情報工学実験 I

課題5 カオスとフラクタル

– 第 1 週目 –

担当：池口 徹

TA：菅野嘉隆

tohru@ics.saitama-u.ac.jp

埼玉大学 工学部 情報システム工学科

本実験の内容

- 非線形システムにおいて観測されるカオスとフラクタルについて，MATLAB を用いた数値実験により体験する．
- 解析的に解くことが出来ない非線形システムを対象とする場合，数値シミュレーションの有効性を知る．

線形な差分方程式

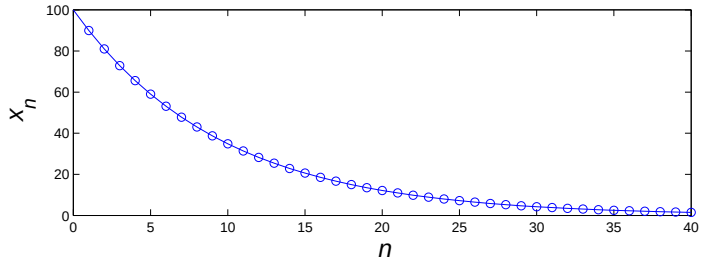
n 番目の状態値 x_n と $n+1$ 番目の状態値 x_{n+1} の間の関係が，線形 (linear) な関係を有する

$$x_{n+1} = ax_n \tag{1}$$

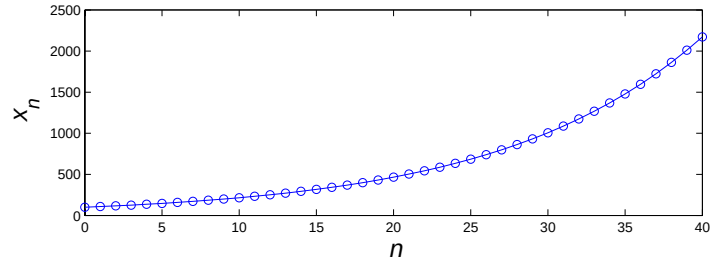
↓

1 次元 (あるいは自由度 1) の
線形な差分方程式 (漸化式，数列，写像)

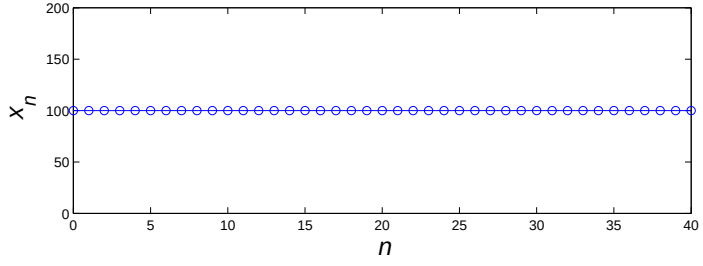
線形な差分方程式の振る舞い



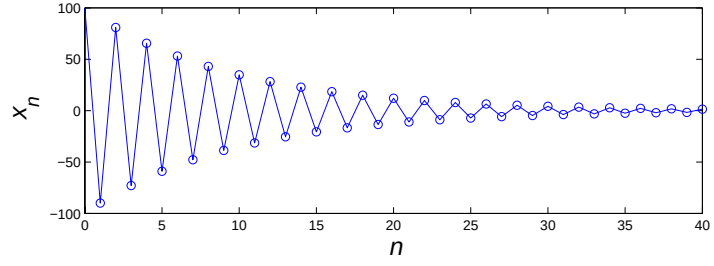
(a)$a = 0.9$



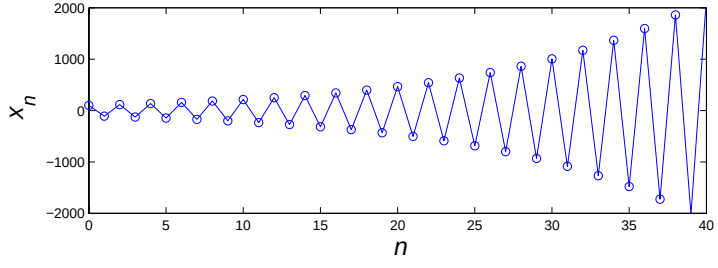
(b) $a = 1.08$



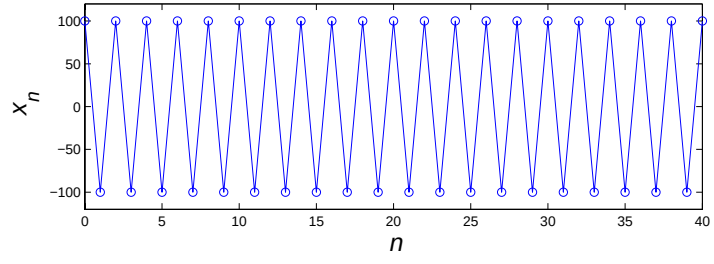
(c)$a = 1.00$



(d) $a = -0.90$



(e)$a = -1.08$



(f)$a = -1.00$

非線形な差分方程式

非線形な差分方程式

非線形 = 線形でない

非線形な差分方程式

非線形 = 線形でない

$$x_{n+1} = a x_n (1 - x_n) \tag{2}$$

↓

ロジスティック写像

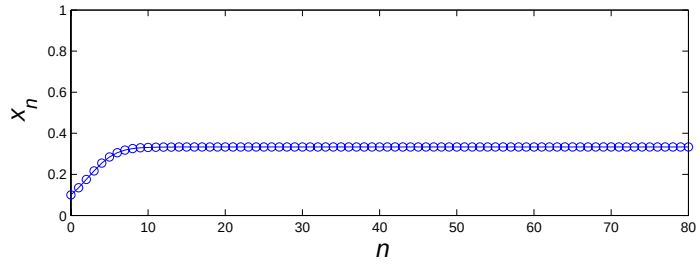
↓

解を解析的に求めることはできない

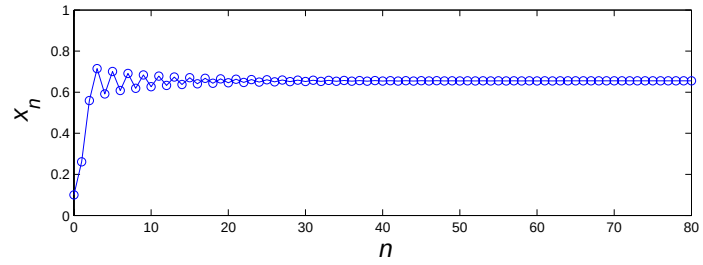
↓

コンピュータを用いた数値実験で解を求めることが必要

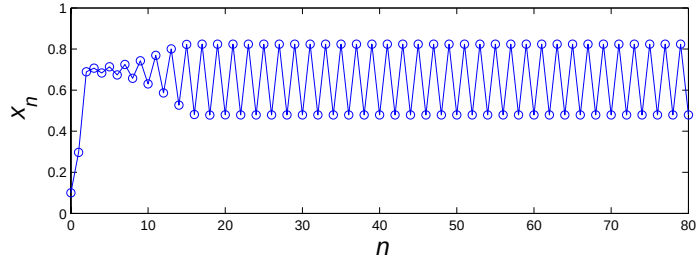
ロジスティック写像の振る舞い例



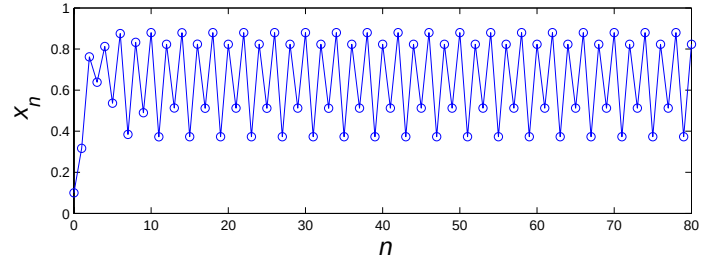
(a)$a = 1.5$



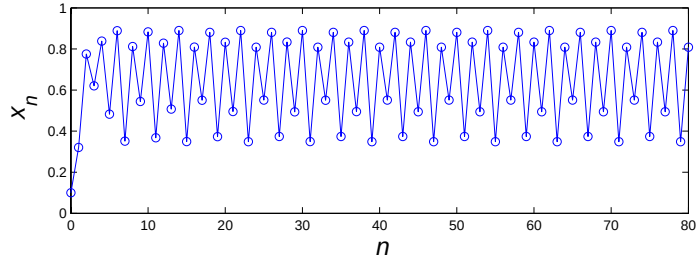
(b)$a = 2.98$



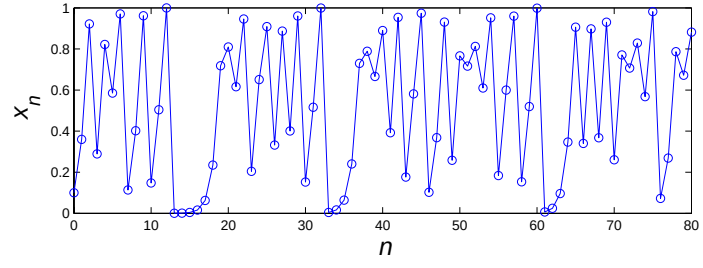
(c)$a = 3.3$



(d)$a = 3.52$



(e)$a = 3.56$



(f)$a = 4.0$

ポイント

$$x_{n+1} = a x_n (1 - x_n)$$

- $n = 0$ での値 x_0(初期値) から x_1 を決定できる
  確率的な要素は全く含まれていない
- $n = 1$ での値 x_1 から x_2 を決定できる
- この過程は，次々と繰り返すことができる

初期値が与えられると，未来永劫，
全てが決定される

↓

「決定論に従う」

カオスとは

- 少数自由度の決定論的非線形システムにおいて生じる複雑な現象
  ロジスティック写像では $a = 4$ のときに相当

カオスとは

- 少数自由度の決定論的非線形システムにおいて生じる複雑な現象
  ロジスティック写像では $a = 4$ のときに相当
  1. 少数自由度
     ロジスティック写像の自由度は 1

$$x_{n+1} = 4x_n(1 - x_n)$$

カオスとは

- 少数自由度の決定論的非線形システムにおいて生じる複雑な現象
  ロジスティック写像では $a = 4$ のときに相当
  1. 少数自由度
     ロジスティック写像の自由度は 1
  2. 決定論的
     ロジスティック写像では現状態 x_n が決まれば，次状態 x_{n+1} は完全に決定される

$$x_{n+1} = 4x_n(1 - x_n)$$

カオスとは

- 少数自由度の決定論的非線形システムにおいて生じる複雑な現象
  ロジスティック写像では $a = 4$ のときに相当
  1. 少数自由度
     ロジスティック写像の自由度は 1
  2. 決定論的
     ロジスティック写像では現状態 x_n が決まれば，次状態 x_{n+1} は完全に決定される
  3. 非線形性
     ロジスティック写像には 2 次の非線形性

$$x_{n+1} = 4x_n(1 - x_n)$$

カオスがなぜ重要か?

カオスがなぜ重要か?

- ロジスティック写像 ($a = 4$ の場合) のような決定論的なシステムから生じる複雑な現象

カオスがなぜ重要か?

- ロジスティック写像 ($a = 4$ の場合) のような決定論的なシステムから生じる複雑な現象

決定論的なルールを持つシステム＋非線形性

||

カオスがなぜ重要か?

- ロジスティック写像 ($a = 4$ の場合) のような決定論的なシステムから生じる複雑な現象

決定論的なルールを持つシステム＋非線形性

||

サイコロ投げ，コイントスを再現できる

カオスがなぜ重要か?

- ロジスティック写像 ($a = 4$ の場合) のような決定論的なシステムから生じる複雑な現象

決定論的なルールを持つシステム＋非線形性

$\Vert$

サイコロ投げ，コイントスを再現できる

$\downarrow$

決定論的カオス

カオスがなぜ重要か?

- ロジスティック写像 ($a = 4$ の場合) のような決定論的なシステムから生じる複雑な現象

決定論的なルールを持つシステム＋非線形性

$\|$

サイコロ投げ，コイントスを再現できる

$\downarrow$

決定論的カオス

確率論 $\Longleftrightarrow$ **決定論**

カオスがなぜ重要か?

- ロジスティック写像 ($a = 4$ の場合) のような決定論的なシステムから生じる複雑な現象

決定論的なルールを持つシステム＋非線形性

$\|$

サイコロ投げ，コイントスを再現できる

$\downarrow$

決定論的カオス

確率論 $\Longleftrightarrow$ **決定論**

なぜカオスが重要か?

- この世には，線形な現象など存在しない.

なぜカオスが重要か?

- この世には，線形な現象など存在しない.
- 実際，工学では複雑な現象が多数存在

なぜカオスが重要か?

- この世には，線形な現象など存在しない.
- 実際，工学では複雑な現象が多数存在
- つまり，工学は非線形現象の宝庫である
  (カオス，フラクタルに出会うことが多い!)

なぜカオスが重要か?

- この世には，線形な現象など存在しない.
- 実際，工学では複雑な現象が多数存在
- つまり，工学は非線形現象の宝庫である
  (カオス，フラクタルに出会うことが多い!)
- 非線形性を考えることが重要.

なぜカオスが重要か?

- この世には，線形な現象など存在しない.
- 実際，工学では複雑な現象が多数存在
- つまり，工学は非線形現象の宝庫である
  (カオス，フラクタルに出会うことが多い!)
- 非線形性を考えることが重要.
- 複雑な現象の本質は?決定論?確率論?
  これを知ることも重要

なぜカオスが重要か?

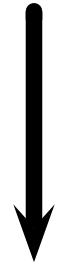
- この世には，線形な現象など存在しない.
- 実際，工学では複雑な現象が多数存在
- つまり，工学は非線形現象の宝庫である
  (カオス，フラクタルに出会うことが多い!)
- 非線形性を考えることが重要.
- 複雑な現象の本質は?決定論?確率論?
  これを知ることも重要
- 本質を知れば，予測精度，制御効率向上等々,
  工学的に貢献が出来る

なぜカオスが重要か?

- この世には，線形な現象など存在しない.
- 実際，工学では複雑な現象が多数存在
- つまり，工学は非線形現象の宝庫である
  (カオス，フラクタルに出会うことが多い!)
- 非線形性を考えることが重要.
- 複雑な現象の本質は?決定論?確率論?
  これを知ることも重要
- 本質を知れば，予測精度，制御効率向上等々，
  工学的に貢献が出来る
- つまり，カオス，フラクタルを勉強しておけば，必ず役に立つ!

但し...

但し，線形な場合とは異なり，コンピュータを用いた数値実験が特に重要な方法論となる



第１週目の実験内容は?

実験の内容 (第1週目)

- ロジスティック写像の実験
  1. ロジスティック写像がどのような振る舞いを示すか
  2. MATLAB を用いて，数値計算で確かめる

MATLAB 入門

1. ログインする.
2. 各自のホームディレクトリに，`matlab` というディレクトリを作る
   % mkdir matlab [RET]
   % cd matlab [RET]
   以降このディレクトリにおいて作業する.
3. MATLAB を起動する.
   % /usr/local/matlab5/bin/matlab [RET]
4. MATLAB のプロンプトが現れる.
   >>
5. >>の後にコマンド入力し作業を行なう.
6. >>`quit` で終了.

変数の定義と基本演算

1. ベクトル，配列の定義，代入
2. 命令の最後に;(セミコロン) をつけると
   画面に表示されない
3. MATLAB での配列の添字は 1 から始まる.
   a(1) は OK，a(0) は NG!
4. 行列の一部分にアクセス可
5. B(行，列) と考える
6. 行列，ベクトルの演算も，通常用いる書式と同じように
   用いることができる.

関数 m–ファイルを使おう!

MATLAB に命令を伝える方法

1. プロンプトに直接コマンドを入力する
2. コマンドを自分用に作成し，ファイルとして保存する

m–ファイル ··· `foo.m`

3. mule 等のエディタで，m–ファイルを作成しセーブする.
4. MATLAB プロンプトにおいて
   >> foo [RET]
   で実行

rotation.m

```
function R=rotation(theta)
% ROTATION returns Rotation Matrix of angle theta
% Usage: R=rotation(theta)

% 2x2 matrix R is defined.
R = [cos(theta) -sin(theta);...
     sin(theta)  cos(theta)];
```

キーワード function：このファイルが関数であることを示す.

入力：theta, 出力：R
使い方：

```
>> rotation(pi/3)
>> R=rotation(pi/4)

>> help rotation
```

logistic.m

```
function x=logistic(R,xinit,N)
% LOGISTIC caclulates a solution of the logistic map,
% with parameter R, an initial condition xinit and
% the number of iterations N.
% Usage: x=logistic(R,xinit,N)

% Default values
if nargin<1,R=3.999999;end
if nargin<2,xinit=0.2; end
if nargin<3,N=1000; end

% Allocate an Nx1 vector as an output x.
x=zeros(N,1);

% Calculate x_n for n=1,2,...,N.
x(1)=xinit;
for n=1:N-1
        x(n+1)=R*x(n)*(1-x(n));
end
```

入力は　R, xinit, N

出力は　x

デフォルト値の設定

N× 1 の配列を確保し，0 を代入する

x(1) に xinit(初期値) を代入する

写像の繰り返しを for 文で実現

logistic.m

```
function x=logistic(R,xinit,N)
% LOGISTIC caclulates a solution of the logistic map,
% with parameter R, an initial condition xinit and
% the number of iterations N.
% Usage: x=logistic(R,xinit,N)

% Default values
if nargin<1,R=3.999999;end
if nargin<2,xinit=0.2; end
if nargin<3,N=1000; end

% Allocate an Nx1 vector as an output x.
x=zeros(N,1);

% Calculate x_n for n=1,2,...,N.
x(1)=xinit;
for n=1:N-1
        x(n+1)=R*x(n)*(1-x(n));
end
```

入力は　R, xinit, N

出力は　x

デフォルト値の設定

N× 1 の配列を確保し，0 を代入する

x(1) に xinit(初期値) を代入する

写像の繰り返しを for 文で実現

logistic.m

```
function x=logistic(R,xinit,N)
% LOGISTIC caclulates a solution of the logistic map,
% with parameter R, an initial condition xinit and
% the number of iterations N.
% Usage: x=logistic(R,xinit,N)

% Default values
if nargin<1,R=3.999999;end
if nargin<2,xinit=0.2; end
if nargin<3,N=1000; end

% Allocate an Nx1 vector as an output x.
x=zeros(N,1);

% Calculate x_n for n=1,2,...,N.
x(1)=xinit;
for n=1:N-1
        x(n+1)=R*x(n)*(1-x(n));
end
```

入力は　R, xinit, N

出力は　x

デフォルト値の設定

N× 1 の配列を確保し，0 を代入する

x(1) に xinit(初期値) を代入する

写像の繰り返しを for 文で実現

logistic.m

```
function x=logistic(R,xinit,N)
% LOGISTIC caclulates a solution of the logistic map,
% with parameter R, an initial condition xinit and
% the number of iterations N.
% Usage: x=logistic(R,xinit,N)

% Default values
if nargin<1,R=3.999999;end
if nargin<2,xinit=0.2; end
if nargin<3,N=1000; end

% Allocate an Nx1 vector as an output x.
x=zeros(N,1);

% Calculate x_n for n=1,2,...,N.
x(1)=xinit;
for n=1:N-1
        x(n+1)=R*x(n)*(1-x(n));
end
```

入力は　R, xinit, N

出力は　x

デフォルト値の設定

N× 1 の配列を確保し，0 を代入する

x(1) に xinit(初期値) を代入する

写像の繰り返しを for 文で実現

logistic.m

```
function x=logistic(R,xinit,N)
% LOGISTIC caclulates a solution of the logistic map,
% with parameter R, an initial condition xinit and
% the number of iterations N.
% Usage: x=logistic(R,xinit,N)

% Default values
if nargin<1,R=3.999999;end
if nargin<2,xinit=0.2; end
if nargin<3,N=1000; end

% Allocate an Nx1 vector as an output x.
x=zeros(N,1);

% Calculate x_n for n=1,2,...,N.
x(1)=xinit;
for n=1:N-1
        x(n+1)=R*x(n)*(1-x(n));
end
```

入力は　R, xinit, N

出力は　x

デフォルト値の設定

N× 1 の配列を確保し，0 を代入する

x(1) に xinit(初期値) を代入する

写像の繰り返しを for 文で実現

logistic.m

```
function x=logistic(R,xinit,N)
% LOGISTIC caclulates a solution of the logistic map,
% with parameter R, an initial condition xinit and
% the number of iterations N.
% Usage: x=logistic(R,xinit,N)

% Default values
if nargin<1,R=3.999999;end
if nargin<2,xinit=0.2; end
if nargin<3,N=1000; end

% Allocate an Nx1 vector as an output x.
x=zeros(N,1);

% Calculate x_n for n=1,2,...,N.
x(1)=xinit;
for n=1:N-1
        x(n+1)=R*x(n)*(1-x(n));
end
```

入力は　R, xinit, N

出力は　x

デフォルト値の設定

N× 1 の配列を確保し，0 を代入する

x(1) に xinit(初期値) を代入する

写像の繰り返しを for 文で実現

logistic.m

```
function x=logistic(R,xinit,N)
% LOGISTIC caclulates a solution of the logistic map,
% with parameter R, an initial condition xinit and
% the number of iterations N.
% Usage: x=logistic(R,xinit,N)

% Default values
if nargin<1,R=3.999999;end
if nargin<2,xinit=0.2; end
if nargin<3,N=1000; end

% Allocate an Nx1 vector as an output x.
x=zeros(N,1);

% Calculate x_n for n=1,2,...,N.
x(1)=xinit;
for n=1:N-1
        x(n+1)=R*x(n)*(1-x(n));
end
```

入力は　R, xinit, N

出力は　x

デフォルト値の設定

N× 1 の配列を確保し，0 を代入する

x(1) に xinit(初期値) を代入する

写像の繰り返しを for 文で実現

実行

```
>> xc=logistic;
```

実行

```
>> xc=logistic;
>> size(xc);
```

実行

```
>> xc=logistic;
>> size(xc);
>> xc2=logistic(3.2,0.3,2000);
```

実行，結果の表示 – 基本 –

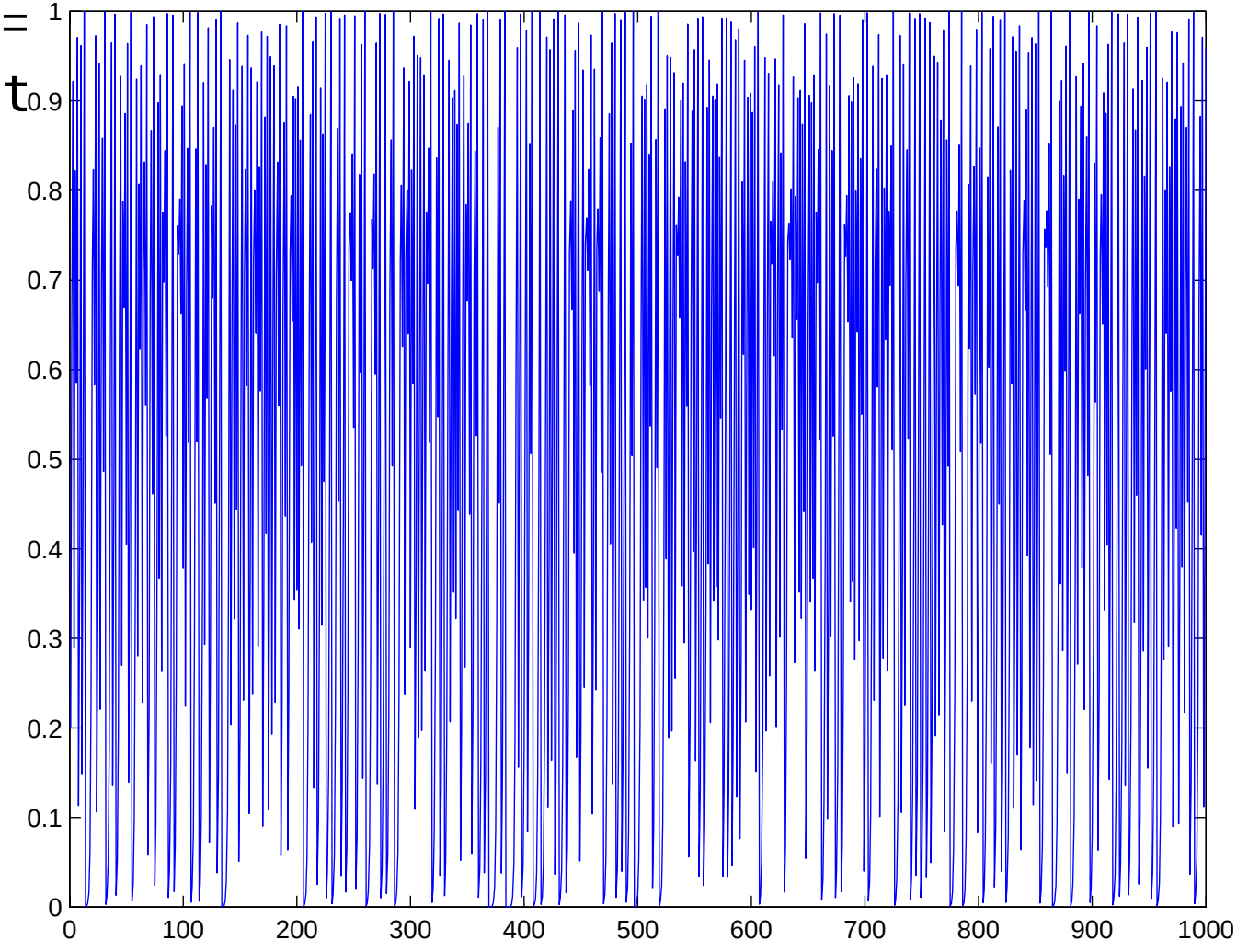
```
>> xc=logistic;
>> size(xc);
>> xc2=logistic(3.2,0.3,2000);
```

実行，結果の表示 – 基本 –

```
>> xc=logistic;
>> size(xc);
>> xc2=logistic(3.2,0.3,2000);
>> plot(xc)
```

実行，結果の表示 – 基本 –

```
>> xc=logistic;
>> size(xc);
>> xc2=
>> plot
```

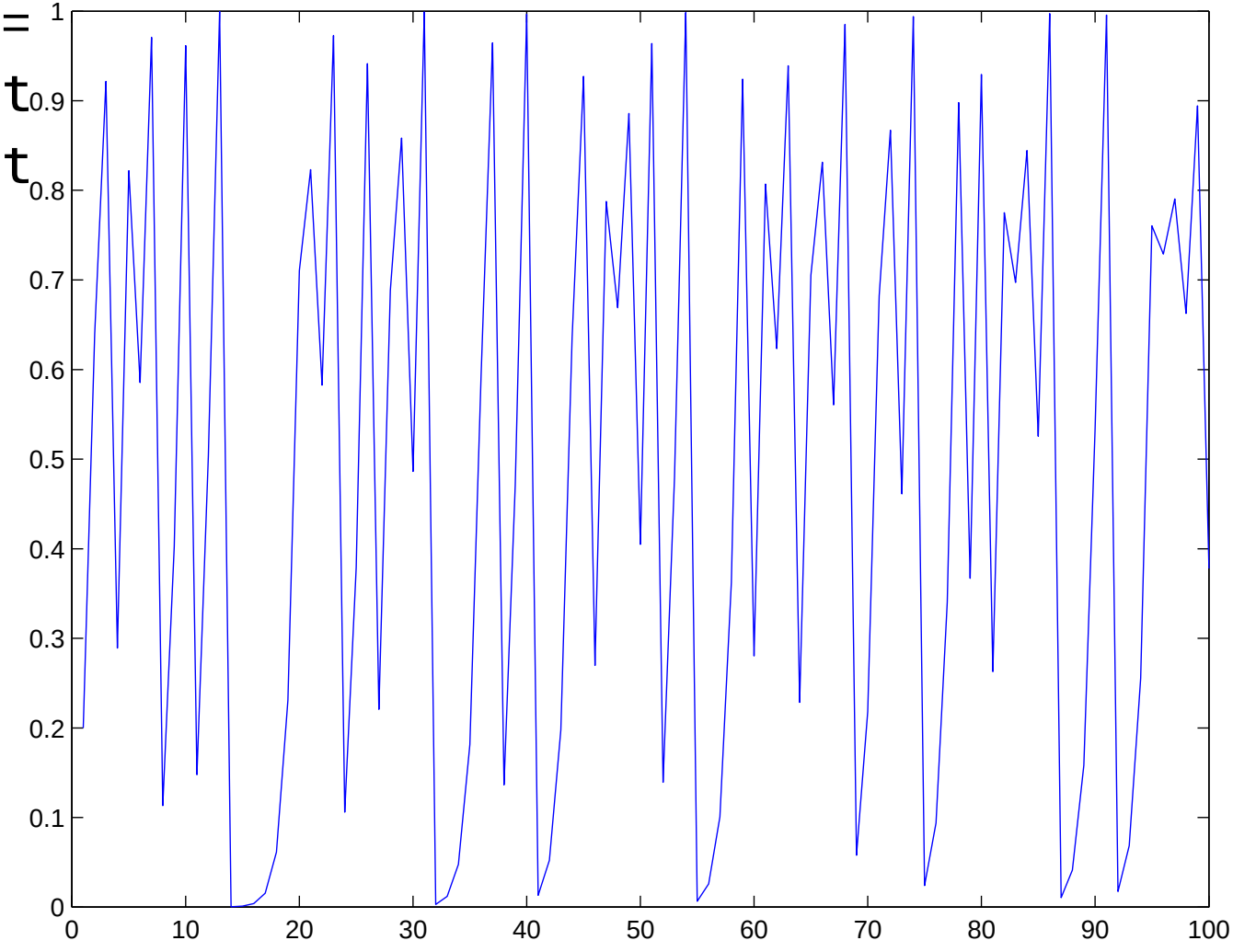


実行，結果の表示 – 基本 –

```
>> xc=logistic;
>> size(xc);
>> xc2=logistic(3.2,0.3,2000);
>> plot(xc)
>> plot(xc(1:100))
```

実行，結果の表示 – 基本 –

```
>> xc=logistic;
>> size(xc);
>> xc2=
>> plot
>> plot
```

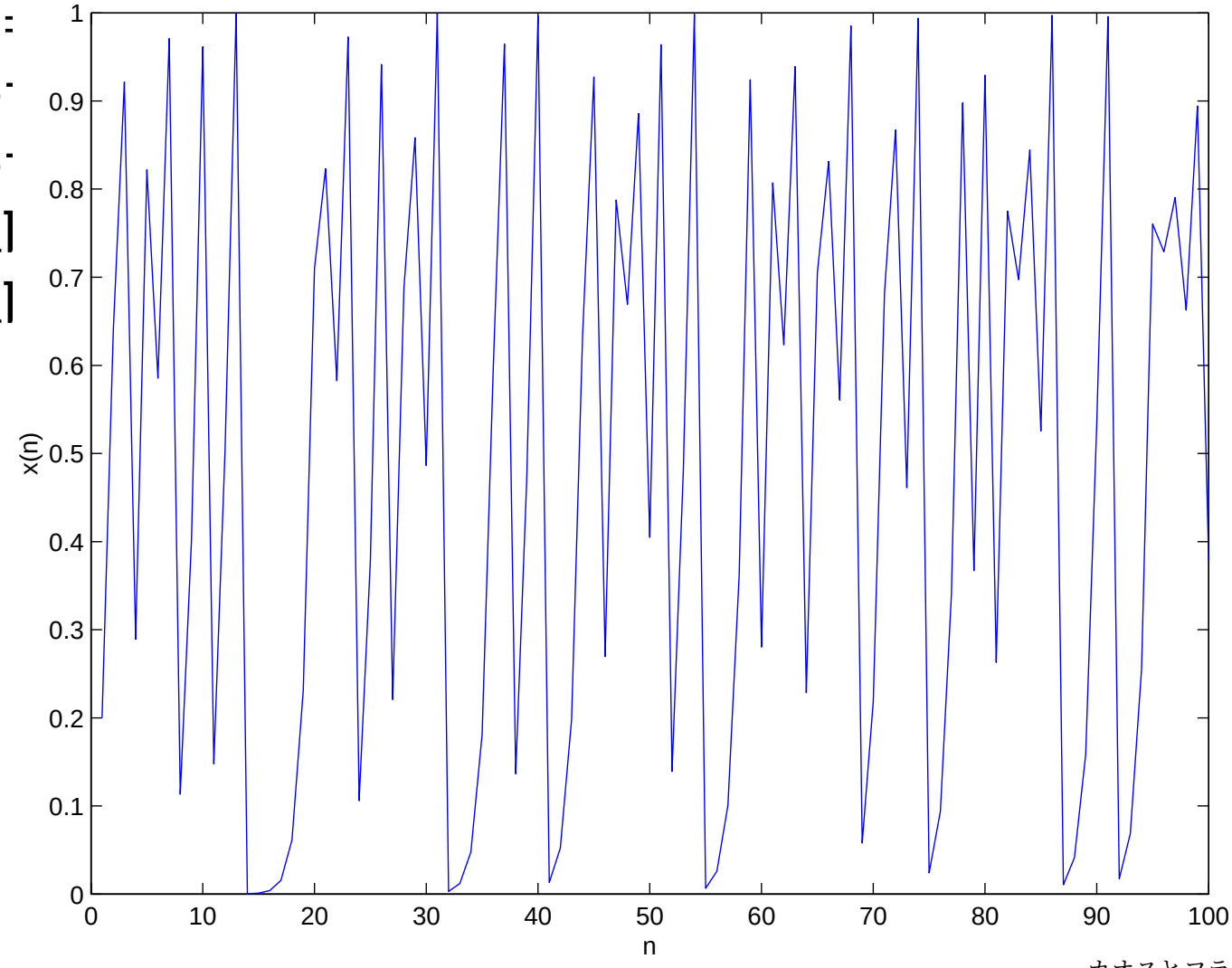


実行，結果の表示 – 基本 –

```
>> xc=logistic;
>> size(xc);
>> xc2=logistic(3.2,0.3,2000);
>> plot(xc)
>> plot(xc(1:100))
>> xlabel('n')
>> ylabel('x(n)')
```

実行，結果の表示 – 基本 –

```
>> xc=logistic;
>> size(xc);
>> xc2=
>> plo
>> plo
>> xlal
>> ylal
```



実行，結果の表示 – 応用–

実行，結果の表示 – 応用–

```
>> clf
```

実行，結果の表示 – 応用–

```
>> clf
>> hold on
```

実行，結果の表示 – 応用–

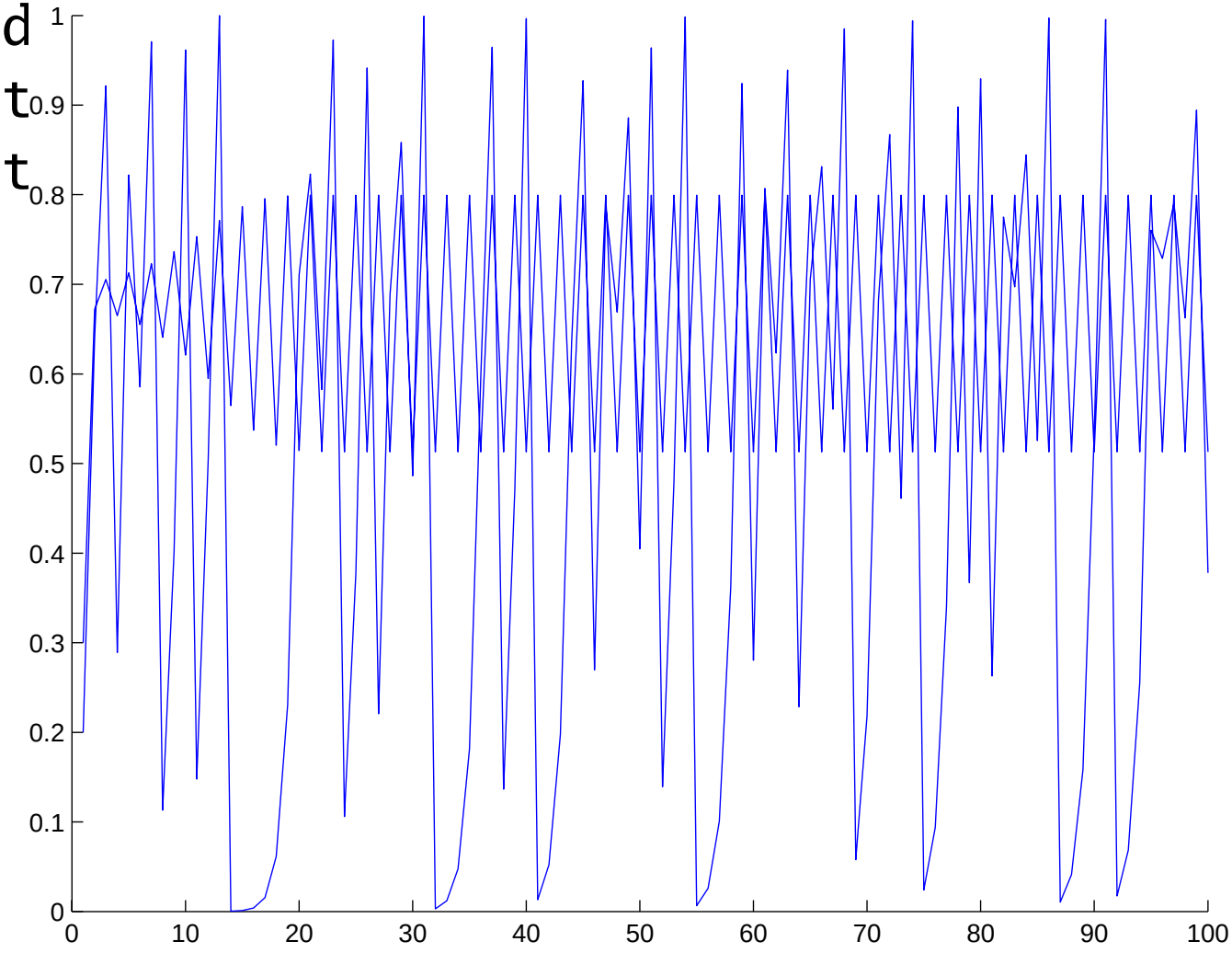
```
>> clf
>> hold on
>> plot(xc(1:100))
```

実行，結果の表示 – 応用–

```
>> clf
>> hold on
>> plot(xc(1:100))
>> plot(xc2(1:100))
```

実行，結果の表示 – 応用–

```
>> clf
>> hold
>> plot
>> plot
```



実行，結果の表示 – 応用–

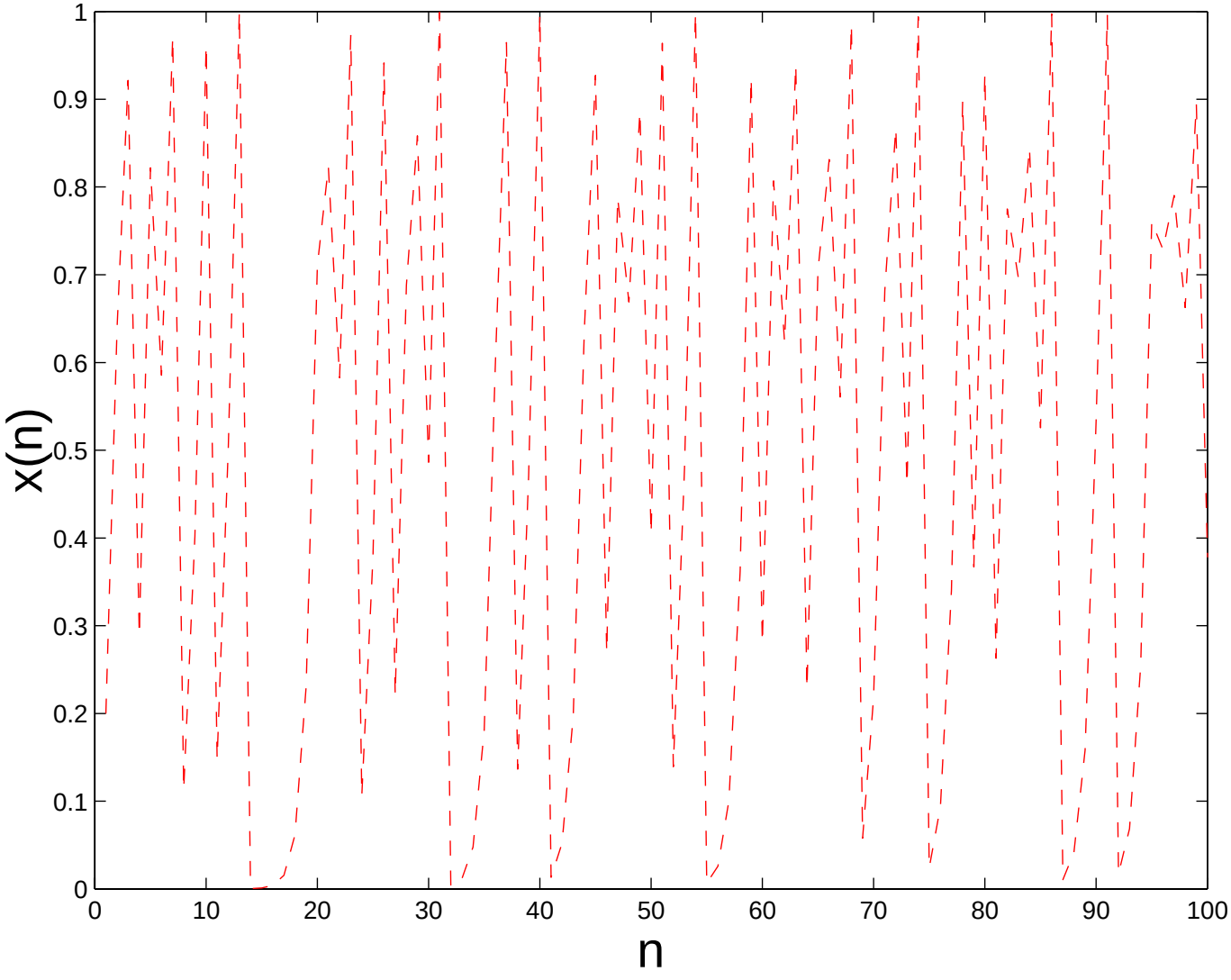
```
>> clf
>> hold on
>> plot(xc(1:100))
>> plot(xc2(1:100))
>> plot(xc(1:100),'r--')
```

実行，結果の表示 – 応用–

```
>> clf
>> hold on
>> plot(xc(1:100))
>> plot(xc2(1:100))
>> plot(xc(1:100),'r--')
>> xlabel('n','FontSize',20)
>> ylabel('x(n)','FontSize',20)
```

実行，結果の表示 – 応用–

```
>> clf
>> hol
>> plo
>> plo
>> plo
>> xla
>> yla
```

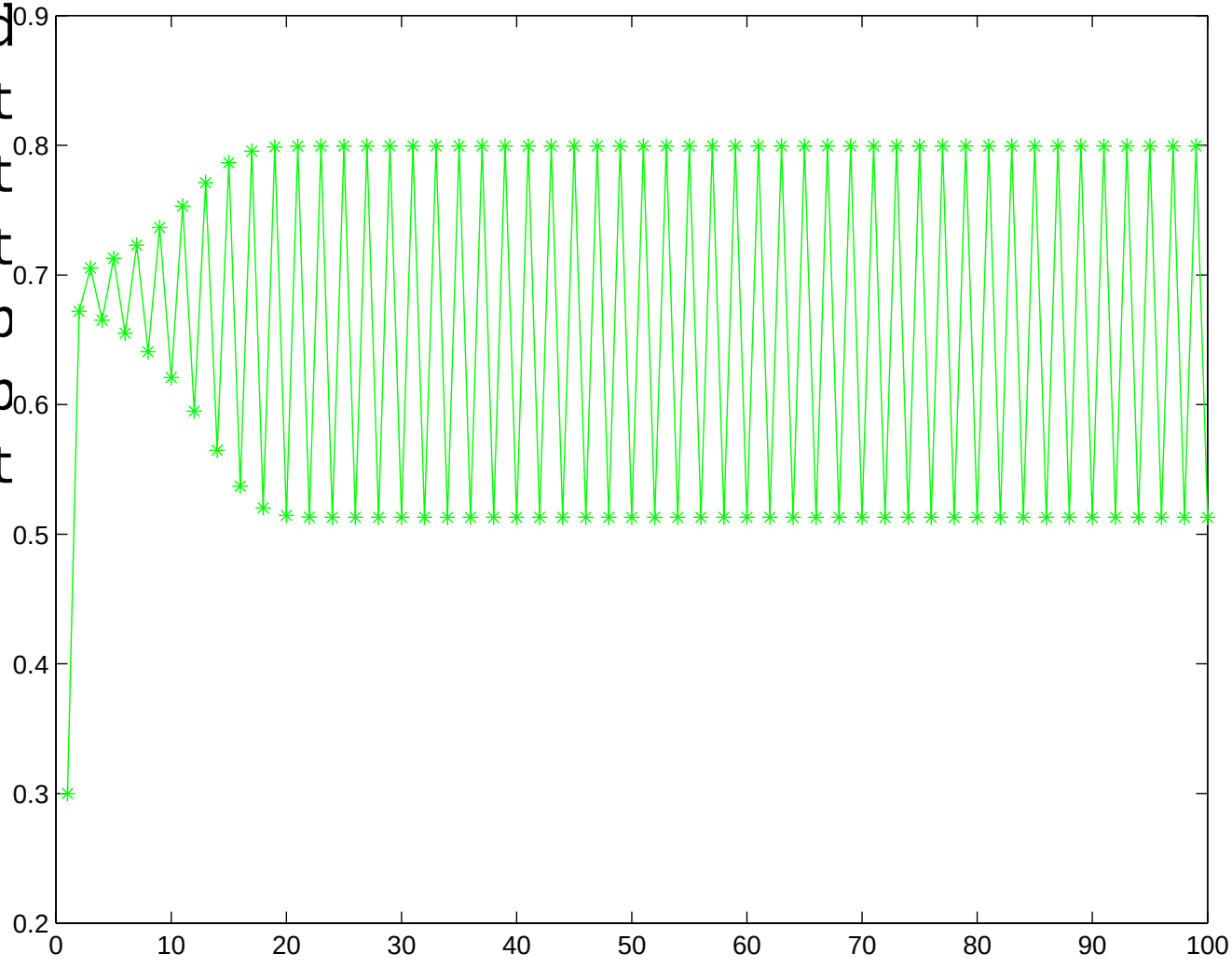


実行，結果の表示 – 応用–

```
>> clf
>> hold on
>> plot(xc(1:100))
>> plot(xc2(1:100))
>> plot(xc(1:100),'r--')
>> xlabel('n','FontSize',20)
>> ylabel('x(n)','FontSize',20)
>> plot(xc2(1:100),'g*-')
```

実行，結果の表示 – 応用–

```
>> clf
>> hold
>> plot
>> plot
>> plot
>> xlab
>> ylab
>> plot
```

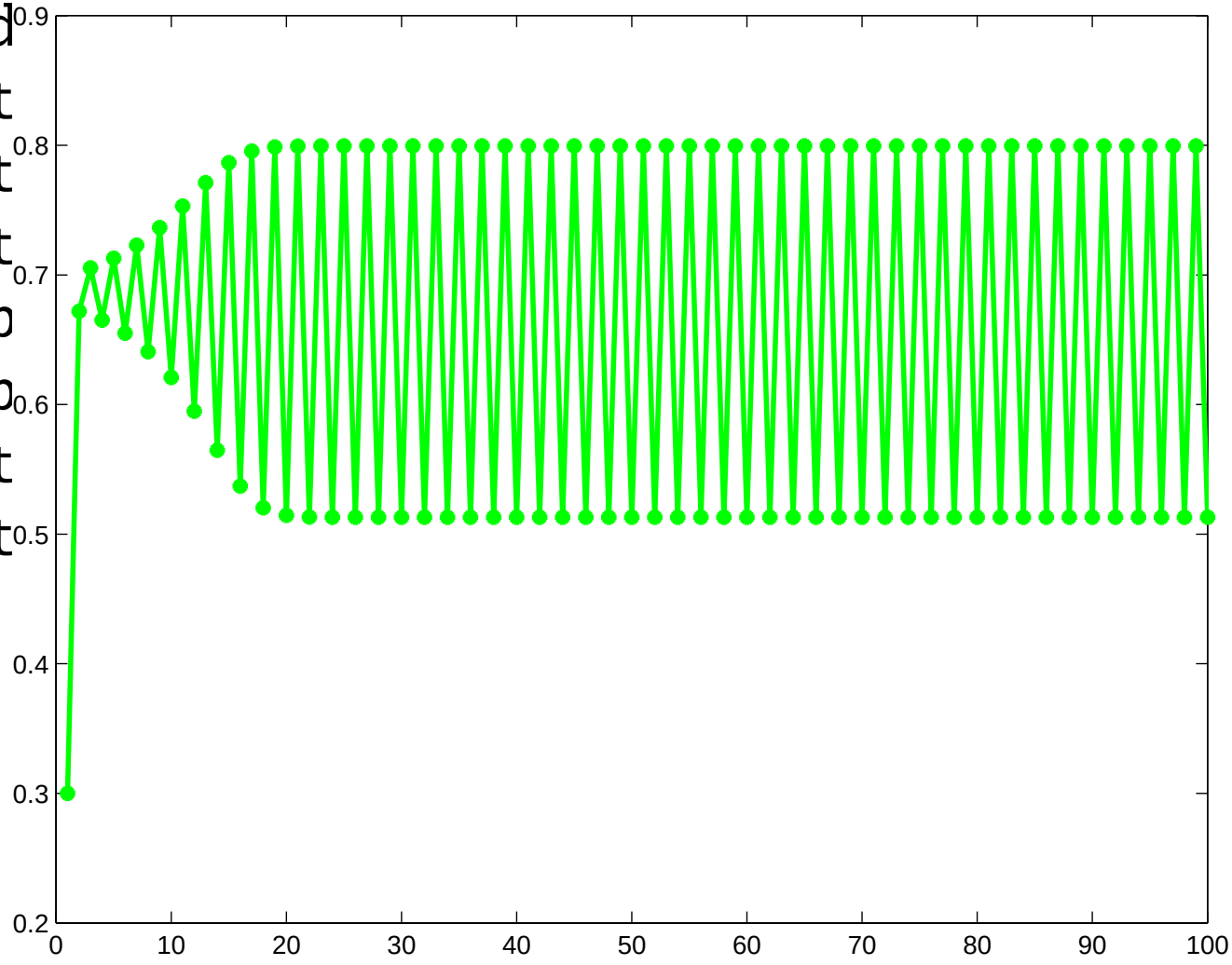


実行，結果の表示 – 応用–

```
>> clf
>> hold on
>> plot(xc(1:100))
>> plot(xc2(1:100))
>> plot(xc(1:100),'r--')
>> xlabel('n','FontSize',20)
>> ylabel('x(n)','FontSize',20)
>> plot(xc2(1:100),'g*-')
>> plot(xc2(1:100),'g*-','Linewidth',[2])
```

実行，結果の表示 – 応用–

```
>> clf
>> hold
>> plot
>> plot
>> plot
>> xlab
>> ylab
>> plot
>> plot
```



実行，結果の表示 – 応用– と印刷

```
>> clf
>> hold on
>> plot(xc(1:100))
>> plot(xc2(1:100))
>> plot(xc(1:100),'r--')
>> xlabel('n','FontSize',20)
>> ylabel('x(n)','FontSize',20)
>> plot(xc2(1:100),'g*-')
>> plot(xc2(1:100),'g*-','Linewidth',[2])
>> help plot
```

実行，結果の表示 – 応用– と印刷

```
>> clf
>> hold on
>> plot(xc(1:100))
>> plot(xc2(1:100))
>> plot(xc(1:100),'r--')
>> xlabel('n','FontSize',20)
>> ylabel('x(n)','FontSize',20)
>> plot(xc2(1:100),'g*-')
>> plot(xc2(1:100),'g*-','Linewidth',[2])
>> help plot
>> print -deps2 foo.eps
```

実行，結果の表示 – 応用– と印刷

```
>> clf
>> hold on
>> plot(xc(1:100))
>> plot(xc2(1:100))
>> plot(xc(1:100),'r--')
>> xlabel('n','FontSize',20)
>> ylabel('x(n)','FontSize',20)
>> plot(xc2(1:100),'g*-')
>> plot(xc2(1:100),'g*-','Linewidth',[2])
>> help plot
>> print -deps2 foo.eps
>> help print
```

実験と課題 (第1週目)

1. ロジスティック写像におけるカオス

   パラメータを様々な値にした場合について,
   $n = 0, 1, 2, \ldots$ としたときの解を横軸 n, 縦軸 x_n としてプロットせよ.

   (a) 関数`logistic` を用いると良い.
   (b) 「パラメータを様々な値」$\rightarrow 0 \leq a \leq 4$ で変えてみる.
   (c) 結果を`plot` を用いて表示する.

実験と課題 (第1週目)

2. 初期値に対する鋭敏な依存性

ロジスティック写像のカオス応答の場合 (例えば，$a = 4$ のとき) について，初期値に対する鋭敏な依存性が存在するかどうかを確かめよ.

(a) 関数`logistic` を用いると良い.

(b) まず，2 種類の初期値を与える.

(c) これらの差が，どのように変化するかを見る.

(d) 余力のある人は，カオス応答で無い場合についても計算し，違いを比べてみよう.