

# Logistic map の数値計算結果

幸谷智紀

1999 年 12 月 18 日 (土)

## 1 初めに

この文書は, logistic map

$$f(x) = 4x(1-x) \quad (1)$$

を使った, 漸化式

$$x_{i+1} := f(x_i) \quad (2)$$

によって導出される実数列  $\{x_i\}_{i=1}^{100}$  を, C プログラムと多倍長計算機能を備えた数式処理システム MuPAD を使って計算したプログラムと結果をまとめたものである。初期値は全て  $x_0 := 0.7501$  と指定した。

この数列の計算には多倍長計算が必要である。何故なら,  $0 < x_i < 1$  であれば,  $1 - x_i$  のところで, 必ず少しずつではあるが桁落ちが起こり, この関数の性質から再び桁落ちする領域に  $x_{i+1}$  が戻ってくるからである。従って, 断続的に桁落ちした結果,  $i$  が大きくなるにつれて, 精度が悪化していくことになる。それを防ぐには多倍長計算を行うしかない。

## 2 Logistic Map の計算 (1) — C プログラム

以下の C プログラムを, DELL DIMENSION XPS H266 (Pentium II 266MHz, 128MB RAM, Windows NT Workstation 4.0 SP5) 上で, gcc 2.95 19990728 を使ってコンパイルし, 実行した。

```
#include <stdio.h>

main()
{
    int i;
    float x[101]; /* 倍精度の時には double x[101] とする */

    x[0] = 0.7501;
    for(i = 0; i <= 100; i)
    {
        x[i+1] = 4 * x[i] * (1 - x[i]);
        if(i%10 == 0) printf("%5d %25.17e\n", i, x[i]);
    }
}
```

以下は IEEE754 単精度で配列を宣言したときの結果である。

```
0 7.50100016593933105e-01
10 8.44516813755035400e-01
20 1.22903920710086823e-01
30 4.97778177261352539e-01
40 5.81643998622894287e-01
50 5.58012068271636963e-01
60 2.28748228400945663e-02
70 9.98548150062561035e-01
80 9.42180097103118896e-01
90 2.12938979268074036e-01
100 7.56864011287689209e-01
```

以下は IEEE754 倍精度で配列を宣言したときの結果である。

```
0 7.50099999999999989e-01
10 8.44495953602201199e-01
20 1.42939724528399537e-01
30 8.54296020314155413e-01
40 7.74995884542777125e-01
50 7.95132827423636751e-02
60 2.73872762849587226e-01
70 9.97021556611396687e-01
80 3.52785425069070790e-01
90 6.35558111134618908e-01
100 5.15390006286616020e-01
```

両者見比べると、 $x_{20}$  ではもう殆ど精度がないことがわかる。それ以降の計算はどうも怪しい、と勘ぐらねばならない。

### 3 Logistic Map の計算 (2) — MuPAD

前節と同じハードウェアと OS を使い、ドイツで開発されている数式処理ソフトウェア MuPAD 1.4.4 Pro を使って、以下のスクリプトを実行した。MuPAD で計算桁数を指定するときには DIGITS というグローバル変数に 10 進数での桁数を指定すればよい。

```
DIGITS := 100:
x[0] := 0.7501:
for i from 0 to 100 step 1 do
  x[i + 1] := 4 * x[i] * (1 - x[i]):
end_for:
for i from 0 to 100 step 10 do
  print(i, x[i]):
end_for;
```

以下は DIGITS := 20(10 進 20 桁計算) としたときの結果である。

```
0, 0.7501
10, 0.84449595360221744753
20, 0.14293972451230765528
30, 0.85429600370442189166
40, 0.77497575311820123972
50, 0.093375332197704141841
```

60, 0.40822016829280303573  
70, 0.071511998671680367325  
80, 0.46325125515737645908  
90, 0.0011853049045088519022  
100, 0.41894573012901575815

これと先ほどの IEEE754 倍精度の計算結果と見比べると、どうも  $x_{40}$  の頭 4 桁ほどは一致しているが、それ以下の桁は丸め誤差が桁落ちにより上がってしまっている、ということが分かる。 $x_{50}$  より以降は全く信頼できない計算結果である。

以下は DIGITS := 50(10 進 50 桁計算) としたときの結果である。

0, 0.7501  
10, 0.84449595360221744753714870256154137267401952291229  
20, 0.14293972451230765528428175723130628307376969200214  
30, 0.8542960037044218916661311842588874437479758900823  
40, 0.77497575311820124128022346126421168481511313675586  
50, 0.093375332197703029055189668015168234762823013462054  
60, 0.40822016829087813187102766181952686730359229911213  
70, 0.071511999705058574897099346417593218350311240577887  
80, 0.46325330290077571055993467458320747053550671068192  
90, 0.0013344050120868840464692012150021107621136608489291  
100, 0.078817989371509906806704770440086762650658767283966

ここにきて、ようやく  $x_{90}$  の一桁目ぐらいは信頼しても良さそうだ、ぐらいのところまで来た。しかし  $x_{100}$  の数値はまだ信用できない。

以下は DIGITS := 100(10 進 100 桁計算) としたときの結果である。紙面の都合上、末尾の 10 桁は切り捨てさせていただいた。

0, 0.7501  
10, 0.844495953602217447537148702561541372674019522912291280007848388118986290823419270975121157  
20, 0.142939724512307655284281757231306283073769692002141121387637872878254442652405447154653323  
30, 0.854296003704421891666131184258887443747975890082277146848716525830873157517999293072280730  
40, 0.774975753118201241280223461264211684815113136725464353115716739919423777262939033931884043  
50, 0.0933753321977030290551896680151682347628230351487453020066411681729685131997586455460349178  
60, 0.408220168290878131871027661819526867303629812881299069587191914174238088850476295365965331  
70, 0.0715119997050585748970993464175932183301720988330772170512558519903814407400108286900737962  
80, 0.463253302900775710559934674583207430627755245547675072149154489741355095166291517037937750  
90, 0.00133440501208688404646920121499911906832767847107910033479094701619779865654038757784875352  
100, 0.0788179893715099068067047704626992647240094450346038441154434366642728380004689090274960709

ここで、ようやく  $x_{100} = 0.0788179893715099068067047704\dots$  であろうと言うことが出来る。

## 4 最後に

解析的に真の値を求めることの出来ないものを数値計算で求める際に留意すべき事は、伊理の本 [2] の「第 4 章 たった 1 回だけの計算なんて…」に詳しく例題付きで述べられている。ここでやっている手法はそれを適用しただけである。

## 参考文献

- [1] 合原・相澤 編, 臨時別冊・数理科学 カオス研究の最前線, サイエンス社, 1999.
- [2] 伊理・藤野, 数値計算の常識, 共立出版, 1985.