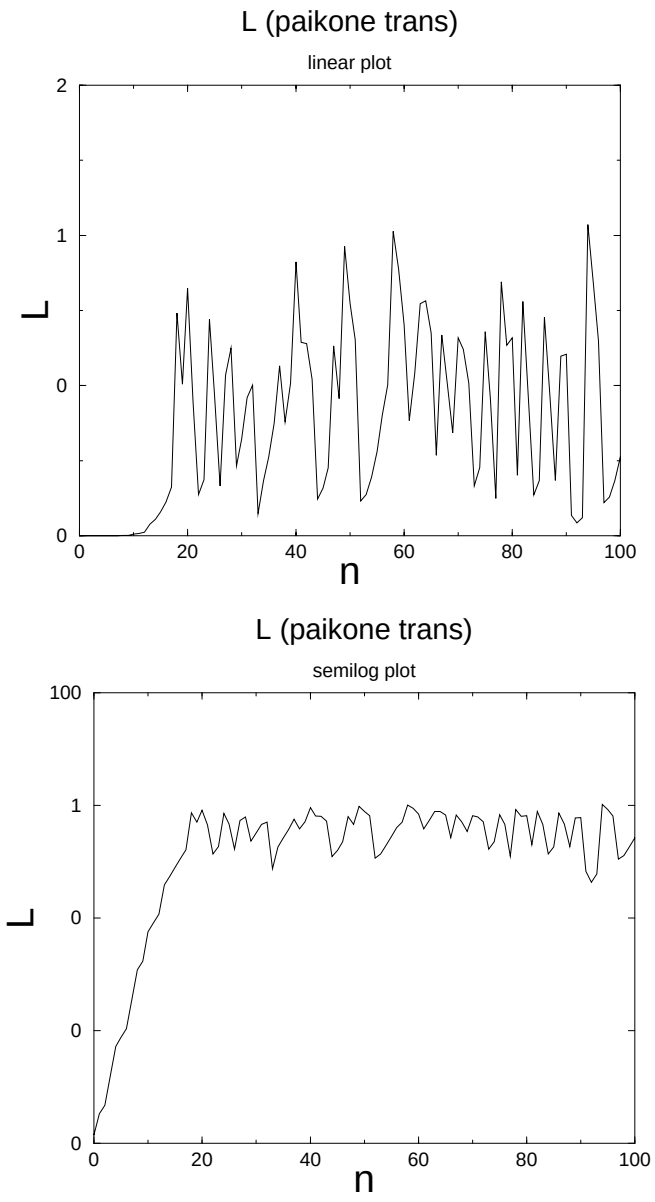

図 8: パイこね変換による 2 点間の距離のグラフ。ただし $\xi_a = 0.1, \xi_b = 0.3, a = 0.7$, 初期値 $x_0 = y_0 = 0.9, x'_0 = y'_0 = 0.90001$ とした。上は線形プロットで下は片対数プロット。片対数プロットで $0 \leq n \leq 20$ 付近の直線の傾きが軌道拡大率を与える。

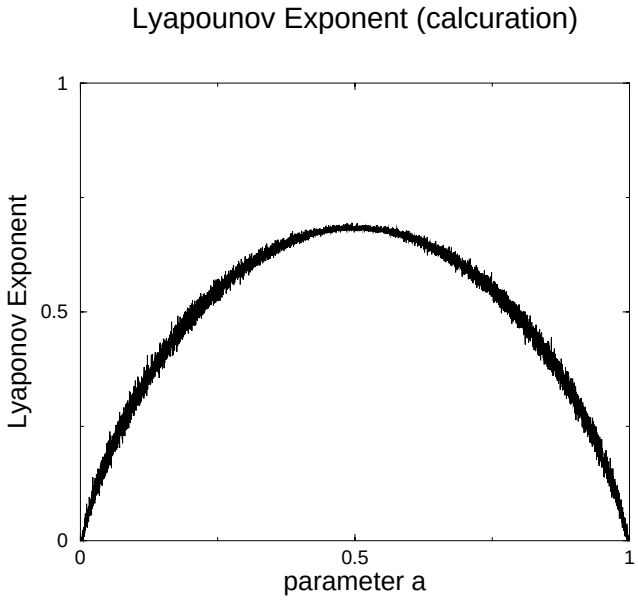

図 9: コンピュータでのシミュレーションにより得たパイこね変換の軌道拡大率。

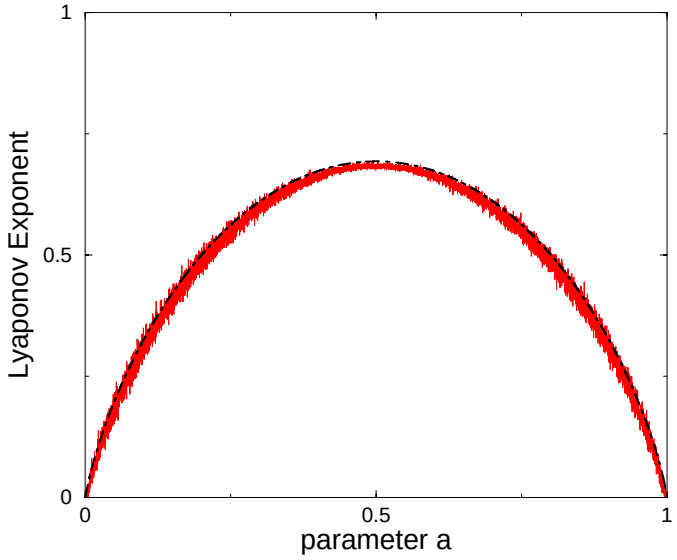

図 10: 軌道拡大率の実験値と理論値を比較したグラフ。実線が実験値で点線が理論値。

6　ローレンツ方程式のシミュレーション（定常状態とカオス状態の両方をとる系）

５節でパイこね変換の軌道拡大率の理論値と実験値が非常に良く一致することが確かめられた。そこで同じ方法を用いて定常状態とカオス状態の両方をとる系であるローレンツ方程式の軌道拡大率をシミュレーションによって求めその振舞いを調べる。本研究では微分方程式であるローレンツ方程式をコンピュータで計算する際に

$$\begin{cases} \frac{dx}{dt} = -\sigma(x-y) \\ \frac{dy}{dt} = -xy + rx - y \\ \frac{dz}{dt} = xy - bz \end{cases} \tag{14}$$

をオイラー差分して、

$$\begin{cases} \frac{x_{t+1}-x_t}{\Delta t} = -\sigma(x-y) \\ \frac{y_{t+1}-y_t}{\Delta t} = -xy + rx - y \\ \frac{z_{t+1}-z_t}{\Delta t} = xy - bz \end{cases} \tag{15}$$

として計算を行った。また、ここではシミュレーションを行う上でローレンツ方程式が定常状態のとき（図 11）とカオス状態のとき（図 12）とに分けて考えた。

なぜならば、初期値の段階で微少距離だけ離れた 2 点間の距離の振舞いが全く異っていることがシミュレーションを行うとわかるからである（図 13, 図 14）。定常状態とカオス状態の区別の仕方として、微少距離だけ離れた 2 つの点を考え、シミュレーション後にその 2 点間の距離がシミュレーション前に比べて大きいかまたは小さいかを調べる。そして、小ければ定常状態とし大きければカオス状態として軌道拡大率を求めた。

図 15, 図 16 はそれぞれ 2 点間の距離のグラフ図 13, 図 14 を片対数プロットしたものである。このとき軌道拡大率はこのグラフの直線的振舞いをしている部分の傾きである。今回のコンピュータシミュレーションでは、初期値を乱数でランダムに与えてそれにより得られた軌道拡大率の平均をとることとした。そしてパラメーター r を変化させてそのときの軌道拡大率の様子を調べる。すると図 17 のようなグラフを得られた。

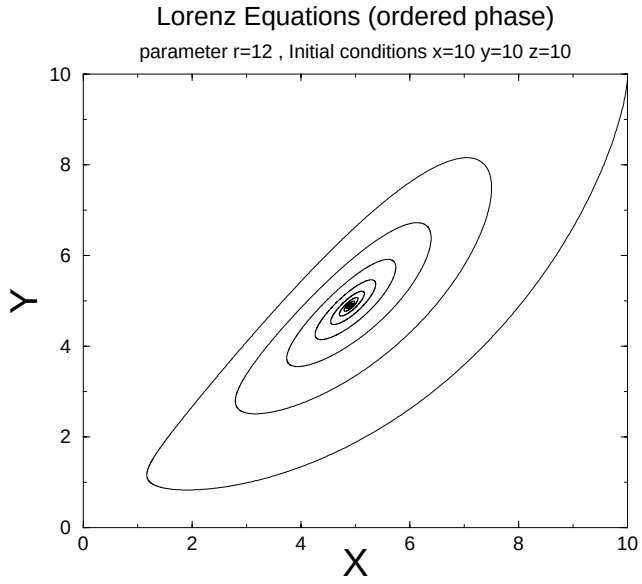

図 11: ローレンツ方程式が定常状態のとき（$r = 12$）の X-Y グラフ。

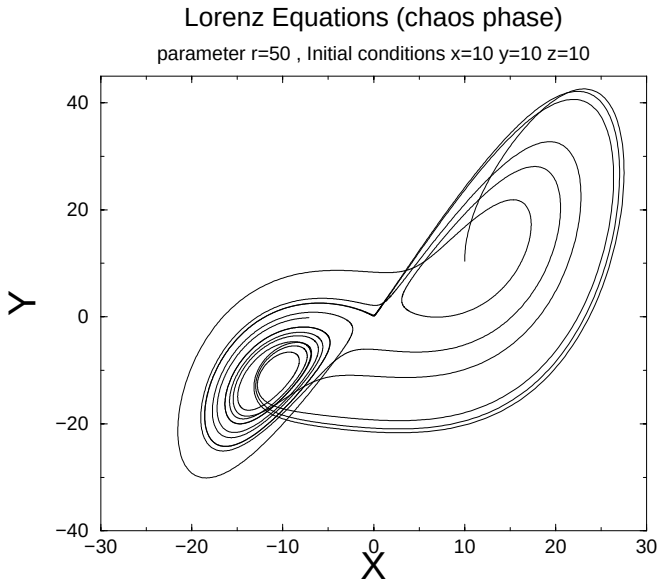

図 12: ローレンツ方程式がカオス状態のとき（$r = 50$）の X-Y グラフ。

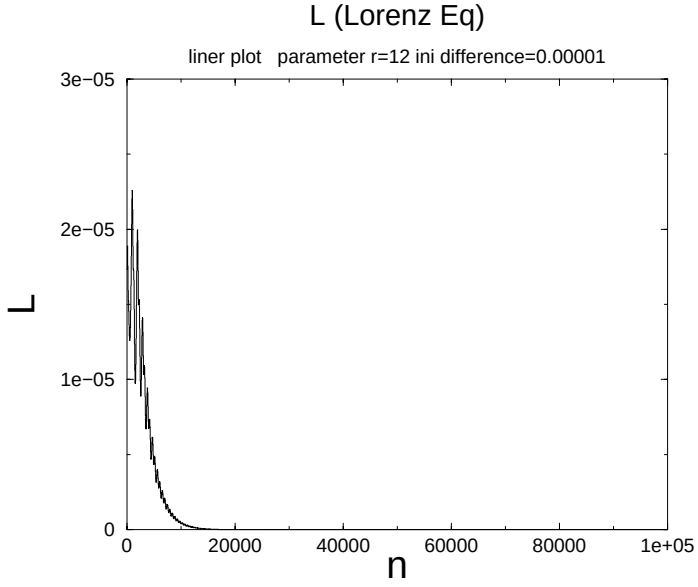

図 13: ローレンツ方程式による 2 点間の距離のグラフ。$r = 12$（定常状態）のとき。ただし、初期値は $x_0 = y_0 = z_0 = 10, x_0' = y_0' = z_0' = 10.00001$ である。

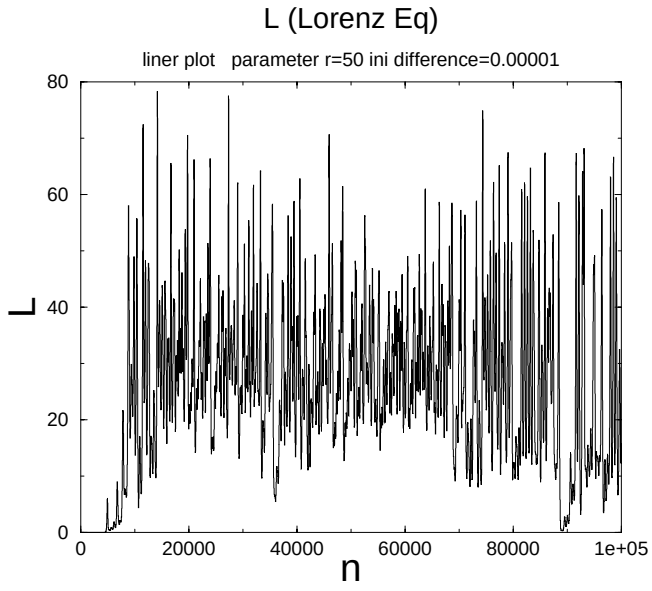

図 14: ローレンツ方程式による 2 点間の距離のグラフ。$r = 50$（カオス状態）のとき。ただし、初期値は $x_0 = y_0 = z_0 = 10, x_0' = y_0' = z_0' = 10.00001$ である。

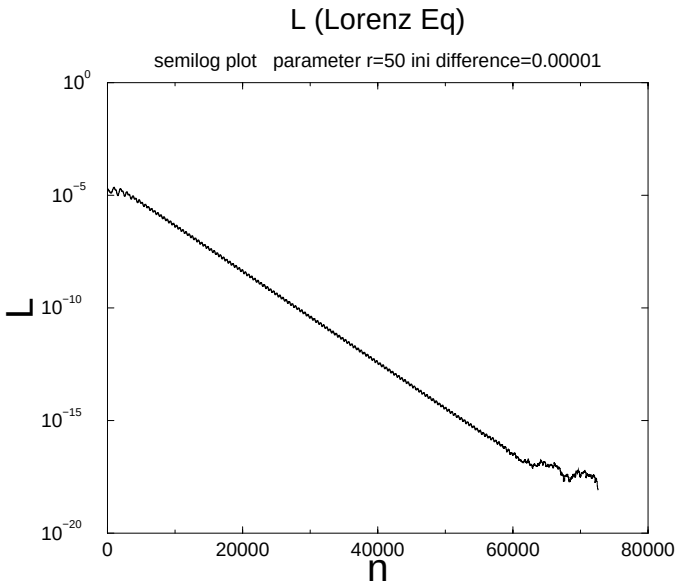

図 15: 図 13 を片対数プロットしたグラフ。$r = 12$（定常状態）のとき。

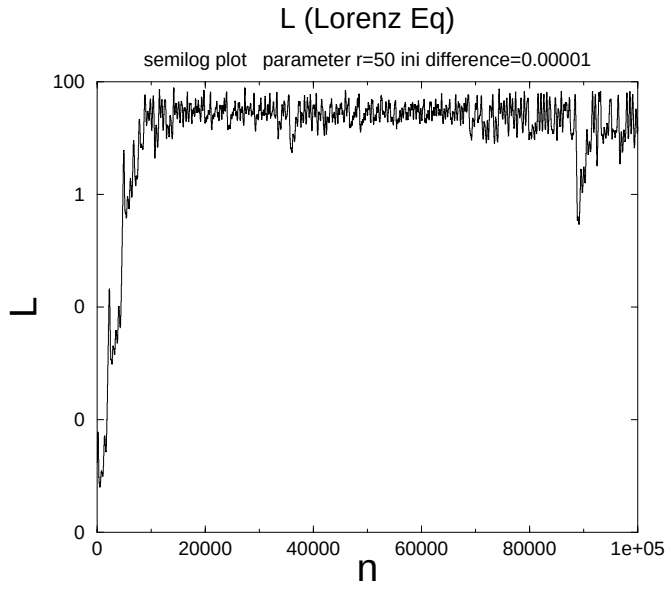

図 16: 図 14 を片対数プロットしたグラフ。$r = 50$（カオス状態）のとき。

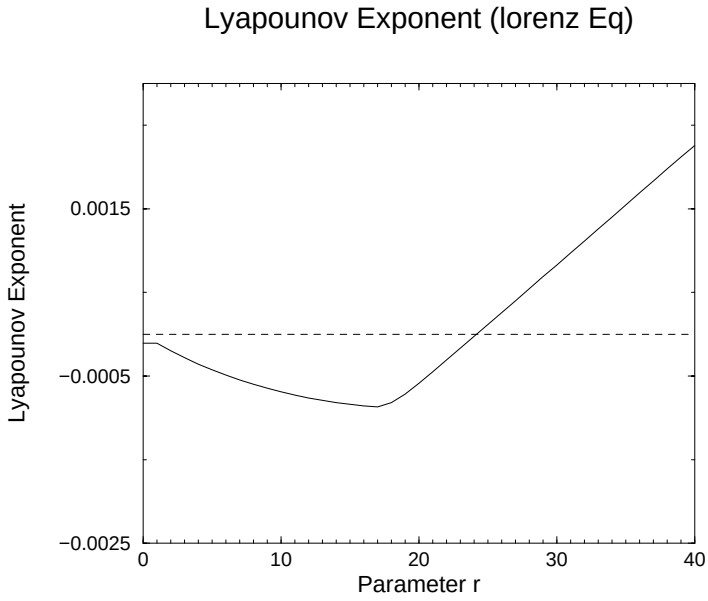

図 17: コンピュータのシミュレーションで得たローレンツ方程式の軌道拡大率。

7　まとめと問題点

５節でも説明したようにパイこね変換のコンピュータシミュレーションにより得た軌道拡大率は理論値と非常に良く一致した（図 4, 図 9, 図 10）。このことより今回行ったコンピュータシミュレーションにより軌道拡大率を正確に求めることができたといえる。また、先に述べたがパイこね変換の場合、変換の回数を $n = 22$ のところでの距離から軌道拡大率を求めたがそれも正しく有効であったと考える。

ローレンツ方程式のコンピュータシミュレーションについての結果は６節でも触れたように図 17 のようになる。この図 17 を解析すると、軌道拡大率はおよそ 25 あたりで負から正に変っていることを読み取ることが出来る。これは４節で説明したように、系の状態が定常状態からカオス状態への相転移が起っていることを意味している。ローレンツ方程式は一般にパラメーター r が $24.74 < r < 145$ のときにカオス状態になるということがいわれている。本研究のコンピュータシミュレーションにより得られた軌道拡大率のグラフ図 17 もそれを証明している。このことから、今回のシミュレーションが正確であったことがわかる。また、このローレンツ方程式が一番安定状態になるのはパラメーター r が $r = 0$ のところではなく r が $r = 17$〜18 の付近であるということも図 17 から読み取ることができる。

最後に、シミュレーションについての問題点を挙げると、ローレンツ方程式の解をコンピュータで計算するために微分方程式であるローレンツ方程式を差分化して近似した。その近似法として１次の近似であるオイラー差分を用いたため、誤差があると思われる。この改善策としては、より精度の高いルンゲ-クッタ法などを用いれば良い。また、もう１つの問題点としては dt の取りかたが挙げられる。本来 dt は無限小であるのでコンピュータで差分近似で計算する際には dt は小さければ小さいほど精度があがる。だが本研究ではコンピュータの性能の問題で計算に時間がかかってしまうためにさほど小さくとることができなかった（実際に dt は 10^{-3} である）。しかし１次の近似法を用いて dt もそこまで小さい値ではないにもかかわらず結果としてはかなり良い値が得られた。

このように、軌道拡大率を求めることによりその系が定常状態にあるのかもしくはカオス状態にあるのかを知ることが出来る。また、何らかのパラメーターを変化させると系の状態が相転移を起すかどうかや、逆に相転移を起させるパラメータは何かを求めることもできる。これは、複雑な振舞いをし予測不可能であるカオスの性質を持つ様々な自然現象の

構造を定量的に解析する上で非常に重要であるといえる。

8　謝辞

一年間この研究を進めてくる上で、私に解りやすく丁寧に指導して下さった羽田野 直道 先生に深く感謝致します。また、研究中に様々なアドバイスやアイディアを出してくれた、同研究室の饗場君、河上君、山崎さんにもこの場をかりてお礼を言いたいと思います。

参考文献

[1] 吉田 健, 別冊 数理科学『現象にひそむ非線形』, サイエンス社 (1989)

[2] 高安 秀樹, フラクタル, 朝倉書店 (1986)

[3] B.A.Huberman , Phys.Rev.Lett.45,3,154(1980)

[4] 合原 一幸, 相澤 洋二 編著, 臨時別冊 数理科学 『カオス研究の最前線』(1999)

[5] 竹山 協三, カオス, 裳華房 (1991)

A　パイこね変換のプログラム

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

struct parameters {
  double xia,xib,a,b;
};

int evolution (double *xo_p, double *yo_p, double *xn_p,
 double *yn_p, struct parameters *param_p);

int main()
{
  FILE *fp1,*fp2,*fp3;
  int i=0,n;
  double xo1,yo1,xn1,yn1,xo2,yo2,xn2,yn2,L;
  struct parameters param;


  /*initial condition*/
  printf("xia:");scanf("%lf",&(param.xia));
  printf("xib:");scanf("%lf",&(param.xib));

  printf("a:");scanf("%lf",&(param.a));

  printf("xo1:");scanf("%lf",&xo1);
  printf("yo1:");scanf("%lf",&yo1);

  printf("xo2:");scanf("%lf",&xo2);
  printf("yo2:");scanf("%lf",&yo2);

  printf("i:");scanf("%d",&n);

  param.b=1.0-param.a;
  printf("b=%f\n",param.b);
```

```
  fp1 = fopen("paikone1.dat","w");
  fp2 = fopen("paikone2.dat","w");
  fp3 = fopen("paikone3.dat","w");

  /*write initial condition to file*/
  fprintf(fp1,"%e %e\n",xo1,yo1);
  fprintf(fp2,"%e %e\n",xo2,yo2);
  L=sqrt((xo2-xo1)*(xo2-xo1)+(yo2-yo1)*(yo2-yo1));
  fprintf(fp3,"0 %e\n",L);

  /*calculation*/
  for(i=1;i<=n;i++){
    evolution(&xo1,&yo1,&xn1,&yn1,&param);
    fprintf(fp1,"%e %e\n",xn1,yn1);
    printf("%e %e\n",xn1,yn1);
    xo1=xn1;
    yo1=yn1;

    evolution(&xo2,&yo2,&xn2,&yn2,&param);
    fprintf(fp2,"%e %e\n",xn2,yn2);
    printf("%e %e\n",xn2,yn2);
    xo2=xn2;
    yo2=yn2;

      /*distance of 1 to 2*/
      L=sqrt((xn2-xn1)*(xn2-xn1)+(yn2-yn1)*(yn2-yn1));

      fprintf(fp3,"%d %e\n",i,L);
      printf("%d %e\n",i,L);
  }
    fclose(fp1);
    fclose(fp2);
    fclose(fp3);
}
```

```
/*paikone program*/
int evolution (double *xo_p, double *yo_p, double *xn_p,
 double *yn_p, struct parameters *param_p){

    if(0 <= *yo_p && *yo_p <= param_p->a){
      *xn_p = param_p->xia * *xo_p;
      *yn_p = *yo_p / param_p->a;
      return 0;
    }
    else if(param_p->a < *yo_p && *yo_p <= 1){
      *xn_p = 0.5 + param_p->xib * *xo_p;
      *yn_p = (*yo_p - param_p->a ) / param_p->b;
      return 0;
    }
    else {
      printf("Error.\n");
      return 1;
    }
}
```

B　パイこね変換の軌道拡大率のプログラム

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

struct parameters {
  double xia,xib,a,b;
};

int evolution (double *xo_p, double *yo_p, double *xn_p,
 double *yn_p, struct parameters *param_p);

int main()
{
  FILE *fp5,*fp6,*fp7;
  int i,j,n,k,l;
  double xo1=0.0,yo1=0.0,xn1=0.0,yn1=0.0;
  double xo2=0.0,yo2=0.0,xn2=0.0,yn2=0.0;
  double sumcallambda=0.0,sumcallambda2=0.0,S=0.0,E=0.0;
  double dt=0.0,Lfirst=0.0,L=0.0;
  double callambda=0.0,thelambda=0.0,avlambda=0.0;
  struct parameters param;

  /* [initial condition] */
  printf("dt;");scanf("%lf",&dt);
  printf("a-loop:");scanf("%d",&l);
  printf("initialcondition-loop:");scanf("%d",&n);

  fp5 = fopen("paikonelambda2.dat","w");
  fp6 = fopen("paikonelambda3.dat","w");
  fp7 = fopen("paikoneError1.dat","w");

  for(k=0;k<l;k++) {
    param.a=(1.0/l)*k;
    param.b=1.0-param.a;
```

```
    sumcallambda=0.0;
    sumcallambda2=0.0;
    S=0.0;
    E=0.0;

    for(j=0;j<n;j++) {

      param.xia=((double)rand()/RAND_MAX)*(0.5);
      param.xib=((double)rand()/RAND_MAX)*(0.5);

      xo1=((double)rand()/RAND_MAX)*(1.0-dt);
      yo1=((double)rand()/RAND_MAX)*(1.0-dt);

      xo2=xo1+dt;
      yo2=yo1+dt;

      /* [first distance of 1to2] */
      Lfirst=dt*sqrt(2);

      /* [calculation] */
      for(i=0;i<22;i++){
evolution(&xo1,&yo1,&xn1,&yn1,&param);

xo1=xn1;
yo1=yn1;

evolution(&xo2,&yo2,&xn2,&yn2,&param);

xo2=xn2;
yo2=yn2;

/* [distance of 1 to 2] */
L=sqrt((xn2-xn1)*(xn2-xn1)+(yn2-yn1)*(yn2-yn1));
      }
```

```
      callambda=(log(L)-log(Lfirst))/i;

      sumcallambda+=callambda;
      sumcallambda2+=callambda*callambda;
    }
    avlambda=sumcallambda/n;

    /* printf("%e %e\n",param.a,avlambda); */
    fprintf(fp5,"%e %e\n",param.a,avlambda);

    thelambda=-(param.a)*log(param.a)
                    -(param.b)*log(param.b);
    /* printf("%e %e\n",param.a,thelambda); */
    fprintf(fp6,"%e %e\n",param.a,thelambda);

    /* [Error of lambda] */
    S=
    sqrt((sumcallambda2-n*(avlambda*avlambda))/(n-1));
    E=S/(sqrt(n));
    /* printf("%e %e\n",param.a,E); */
    fprintf(fp7,"%e %e %e\n",param.a,avlambda,E);
  }

  fclose(fp5);
  fclose(fp6);
  fclose(fp7);
}

/* [paikone program] */
int evolution (double *xo_p, double *yo_p, double *xn_p,
 double *yn_p, struct parameters *param_p){

  if(0 <= *yo_p && *yo_p <= param_p->a){
    *xn_p = param_p->xia * *xo_p;
    *yn_p = *yo_p / param_p->a;
```

```
    return 0;
  }
  else if(param_p->a < *yo_p && *yo_p <= 1){
    *xn_p = 0.5 + param_p->xib * *xo_p;
    *yn_p = (*yo_p - param_p->a ) / param_p->b;
    return 0;
  }
  else {
    printf("Error.\n");
    return 1;
  }
}
```

C　ローレンツ方程式のプログラム

```
#include<stdio.h>

main()
{
  FILE *fp;
  int i,n;
  double dt,x,y,z,xx,yy,zz,R;
  dt=0.001;

  printf("x:");scanf("%lf",&x);
  printf("y:");scanf("%lf",&y);
  printf("z:");scanf("%lf",&z);
  printf("R:");scanf("%lf",&R);
  printf("i:");scanf("%d",&n);

  printf("Initial conditions\n
              x=%f, y=%f, z=%f\n",x,y,z);

  fp = fopen("lorentz4.dat", "w");

       for(i=0;i<n;i++){
 xx=x+(-10*(x-y))*dt;
 yy=y+(-x*z+R*x-y)*dt;
 zz=z+(x*y-2.66667*z)*dt;
 x=xx;
 y=yy;
 z=zz;
   fprintf(fp, "%f %f %f\n",x,y,z);
       }
   fclose(fp);
}
```

D　ローレンツ方程式の軌道拡大率のプログラム

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

struct parameters {
  long double R;
};

int equations (long double *xo_p, long double *yo_p,
 long double *zo_p, long double *xn_p,
long double *yn_p, long double *zn_p,
long double *dt_p, struct parameters *param_p);

long double abs_ld(long double x);

int main()
{
  FILE *fp4;
  int i=0,j=0,n=0,m=0;
  long double dt,delta,k=0.0;
  double a=0.0,b=0.0;
  long double xo1=0.0,yo1=0.0,zo1=0.0,
                   xo2=0.0,yo2=0.0,zo2=0.0;
  long double xn1=0.0,yn1=0.0,zn1=0.0,
                   xn2=0.0,yn2=0.0,zn2=0.0;
  long double D=0.0,L[21];
  long double lambda=0.0,avlambda=0.0,avlambda2=0.0;
  /* sumlambda=0.0,sumlambda2=0.0 */
  long double S=0.0,E=0.0;
  struct parameters param;

  fp4 = fopen("LyaExp_lorentzmodel4.dat","w");

  printf("initialcondition-loop:");scanf("%d",&m);
```

```
printf("parameter.R-loop a to b\n");
printf("a:");scanf("%lf",&a);
printf("b:");scanf("%lf",&b);

delta=0.0001;
printf("difference:%e\n",(double)delta);

n=1000000;
printf("equation-loop:%d\n",n);

dt=0.0001;
printf("dt = %e\n",(double)dt);

for(k=a;k<b;k+=0.01) {
  param.R=k;

  avlambda=0.0;
  avlambda2=0.0;
  S=0.0;
  E=0.0;

  for(j=0;j<m;j++) {

    xo1=((long double)rand()/RAND_MAX)*100;
    yo1=((long double)rand()/RAND_MAX)*100;
    zo1=((long double)rand()/RAND_MAX)*100;

    xo2=xo1+delta;
    yo2=yo1+delta;
    zo2=zo1+delta;

    L[0]=delta*sqrt(3);

    /*calculation*/
    for(i=1;i<=n;i++) {
```

```
equations(&xo1,&yo1,&zo1,&xn1,&yn1,&zn1,&dt,&param);
xo1=xn1;
yo1=yn1;
zo1=zn1;

equations(&xo2,&yo2,&zo2,&xn2,&yn2,&zn2,&dt,&param);
xo2=xn2;
yo2=yn2;
zo2=zn2;

/*distance of 1to 2*/
D=sqrt((xn2-xn1)*(xn2-xn1)
                +(yn2-yn1)*(yn2-yn1)
                +(zn2-zn1)*(zn2-zn1));

if(D<0.00000000000001) {
  break;
}
if(i%50000==0) {
  L[i/50000]=
sqrt((xn2-xn1)*(xn2-xn1)
             +(yn2-yn1)*(yn2-yn1)
             +(zn2-zn1)*(zn2-zn1));
}
      }
      /*calculation of lambda*/
      if(L[0]<D) {
lambda=(log(D)-log(L[0]))/(0.08*n);

      } else {
lambda=(log(L[i/50000])-log(L[(i/50000)-1]))/50000;
      }
       /*      sumlambda+=lambda;
      sumlambda2+=lambda*lambda;   */
      avlambda += (lambda - avlambda) / (j+1);
```

```
      avlambda2  +=  (lambda*lambda - avlambda2 ) /(j+1);

    }
    /* avlambda=sumlambda/m; */

    /*Error of lambda */
    /*    S=sqrt((sumlambda2-m*(avlambda*avlambda))/(m-1));
  E=S/(sqrt(m));   */

    E=sqrt(abs_ld( avlambda2 - avlambda*avlambda)/(m-1));

    fprintf(fp4,"%e %e %e\n",
                (double)param.R,(double)avlambda,(double)E);
    printf("%e %e %e\n",
                (double)param.R,(double)avlambda,(double)E);
  }
  fclose(fp4);
}

int equations (long double *xo_p, long double *yo_p,
 long double *zo_p, long double *xn_p,
 long double *yn_p, long double *zn_p,
 long double *dt_p, struct parameters *param_p) {

  *xn_p =
  *xo_p + ( - 10 * ( *xo_p - *yo_p )) * *dt_p;
  *yn_p =
  *yo_p + ( - *xo_p * *zo_p + param_p->R * *xo_p - *yo_p )
  * *dt_p;
  *zn_p =
  *zo_p + ( *xo_p * *yo_p - ( 8 / 3 ) * *zo_p ) * *dt_p;

}
```

```
long double abs_ld(long double x) {
return x>=0 ? x : -x;
}
```