計算機実習 問題 6.9 - ロジスティック写像のリアプノフ指数

早稲田大学先進理工学部物理学科 B4 藤本將太郎

2014/05/14

1　シミュレーションの目的

系がカオス的かどうか判断する指標として、カオスの重要な性質である初期条件への敏感さを考えることができる。一般に、同一の力学系で異なる初期条件から開始するなら、軌跡差は n の関数として変化するであろう。図 1 にロジスティック写像における差 $|\Delta x_n|$ と n の関係を図示してある。大まかに言って、$\ln|\Delta x_n|$ は n の線形な増加関数であり、このことは、系がカオス的なら軌跡間の距離は指数関数的に大きくなることを意味する。このような発散の特徴は以下に示すリアプノフ (Lyapunov) 指数 λ によって記述され、関係式

$$|\Delta x_n| = |\Delta x_0| e^{\lambda n} \tag{1}$$

で定義される。ここで Δx_n は時刻 n における軌跡間の差である。もしリアプノフ指数 λ が正なら、近接した軌跡間の距離も指数関数的に広がり、系はカオス的であると言える。本シミュレーションではロジスティック写像について、このリアプノフ指数を求めることを目的とする。

リアプノフ指数を測定する素朴な方法は、わずかに異なる初期条件のもとで同じ力学系を時間発展させ、n の関数として実際に軌跡の差を測定することである。これらの軌跡が互いに離れていく速さは x_0 の選び方によるので、多くの x_0 の値について離れていく速さを計算する必要がある。x_0 の各値に対して距離を式 (1) に合わせ、それから λ の平均値を求めなければならないので、この方法には手間がかかる。

素朴な方法のさらに重大な限界は、軌跡が 0 から 1 までの区間に制限されているような場合に、n が十分に大きくなると距離 $|\Delta x_n|$ の値はある一定の大きさ以上には大きくなれなくなってしまうことである。しかし、λ の計算を可能な限り正確にするために、可能な限り多くの反復により平均を求めたい。これを実現する方法を説明するために、まず式 (1) を変形して、λ を

$$\lambda = \frac{1}{n} \ln \left| \frac{\Delta x_n}{\Delta x_0} \right| \tag{2}$$

と書く。過渡的な振る舞いが終わった後の全軌跡のデータを使いたいので、

$$\frac{\Delta x_n}{\Delta x_0} = \frac{\Delta x_1}{\Delta x_0} \frac{\Delta x_2}{\Delta x_1} \cdots \frac{\Delta x_n}{\Delta x_{n-1}} \tag{3}$$

という関係に着目すると、λ を

$$\lambda = \frac{1}{n} \sum_{i=0}^{n-1} \ln \left| \frac{\Delta x_{i+1}}{\Delta x_i} \right| \tag{4}$$

と表すことができる。式 (4) は、任意の i について x_i を初期条件として考えることができるということを意味している。

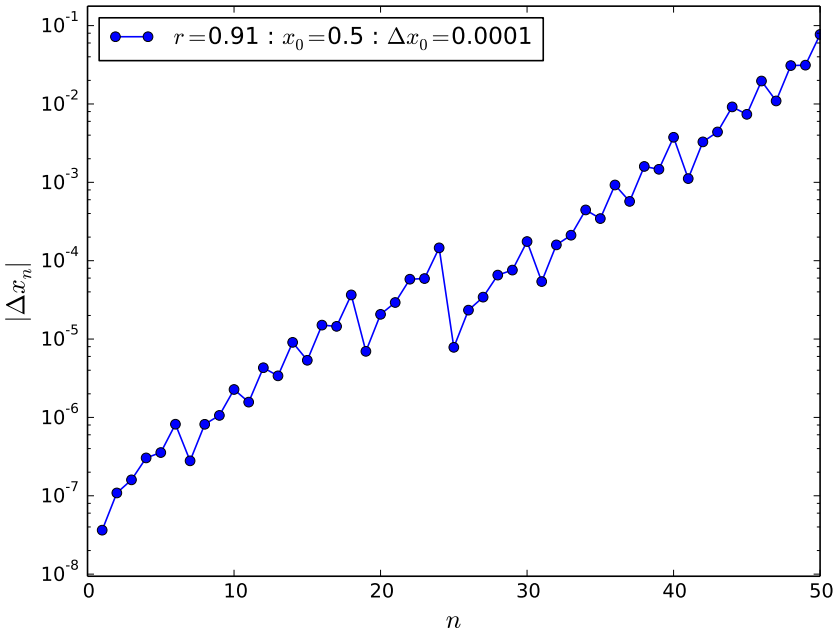

図 1　$r = 0.91$ のときのロジスティック写像の $x_0 = 0.5$ と $x_0 = 0.5001$ で開始した軌跡間の差 Δx_n の変化

式 (4) から、λ を計算する問題は比 $\Delta x_{i+1}/\Delta x_i$ を求めることに帰着することが分かる。2 つの軌跡間の最初の差を可能な限り小さくしたいので、極限 $\Delta x_i \to 0$ を考えて、微分 dx_i を計算すればよい。例としてロジスティック写像

$$f(x_i) = 4rx_i(1 - x_i) \tag{5}$$

の微分は

$$\frac{dx_{i+1}}{dx_i} = f'(x_i) = 4r(1 - 2x_i) \tag{6}$$

であるが、問題 6.2 で行ったようにロジスティック写像の反復を行い、その反復ごとに関係 (6) と x_i の値を使えば、dx_{i+1}/dx_i を計算することができる。したがってリアプノフ指数は

$$\lambda = \lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} \ln |f'(x_i)| \tag{7}$$

で与えられることとなる。ここで、式 (7) の和は過渡的な振る舞いが終わってからとり始める。n を十分に大きく選ぶことを意識するために、式 (7) には $n \to \infty$ の極限を明示的に書いた。この方法はアトラクタ上の点に正しく重みを付けていて、軌跡がアトラクタのある特定の領域を多くは訪れない場合に、その領域は式 (7) の和にあまり寄与しない。

2　作成したプログラム

本シミュレーションで作成したプログラムを以下に示す。

2.1 パラメータの設定ダイアログ (SetParameter.py)

以前作成したプログラムをモジュールとして用いる。self.show_setting_window() によって、その引数として与えられたパラメータのリストの数に応じてパラメータの入力エントリが生成され、self.entry[i].get() で i 番目の要素の取得ができる。OK ボタンを押した時の挙動は、command に関数を指定することで決めることができる。

```
1  #!usr/bin/env python
2  # -*- coding: utf-8 -*-
3  #
4  # written by Shotaro Fujimoto, Mqy 2014.
5  #
6
7  from Tkinter import *
8
9
10 class SetParameter():
11
12     def show_setting_window(self, parameters, commands):
13         """ Show a parameter setting window.
14
15         parameters: A list of dictionaries {'parameter name':default_value}
16         commands: A dictionary {'name of button': command}
17         """
18         self.root = Tk()
19         self.root.title('Control Widget')
20
21         frame1 = Frame(self.root, padx=5, pady=5)
22         frame1.pack(side='top')
23         self.entry = []
24         for i, parameter in enumerate(parameters):
25             label = Label(frame1, text=parameter.items()[0][0] + ' = ')
26             label.grid(row=i, column=0, sticky=E)
27             self.entry.append(Entry(frame1, width=10))
28             self.entry[i].grid(row=i, column=1)
29             self.entry[i].delete(0, END)
30             self.entry[i].insert(0, parameter.items()[0][1])
31         self.entry[0].focus_set()
32
```

```
33          frame2 = Frame(self.root, padx=5, pady=5)
34          frame2.pack(side='bottom')
35          for name, command in commands.items():
36              button = Button(frame2, text=name, command=command)
37              button.pack(side='left', fill='x')
38
39          self.root.mainloop()
40
41      def quit(self):
42          self.root.destroy()
```

2.2 素朴な方法によりリアプノフ指数を計算するプログラム

2.2.1 6-9_lyapunov-a0.py

$\ln|\Delta x_n/\Delta x_0|$ を n に対してプロットするプログラム。このプログラムを実行すると、まず、先の SetParameter を利用して、各パラメータを代入し、その値を用いてロジスティック写像の時間発展を計算する。その際同時に $\ln|\Delta x_n/\Delta x_0|$ の計算も行い、配列に格納する。最後に横軸 n、縦軸 $\ln|\Delta x_n/\Delta x_0|$ としてグラフを描画し、表示させている。

```
1   #!/usr/bin/env python
2   # -*- coding: utf-8 -*-
3   #
4   # written by Shotaro Fujimoto, May 2014.
5   #
6   # 計算機実習
7   # 問題 6.9 リアプノフ指数
8   # -a0
9   # \ln | \Delta x_{n} / \Delta x_{0} |のグラフの描画
10
11  import matplotlib.pylab as plt
12  import array as array
13  import numpy as np
14  import SetParameter as sp  # 同じディレクトリに SetParameter.py を置く
15
16
17  def assignment():
18      global r, x0, delta_x0, nmax
19      r = float(window.entry[0].get())
20      x0 = float(window.entry[1].get())
21      delta_x0 = float(window.entry[2].get())
```

```
22      nmax = int(window.entry[3].get())
23      window.quit()
24      mapping(r, x0, delta_x0, nmax)
25
26
27  def func(x_i, r):
28      return 4.0 * r * x_i * (1 - x_i)
29
30
31  def mapping(r, x0, delta_x0, nmax):  # グラフの描画
32      x1 = array.array('d')
33      x1.append(x0)
34      x2 = array.array('d')
35      x2.append(x0 + delta_x0)
36      lya = array.array('d')
37      for count in range(nmax):
38          x1.append(func(x1[-1], r))
39          x2.append(func(x2[-1], r))
40          l = np.log(abs((x2[-1] - x1[-1]) / delta_x0))
41          lya.append(l)
42      n = range(1, nmax + 1)
43      plt.plot(n, lya,
44               label=r'$r=$' + str(r) + ' : '
45               + r'$x_{0}=$' + str(x0) + ' : '
46               + r'$\Delta x_{0}=$' + str(delta_x0)
47               )
48      plt.gca().set_xlim(0, nmax)
49      plt.gca().set_ylim(min(lya) - 0.3, max(lya) + 0.3)
50      plt.xlabel(r'$n$', fontsize=16)
51      plt.ylabel(r'$\ln | \Delta x_{n} / \Delta x_{0}| $', fontsize=16)
52      plt.title('Graph of 'r'$\ln | \Delta x_{n} / \Delta x_{0} |$')
53      plt.legend(loc="best")
54      plt.show()
55
56  if __name__ == '__main__':
57      parameters = [{'r': 0.91}, {'x0': 0.5},
58                    {'delta_x0': 0.000001}, {'nmax': 200}]
59      window = sp.SetParameter()
60      window.show_setting_window(parameters, {'OK': assignment})
```

2.2.2 6-9_lyapunov-a.py

このプログラムを実行すると、まず設定値 r、Δx_0、dx の設定ダイアログが表示され、値を入れて OK ボタンを押すと、$0 < x_0 < 1$ の範囲で dx の間隔でリアプノフ指数が計算され、横軸 x_0、縦軸 λ のグラフと、その平均値が表示される。プログラムの中身については、まず、与えられた dx、Δx_0 を元に CalculateLambda 内の関数 set_parameter によって初期値 x_0 と $x_0 + \Delta x_0$ の配列が作成される。次にその各値について lambda_for_x0 が適用されて λ が計算され、インスタンス配列 self.lyapunovs に格納される。最後に plot_x0_lambda と plot_average_lambda がその値を用いてグラフを描画するという仕組みになっている。数値計算の肝となる CalculateLambda の部分については次項で説明する。

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  #
4  # written by Shotaro Fujimoto, May 2014.
5  #
6  # 計算機実習
7  # 問題 6.9 リアプノフ指数
8  # -a 素朴な方法でロジスティック写像のリアプノフ指数を求める
9  # 例:
10 #   r        = 0.91    : control parameter
11 #   delta_x0 = 0.00005 : width of initial values
12 #   dx       = 0.001   : step size of x0
13
14
15 import matplotlib.pylab as plt
16 import SetParameter as sp  # 同じディレクトリに SetParameter.py を置く
17 import CalculateLambda as cl  # 同じディレクトリに CalculateLambda.py を置く
18
19
20 def assignment():
21     global r, delta_x0, dx
22     r = float(window.entry[0].get())
23     delta_x0 = float(window.entry[1].get())
24     dx = float(window.entry[2].get())
25     window.quit()
26
27
28 def plot_x0_lambda(x0, lyapunovs):
29     plt.plot(x0, lyapunovs,
30              label='\n' + r'$r=$' + str(r) + '\n'
```

```
31                + '$\Delta x_{0}\ (:\mathrm{width\ of\ initial\ values})=$'
32                + str(delta_x0) + '\n'
33                + '$dx\ (:\mathrm{step\ size\ of\ }x_{0})=$'
34                + str(dx)
35                )
36
37
38  def plot_average_lambda(x0, lyapunovs):
39      ave_lambda = sum(lyapunovs) / len(x0)
40      plt.plot([0, 1], [ave_lambda] * 2,
41               label=r'$\mathrm{average\ of\ } \lambda=$' + str(ave_lambda)
42               )
43      plt.gca().set_xlim(0, 1)
44      plt.gca().set_ylim(min(lyapunovs) - 0.1, max(lyapunovs) + 0.1)
45      plt.xlabel(r'$x_{0}$', fontsize=16)
46      plt.ylabel(r'$\lambda$', fontsize=16)
47      plt.title('Lyapunov index')
48      plt.legend(loc="best")
49
50  if __name__ == '__main__':
51
52      window = sp.SetParameter()
53      parameters = [{'r': 0.91}, {'delta_x0': 0.000001}, {'dx': 0.01}]
54      window.show_setting_window(parameters, {'OK': assignment})
55
56      calculate = cl.CalculateLambda()
57      calculate.set_parameter(r, delta_x0, dx)
58      calculate.prepare_x0_and_lambda()
59
60      plot_x0_lambda(calculate.x0, calculate.lyapunovs)
61      plot_average_lambda(calculate.x0, calculate.lyapunovs)
62
63      plt.show()
```

2.2.3 CalculateLambda.py

このプログラムは 6-9_lyapunov-a.py から呼び出されるモジュールとして機能しており、これに具体的な値を代入して関数の部分それぞれを独立に実行させることができる。lambda_for_x0 の中で実際に λ が計算されおり、いくつかの試行により先に与えた nplot までの n について、最小二乗法で直線へのフィッティングを行い、その傾きを戻り値となるようにしてある。このプログラムでの nplot の選び方は非常に恣意的で、しか

もこの値によって λ の値は大きく変わりうる。本来ならばプログラム中で有効な範囲を判断してその範囲でフィッティングを行うべきであるが、その過程を自動化する良い解決策が見つからなかった。あるいは x_0 の値ごとに毎回グラフを表示して人の目で適用範囲を決めていくこともできるが、それでは良いプログラムとは言えない。この点において改良の必要がある。

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  #
4  # written by Shotaro Fujimoto, May 2014.
5  #
6
7  import array as array
8  import numpy as np
9  import scipy.optimize as optimize
10
11
12 class CalculateLambda():
13
14     def set_parameter(self, r, delta_x0, dx):
15         self.r = r
16         self.delta_x0 = delta_x0
17         dx = dx
18
19         self.n = int((1.0 - self.delta_x0) / dx)
20         self.x0 = np.array([(n + 1) * dx for n in range(self.n)])
21         self.x0_plus_delta_x0 = self.x0 + self.delta_x0
22         return None
23
24     def prepare_x0_and_lambda(self):
25
26         self.lyapunovs = np.array([self.lambda_for_x0(x1, x2) for x1, x2
27                                    in zip(self.x0, self.x0_plus_delta_x0)
28                                    ])
29
30     def lambda_for_x0(self, x1, x2):
31         r = self.r               # - localize
32         delta_x0 = self.delta_x0  # -
33         nplot = 50  # Note: This value is determined by some trials.
34                     #       It affecs directory the value of lyapunov index.
35
```

```
36          def func(x_i):
37              return 4.0 * r * x_i * (1.0 - x_i)
38
39          ary1 = array.array('d')
40          ary1.append(x1)
41          ary2 = array.array('d')
42          ary2.append(x2)
43          lya = array.array('d')
44          lya.append(np.log(abs((ary2[-1] - ary1[-1]) / delta_x0)))
45          for count in range(nplot - 1):
46              ary1.append(func(ary1[-1]))
47              ary2.append(func(ary2[-1]))
48              l = np.log(abs((ary2[-1] - ary1[-1]) / delta_x0))
49              lya.append(l)
50
51          # -- optimize --
52
53          def fit_func(parameter0, n, lya):
54              a = parameter0[0]
55              b = parameter0[1]
56              residual = lya - (a * n + b)
57              return residual
58
59          parameter0 = [1.0, 1.0]
60              # initial value of parameters a, b (ln||=a*n+b)
61          n = np.arange(1, nplot + 1)
62          result = optimize.leastsq(fit_func, parameter0,
63                                    args=(n, np.array(lya))
64                                    )
65          return result[0][0]
```

2.3 アルゴリズムを用いてリアプノフ指数を計算するプログラム (6-9_lyapunov-b.py)

前述したアルゴリズムに基づいて、リアプノフ指数を求めるプログラムを以下に示す。このプログラムでは、まず dr 刻みの r の配列を作成し、そのそれぞれの r について関数 get_lambda_r を計算する。関数 get_lambda_r は、時間発展で変化する x を計算し、ntransient 回以降のデータのみ切り出し、そのそれぞれの x についての微分係数を計算して、最後にその平均を返している。最終的に、このようにして得られた r と λ の組を使ってグラフを描画する。

```
1   #!/usr/bin/env python
```

```
2  # -*- coding: utf-8 -*-
3  #
4  # written by Shotaro Fujimoto, May 2014.
5  #
6  # 計算機実習
7  # 問題 6.9 リアプノフ指数
8  # -b 教科書で示されたアルゴリズムを用いて
9  # ロジスティック写像のリアプノフ指数を求める。
10 #
11
12 import matplotlib.pylab as plt
13 import array as array
14 import numpy as np
15 import multiprocessing as mp
16
17 # --- set parameters ---
18
19 r0 = 0.76
20 rmax = 1.0
21 dr = 0.001
22 ntransient = 1000
23 n_calc = 100000
24 x0 = 0.5
25
26 nmax = ntransient + n_calc
27
28 pr = mp.cpu_count() * 2
29
30 def lambda_for_r():
31     if (rmax - r0) % dr == 0:
32         count = int((rmax - r0) / dr) - 1
33     else:
34         count = int((rmax - r0) / dr)
35     r = [r0 + dr*n for n in np.arange(count + 1)]
36     r.append(rmax)
37     r = np.array(r)
38     _lambda = array.array('f')
39     pool = mp.Pool(pr)
40
41     _lambda = np.array(pool.map(get_lambda_r, r))
```

```
42      return r, _lambda
43
44
45  def get_lambda_r(r):
46      def function(x_i, r):
47          return 4.0 * r * x_i * (1.0 - x_i)
48
49      x = array.array('d')
50      x.append(x0)
51      for i in np.arange(nmax):
52          x.append(function(x[i], r))
53
54      def operate(cx):
55          return np.log(abs(4.0 * r * (1.0 - 2.0 * cx)))
56
57      cutted_x = x[ntransient:nmax]
58      edited_x = map(operate, cutted_x)
59      lambda_r = sum(edited_x) / len(edited_x)
60      return lambda_r
61
62  r_and_lambda = lambda_for_r()
63  plt.gca().set_xlim(r0, rmax)
64  plt.gca().set_ylim(-2.0, 1.0)
65  plt.scatter(r_and_lambda[0], r_and_lambda[1], color='b', s=0.5, marker='.')
66  plt.plot([r0, rmax], [0, 0], 'r--')
67  plt.xlabel(r'$r$', fontsize=16)
68  plt.ylabel(r'$\lambda$', fontsize=16)
69  plt.title('Lyapunov index')
70  plt.show()
```

3 実習課題

a. ロジスティック写像のリアプノフ指数 λ を素朴な方法で計算せよ。$r = 0.91$、$x_0 = 0.5$、$\Delta x_0 = 10^{-6}$ とし、$\ln|\Delta x_n/\Delta x_0|$ を n に対してプロットせよ。n が大きいときには $\ln|\Delta x_n/\Delta x_0|$ はどうなるか。$r = 0.91$、$r = 0.97$、$r = 1.0$ について λ を求めよ。r の各値に対する λ の値は x_0 や Δx_0 の選び方に強く依存するか。

はじめに、$r = 0.91$、$x_0 = 0.5$、$\Delta x_0 = 10^{-6}$ として $\ln|\Delta x_n/\Delta x_0|$ を n に対してプロットしたものを図 (2) に示す。このグラフから、$\ln|\Delta x_n/\Delta x_0|$ は $1 \leq n \leq 100$ の領域では、n が大きくなるにつれて大まかに見て線形に増加しており、それ以後 n が大きいときには $\ln|\Delta x_n/\Delta x_0|$ はおよそ

11 の周囲で振動していることが分かる。これは、素朴な方法の欠点として前述したとおり、x_n の値に制限があるような今回の場合においては、仮に Δx_n が 1 となり得たとして、

$$\max\left(\ln\left|\frac{\Delta x_n}{\Delta x_0}\right|\right) = \ln\left|\frac{1}{\Delta x_0}\right| \tag{8}$$

であり、今 $\Delta x_0 = 10^{-6}$ としていたため、$\max(\ln|\Delta x_n/\Delta x_0|) = 6\ln 10 \simeq 13.8$ となる。ここで得られたグラフと再び比較すると、$\ln|\Delta x_n/\Delta x_0|$ の極大値は 12 程度であると分かるので、したがって $\max(\Delta x_n)$ は実際には 10^{-1} 程度であることも推測される。

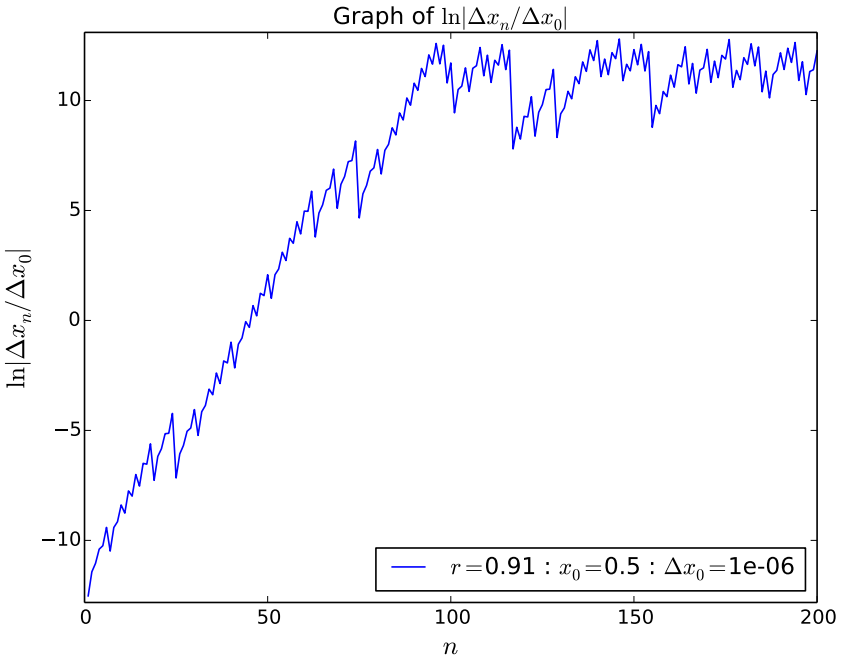

図 2　$n - \ln|\Delta x_n/\Delta x_0|$ のグラフ

次に、$r = 0.91$、$r = 0.97$、$r = 1.0$ について λ を求めることにする。基本的な考え方としては、図 2 で得られたようなグラフを、さまざまな x_0 について試し、グラフで $\ln|\Delta x_n/\Delta x_0|$ が線形に増加している部分の傾きをその x_0 での λ と見ればよい。λ の値が x_0 に依存するかどうか確かめるため、また平均値をとって λ の精度を高めるために、間隔 $dx = 0.01$ おきの x_0 についてそれぞれ λ を求め、横軸を x_0、縦軸を λ としたグラフを描いてみる。$r = 0.91$、$\Delta x_0 = 10^{-6}$ として、傾きをとる区間を n=50 までとしたものを図 3 に示す。また図 4 には $r = 0.97$ でそれ以外の条件を同じにしたものを、図 5 に $r = 1.0$ でそれ以外の数値を同じにしたものを示した。これらのグラフからは、λ の値は x_0 に依存して変動しているが、その取る値は平均値の周辺にまとまって分布しており、なにか決まった関係性があるわけではないということが推測される。しかし、$x_0 = 0.5$ での λ は大きく突出していることに注目する必要があるだろう。

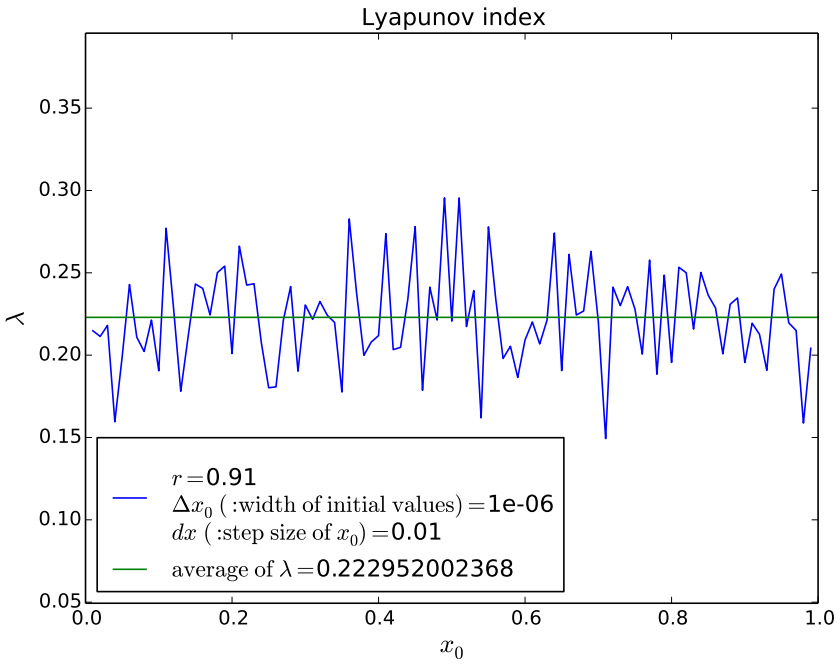

図 3　$r = 0.91$、$\Delta x_0 = 10^{-6}$ としたときの $n - \lambda$ グラフ

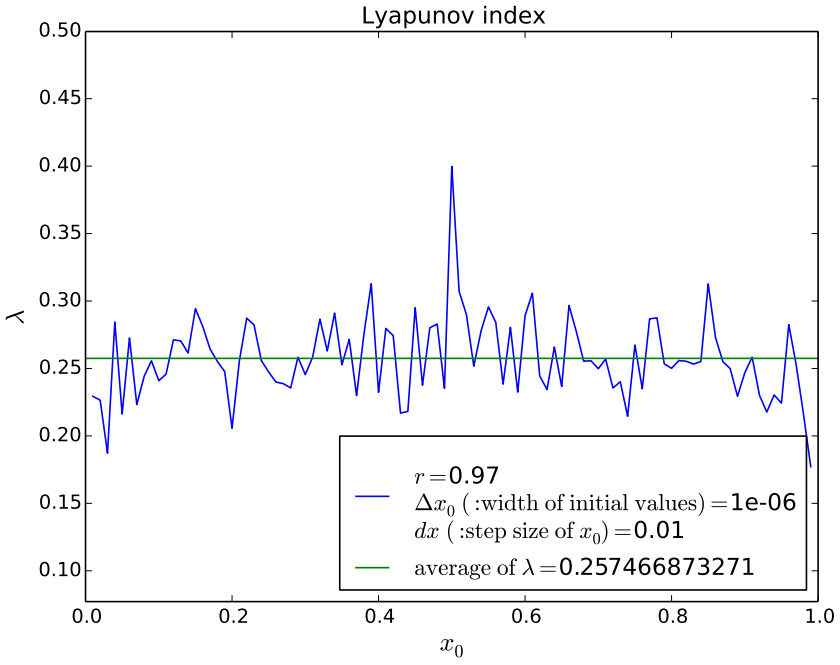

図 4　$r = 0.97$、$\Delta x_0 = 10^{-6}$ としたときの $n - \lambda$ グラフ

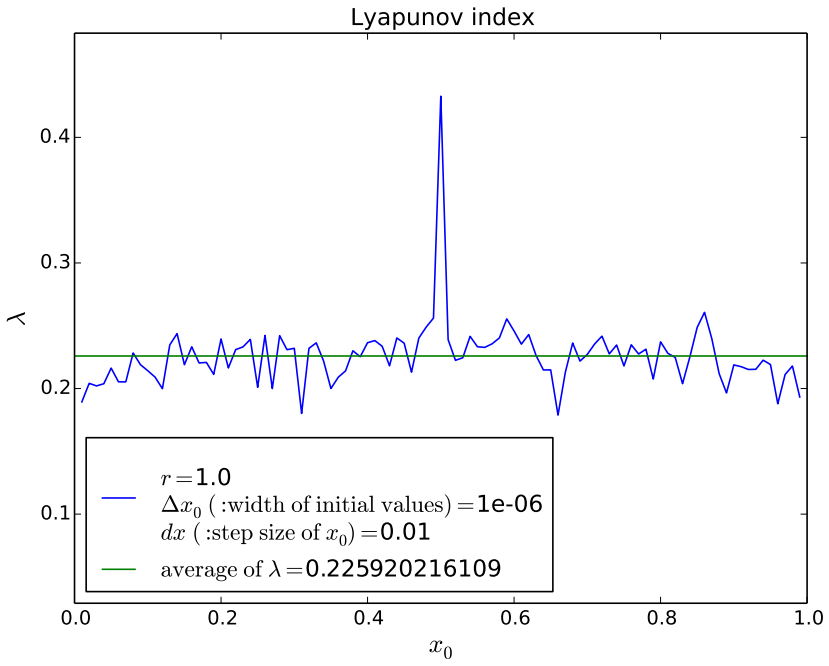

図 5　$r = 1.0$、$\Delta x_0 = 10^{-6}$ としたときの $n - \lambda$ グラフ

では次に、Δx_0 の大きさに関する依存性はどうであろうか。$r = 0.97$ で固定し、Δx_0 を $\Delta x_0 = 10^{-4}$、10^{-6}、10^{-8} とした場合のグラフを図 6、7、8 に示す。このグラフからもわかるように、Δx_0 の大きさは、λ の値に影響を与えていることがわかる。しかし、これは nplot の選び方が一意でないことと、nplot の大きさが小さいために過渡現象の大きな振動の効果が影響してしまうことに起因する。これらの考察からも、素朴な方法でリアプノフ指数を求めることの困難さが理解できる。

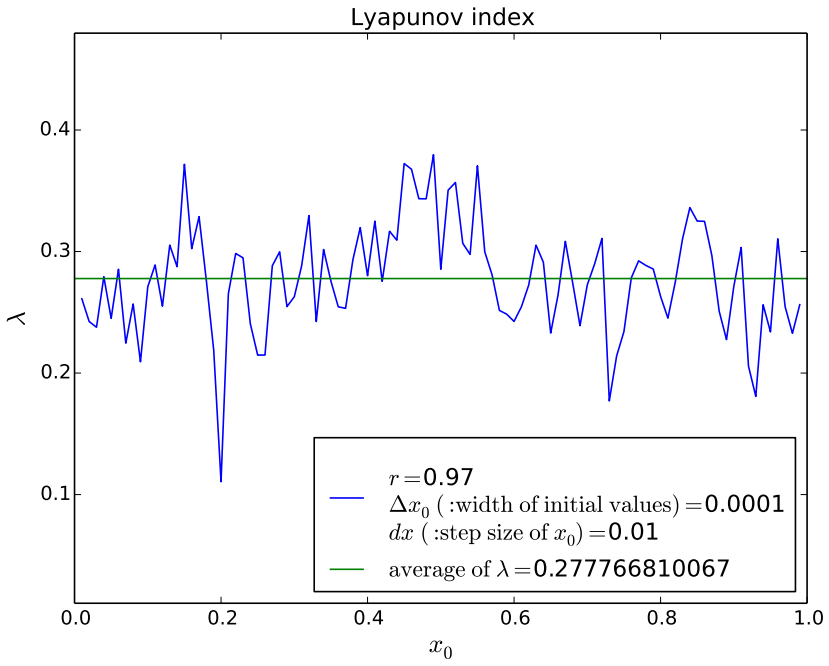

図 6　$r = 0.91$、$\Delta x_0 = 10^{-4}$、nplot=30 としたときの $x_0 - \lambda$ グラフ

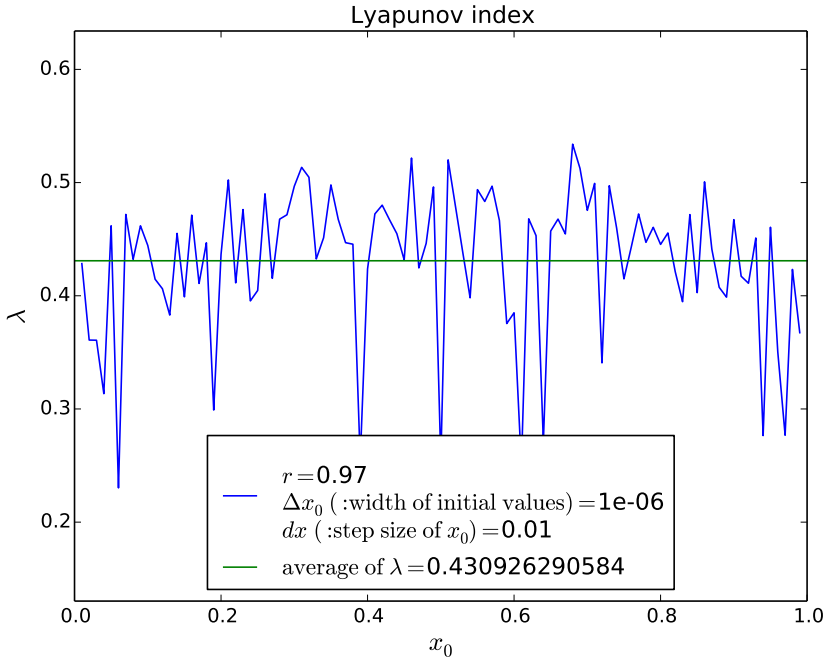

図 7　$r = 0.97$、$\Delta x_0 = 10^{-6}$、nplot=30 としたときの $x_0 - \lambda$ グラフ

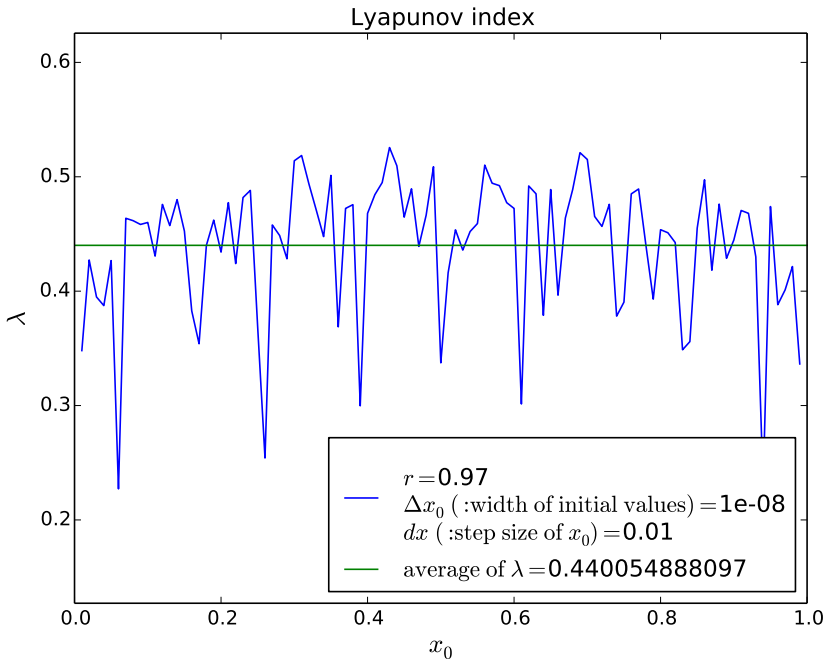

図 8　$r = 1.0$、$\Delta x_0 = 10^{-8}$、nplot=40 としたときの $x_0 - \lambda$ グラフ

b. 本文で述べたアルゴリズムを用いて、$\Delta r = 0.01$ の刻み幅で、$r = 0.76$ から $r = 1.0$ の範囲について、λ を計算せよ。系がカオス的でないとき、λ の符号はどうなるか。λ を r に対してプロットし、問題 6.2 で得られた分岐図と対応させながら、得られた結果を説明せよ。得られた λ の結果を教科書に示されている図 6.9 と比較せよ。λ の符号は分岐図に見られる系の振る舞いとどのような関係があるか。$\lambda < 0$ の場合には、2 つの軌跡は一致し系はカオス的でなくなる。しかし $\lambda = 0$ では、軌跡はべき n で代数的に離れていく。r のどの値に対して λ は最大になるか。

先に示したアルゴリズムを用いて、$x_0 = 0.5$ のとき $\Delta r = 0.01$ の刻み幅で、$r = 0.76$ から $r = 1.0$ の範囲について、λ を計算し、その結果を横軸 r、縦軸 λ のグラフにプロットした。これを図 9 に示す。また、さらに Δr を小さくして $\Delta r = 0.0001$ としたものと、問題 6.2 でロジスティック写像について得られた r に関する分岐図 ($x_0 = 0.5$) とを比較したものを図 10 に示す。この図から明らかなように、r が負のところでは系はカオス的でなく、窓を形成するような r の値においても λ の値は負の値である。また、図の中で点線で示したように、$\lambda = 0$ のときは、ちょうど系の挙動の分岐点にあたることが見て取れる。はじめの λ の定義からも分かるように、この値が負であるということは、初期値がずれていても、十分時間の経過した後には二つの値の間の差はほとんど 0 に等しくなることを意味し、則ち系の安定性を示している。分岐図だけでは、どの程度の安定性があるかまではわからないが、2 つの図を並べて比較すると、$r = 0.81$ と $r = 0.85$ では、$r = 0.81$ の方が系はより安定であることを知ることができる。逆に、λ の値が正であるところでは、図 10 の分岐図ではカオス的挙動を示していて、$r = 1.0$ のときに λ の値が最大となっている。これも先に述べた λ の意味と確かに一致している。これらのことから、系のカオスの度合いを特徴づける指標としてリアプノフ指数が適当であることが理解できる。

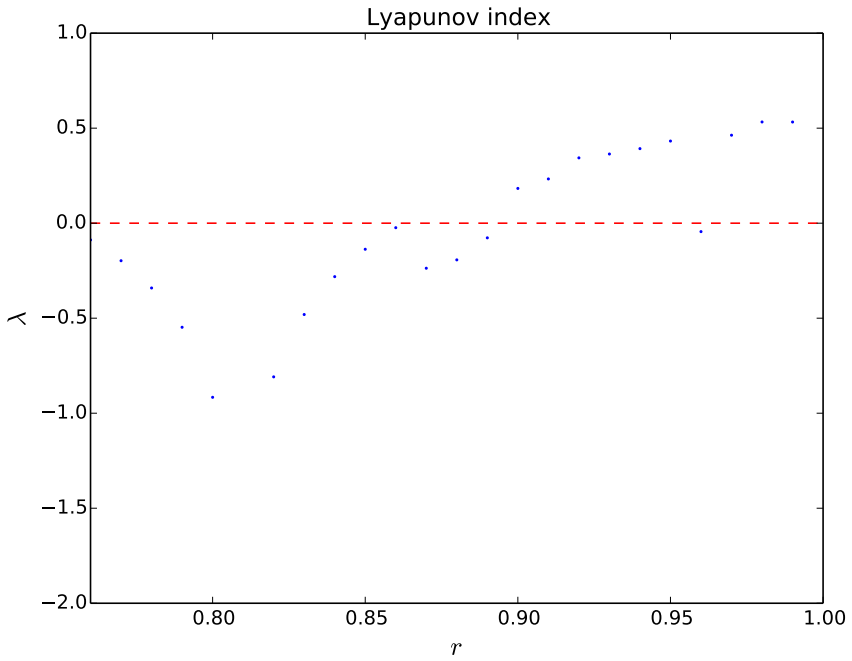

図 9　$\Delta r = 0.01$、ntransient=1000、n_calc=100000、$x_0 = 0.5$ としたときの $r - \lambda$ グラフ

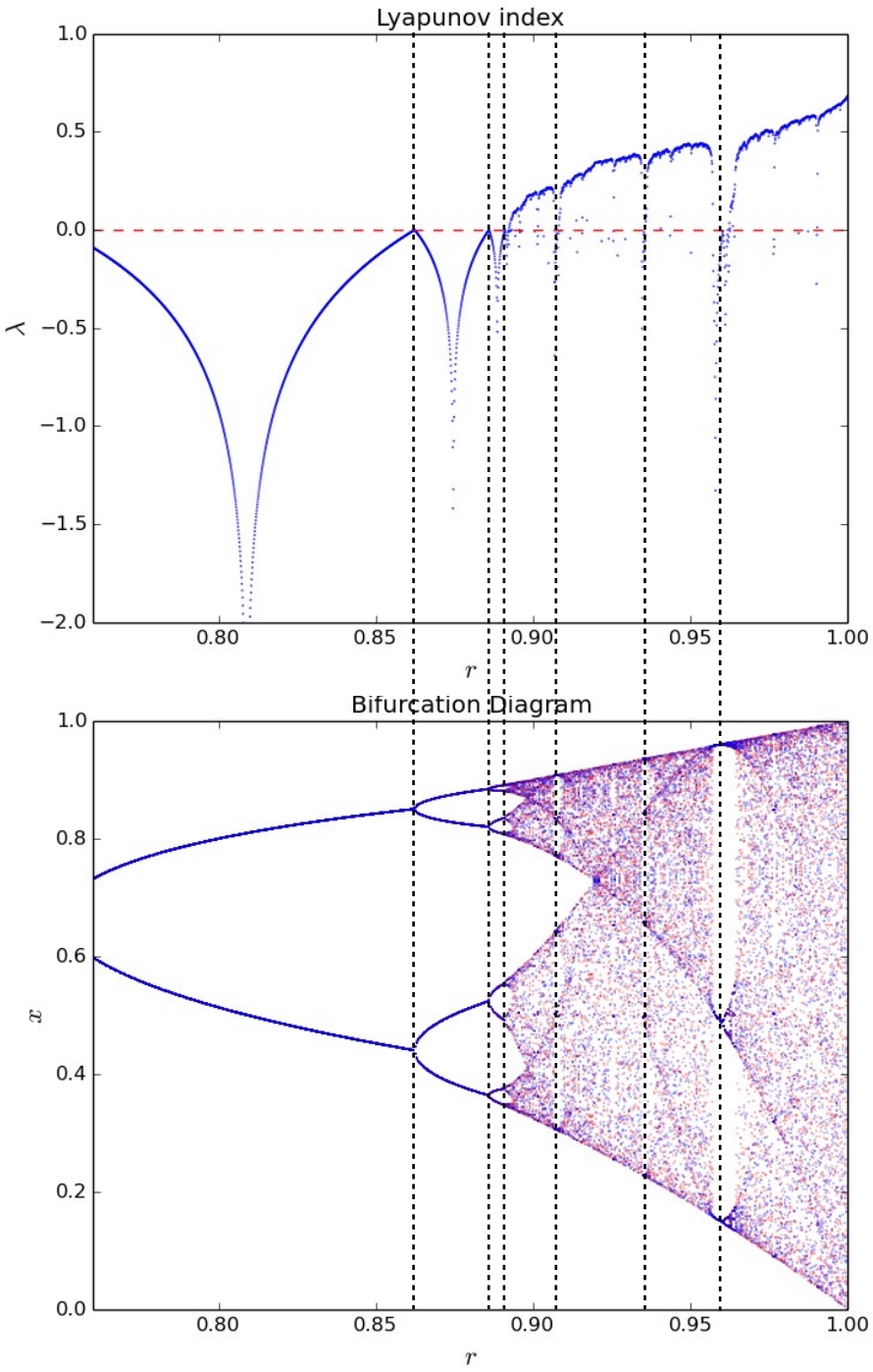

図 10　r の値に対するリアプノフ指数と系の分岐図の比較

4　まとめ

ロジスティック写像のリアプノフ指数を、素朴な方法で、またアルゴリズムを用いて計算することで、系のカオス的であることを表す指標としてのリアプノフ指数について理解を深めることができた。素朴な方法で求める際に、線形近似の適用範囲をどのようにして決めるかという問題に対して有効な方法を見いだせなかったが、そもそも精度よくリアプノフ指数を求めるには違ったアプローチをする必要があるということの理解の助けともなった。問題に対して、素朴な方法で解決するのが難しいと思われる場合、解析的に変形ができないか考えることの重要性を再認識できた。

5　参考文献

- ハーベイ・ゴールド, ジャン・トボチニク, 石川正勝・宮島佐介訳『計算物理学入門』, ピアソン・エデュケーション, 2000.