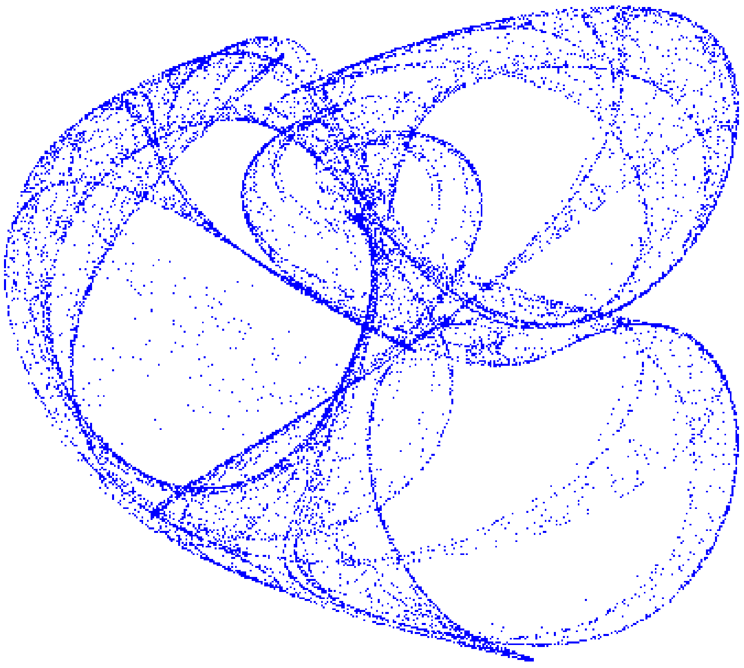
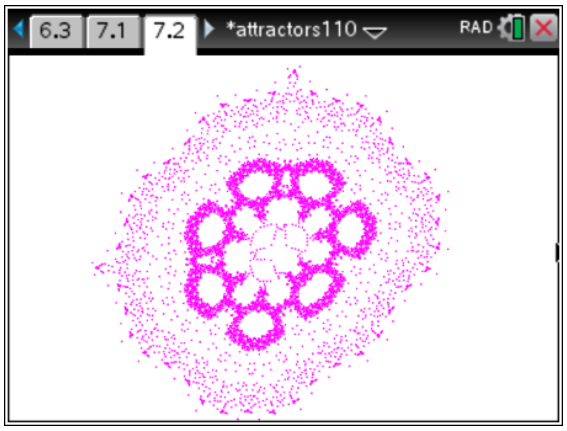
Attracted by (STRANGE) Attractors

An illustrated guided tour from well-known to unknown attractors





Your Tour Guide: Josef Böhm

(for DERIVE and TI-NspireCAS)

Austrian Center for Didactics of Computer Algebra

Contents

Attracted by (STRANGE) Attractors

An illustrated guided tour from well-known to unknown attractors

Your Tour Guide: Josef Böhm

Abstract

When investigating dynamic systems, fractals and chaotic behavior it is inevitable to come across "attractors". An attractor is a set towards which a dynamical system evolves over time. This can be one or more points, a curve, or a kind of fractal structure in the phase space. The latter are called "strange attractors". Investigating these special sets opens a huge box of surprising shapes which might form a linkage between mathematics and arts.

We will use the excellent mathematical capabilities of DERIVE and TI-NspireCAS (and other tools) wrapped in programs to present well known and not so well-known attractors which can be found in "CHAOS-Literature", as there are: Lorenz-, Hénon-, Rössler-, Ikeda-, Gumowski-Mira-attractors and others.

This is a revised and extended summary of my presentation given at TIME 2012 in Tartu, Estonia, together with some additional notes. In particular, I added the treatment by TI-NspireCAS, too – as far as possible and last - but not least – the chapters on the fractal dimension.

1 Introduction and well-known Basics

The *Strange Attractors* can be counted to recreational mathematics and to creational mathematics as well. This field of mathematics offers an affective (emotional) approach to interesting insights rather than a cognitive (rational) one. There is also a lot of pure and hard mathematics behind all these nice patterns which I will present in the following. I served more than 30 years as school teacher and was always glad offering "pretty" mathematical objects to catch the interest of the students which were not so happy with the hard facts of mathematics. One can really see and experience the BEAUTY of mathematics approaching the secrets of STRANGE ATTRACTORS.

An attractor is a set towards a dynamical system evolves over time. This can be

- one or more points
- a kind of fractal structure
- a curve

A key word for all what is following is **ITERATION**:

Iteration of one variable: $x_{n+1} = f(x_n); x_0 = \text{given}$

Iteration of two (or more variables (dynamic system)): $\begin{matrix} x_{n+1} = f_1(x_n, y_n) \\ y_{n+1} = f_2(x_n, y_n) \end{matrix}\ ;\ x_0, y_0 = \text{given}$

The first example of a one variable iteration is the *Logistic Parabola*. Its iteration formula is given by

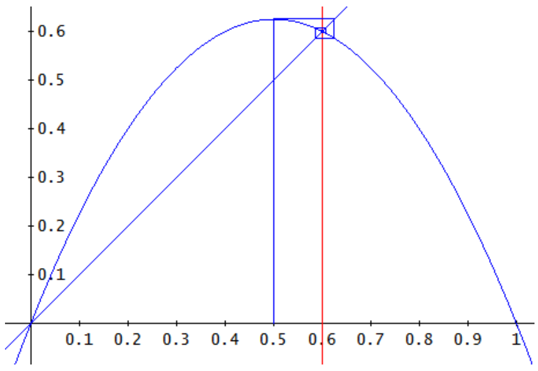
$$x_{n+1} = r \cdot x_n \cdot (1 - x_n).$$

The question is: What happens when plotting the graph of x_{n+1} versus x_n?

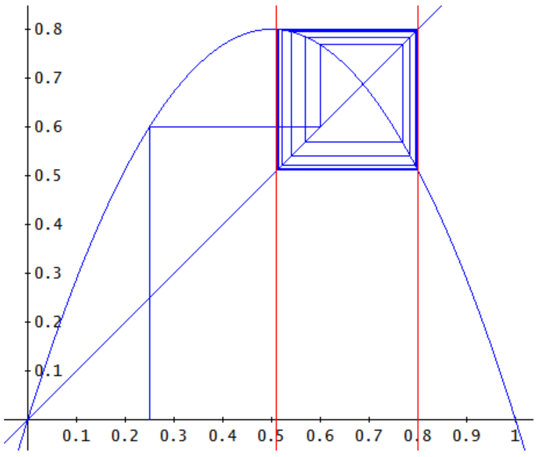
First, we investigate how the sequence develops. We apply DERIVE's versatile ITERATES-command and show the last *k* of the first *n* elements of the sequence.

```
#1:    [Precision := Approximate, Notation := Decimal, NotationDigits := 4]

       last_k(r, x0, n, k) := VECTOR((ITERATES(r·x·(1 - x), x, x0, n))  , i, n - k + 1, n)
#2:                                                                   i

#3:    last_k(2.5, 0.3, 100, 10) = [0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6]

#4:    last_k(3.2, 0.5, 100, 10)

#5:    [0.5130, 0.7994, 0.5130, 0.7994, 0.5130, 0.7994, 0.5130, 0.7994, 0.5130, 0.7994]

#6:    last_k(3.5, 0.75, 100, 10)

#7:    [0.8749, 0.3828, 0.8269, 0.5008, 0.8749, 0.3828, 0.8269, 0.5008, 0.8749, 0.3828]

#8:    last_k(3.8, 0.2, 100, 10)

#9:    [0.4438, 0.9380, 0.2209, 0.6540, 0.8598, 0.4578, 0.9432, 0.2034, 0.6157, 0.8991]
```

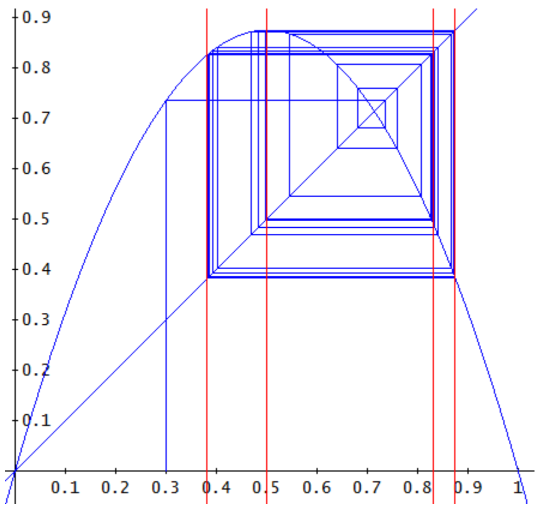
We observe the different behavior of the iteration process for different values for *r*. Let's illustrate this quite better by plotting the graph of x_{n+1} versus x_n producing a cobweb diagram:



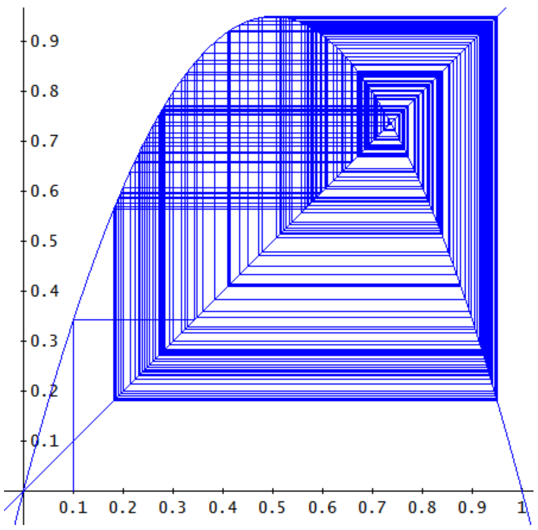
$r = 2.5$



$r = 3.2$



$r = 3.5$



$r = 3.8$

The figures given above illustrate the iteration process for different values for r plotting the cobweb diagrams. All iterations are starting with $x_0 = 0.3$. We can observe one-, two-, four-point attractors, and chaotic behavior, respectively. The end behavior does not depend on the initial value.

I present two ways to produce the logistic parabola together with the cobweb diagram and the straight line $y = x$.

You can take a program. #11 gives the first figure.

The purists among the *DERIVIANs* will miss the very special ITERATES-command for describing this iteration process. (Let's remember the huge and impressive ITERATEs-constructs composed by Hannes Wiesenbauer!).

```
       log_parab(r, x0, n, i, f, web) :=
         Prog
           web := [[x0, 0]]
           i := 1
           Loop
#10:         If i > n exit
             f := r·x0·(1 - x0)
             web := APPEND(web, [x0, f; f, f])
             x0 := f
             i :+ 1
           [web, y = x, r·x·(1 - x)]

#11:   log_parab(2.5, 0.5, 100)
```

In DNL#13 you can find a short note of Jerry Glynn providing a respective routine. Here is a similar one:

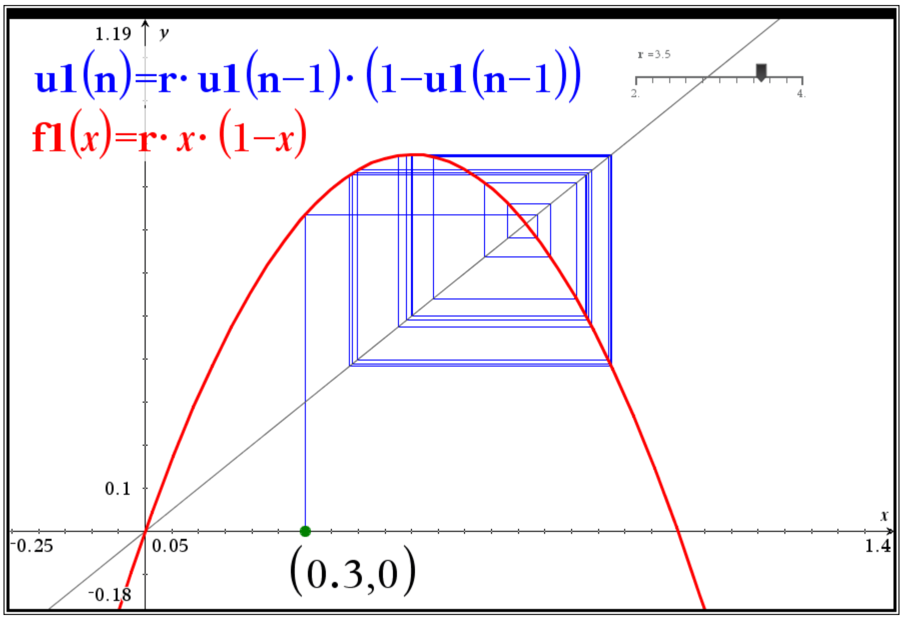
```
f(r, x) := r·x·(1 - x)
```

$$\text{cobw}(r,\ x0,\ n) := \left[f(r,\ x),\ y = x,\ \begin{bmatrix} x0 & 0 \\ x0 & x0 \end{bmatrix},\ \text{ITERATES}\left(\begin{bmatrix} v_{3,1} & v_{3,1} \\ v_{3,1} & f(r,\ v_{3,1}) \\ f(r,\ v_{3,1}) & f(r,\ v_{3,1}) \end{bmatrix},\ v,\ \begin{bmatrix} x0 & x0 \\ x0 & f(r,\ x0) \\ f(r,\ x0) & f(r,\ x0) \end{bmatrix},\ n \right) \right]$$

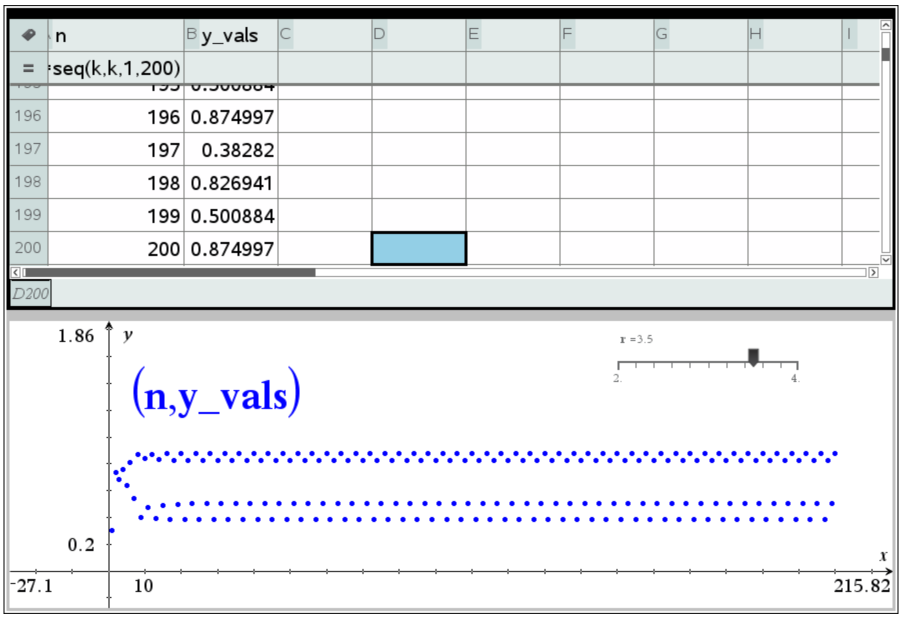
```
cobw(3.5, 0.3, 200)
```

We can observe one-, two- or four-point attractors, and chaotic behavior, respectively.

TI-NspireCX CAS allows comfortable plotting without programming. We define the iteration in the sequence entry and install a slider for r. We set a point on the x-axis (green) and link its x-coordinate to a variable, say $x0$, which is used as initial value for the sequence. Then we can move the initial point and vary the r-value as well.

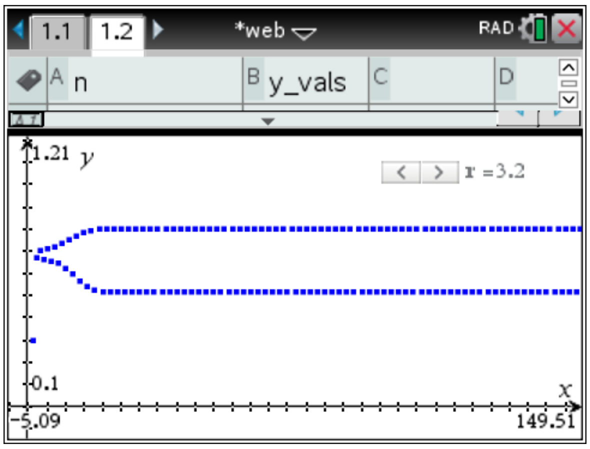


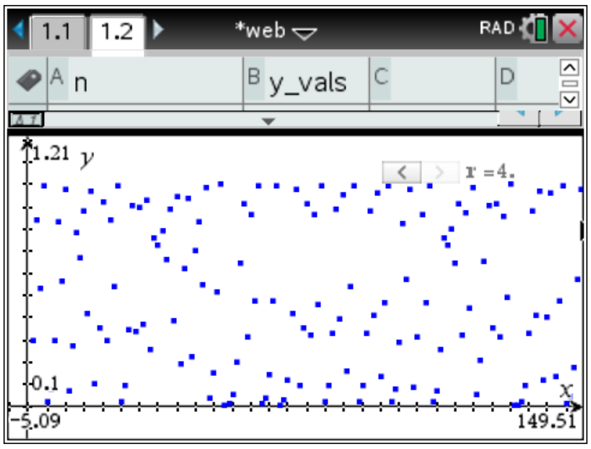
Two columns in the spreadsheet and their representation as a scatter diagram demonstrate the appearance of the attractors in combination with the slider in another attractive way



	A n	B y_vals	C	D	E	F	G	H	I
=	=seq(k,k,1,200)								
195	195	0.500884							
196	196	0.874997							
197	197	0.38282							
198	198	0.826941							
199	199	0.500884							
200	200	0.874997							

D200





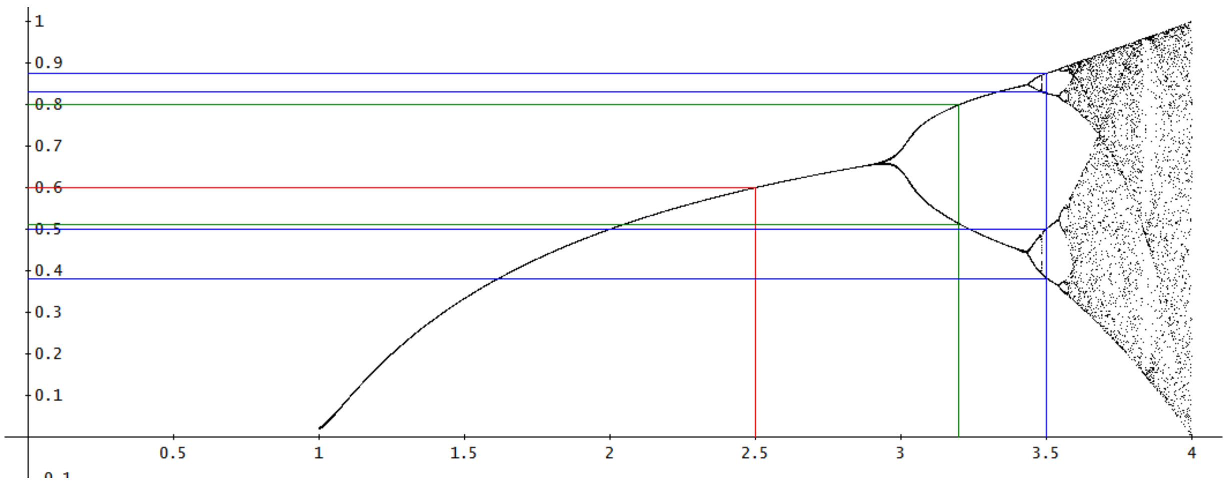
The so-called *period doubling* and its transition to *chaos* can be presented in another famous form.

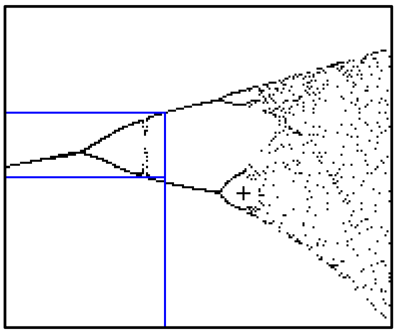
The key word is **Bifurcation** (Feigenbaum-diagram):

We calculate for all values r with $1 < r < r\_e$ with step size *inc* the first n elements of the iteration and we plot the last k values against r. So, we can observe the attractors.

#20 generates the k points belonging to one given r and initial value $x0$ and #21 composes all vertical sets of points to the Feigenbaum diagram. The plot of #22 is presented on the next page. Calculation does not need too long time. You might find out how many points are plotted.

```
        bifurc(r, x0, n, k) := VECTOR([r, (last_k(r, x0, n, k))  ], i, k)
#20:                                                            i

#21:    feigb(x0, n, k, r_e, inc) := VECTOR(bifurc(r_, x0, n, k), r_, 1, r_e, inc)

#22:    feigb(0.3, 50, 10, 4, 0.001)
```

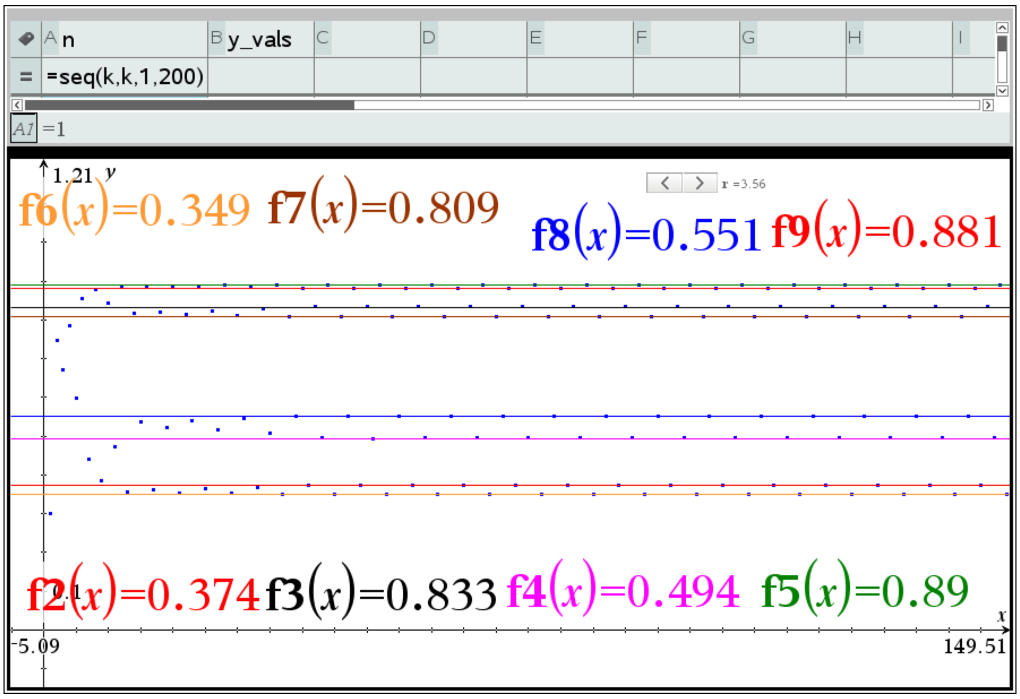




The endpoint behavior for r 2.5, 3.2 and 3.5 is marked in red, green and blue, respectively. Inspecting this scatter plot, we might get the impression that there appears at least one more period doubling. Right of the blue line $x = 3.5$ we detect four "bubbles" lying in a vertical line.

We zoom in and read off the x-coordinate of the cross ≈ 3.56. When we calculate the last 20 elements of the first 200 ones we really can confirm that there is a period of eight elements.

```
#23:  last_k(3.56, 0.4, 200, 20)

#24:  [0.3738, 0.8333, 0.4944, 0.8898, 0.3488, 0.8086, 0.5508, 0.8807, 0.3738, 0.8333,

        0.4944, 0.8898, 0.3488, 0.8086, 0.5508, 0.8807, 0.3738, 0.8333, 0.4944, 0.8898]
```



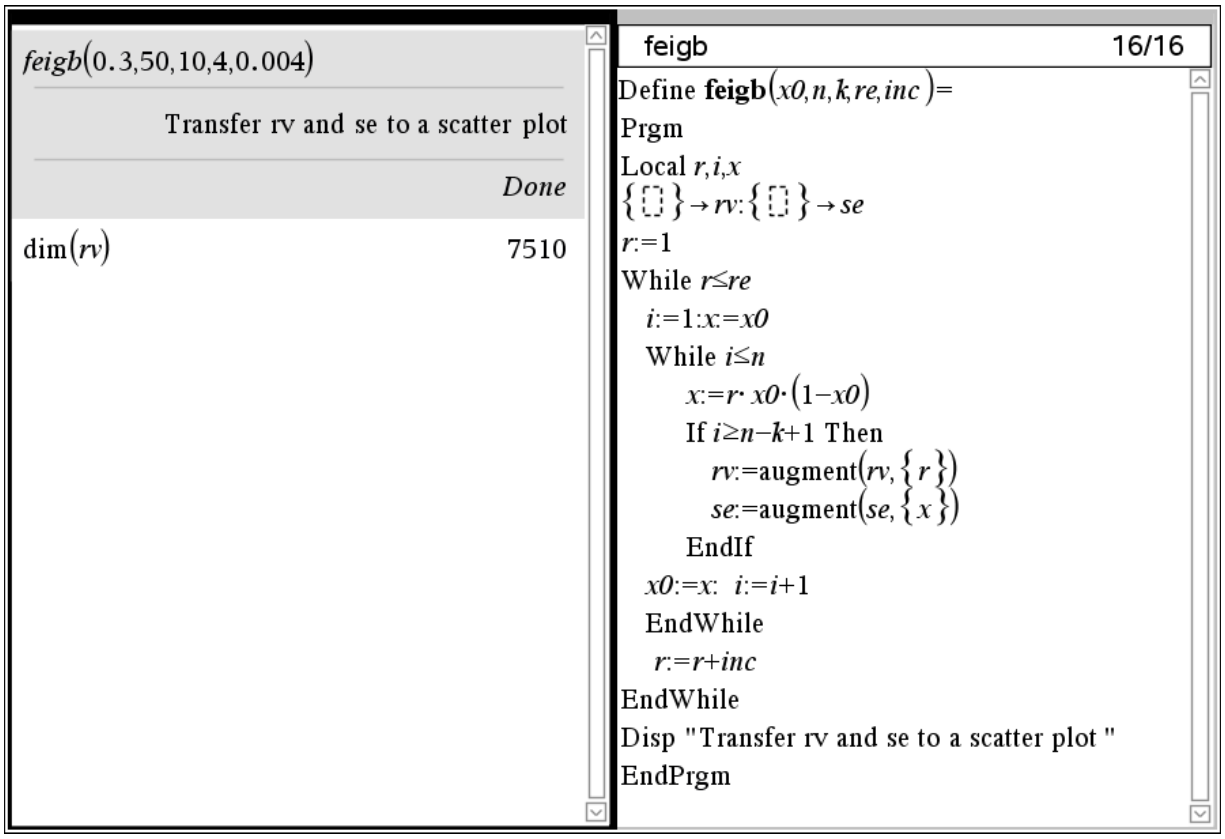
A n	B y_vals	C	D	E	F	G	H	I
=seq(k,k,1,200)								

A1 =1

$f6(x)=0.349$ $f7(x)=0.809$ $f8(x)=0.551$ $f9(x)=0.881$

$f2(x)=0.374$ $f3(x)=0.833$ $f4(x)=0.494$ $f5(x)=0.89$

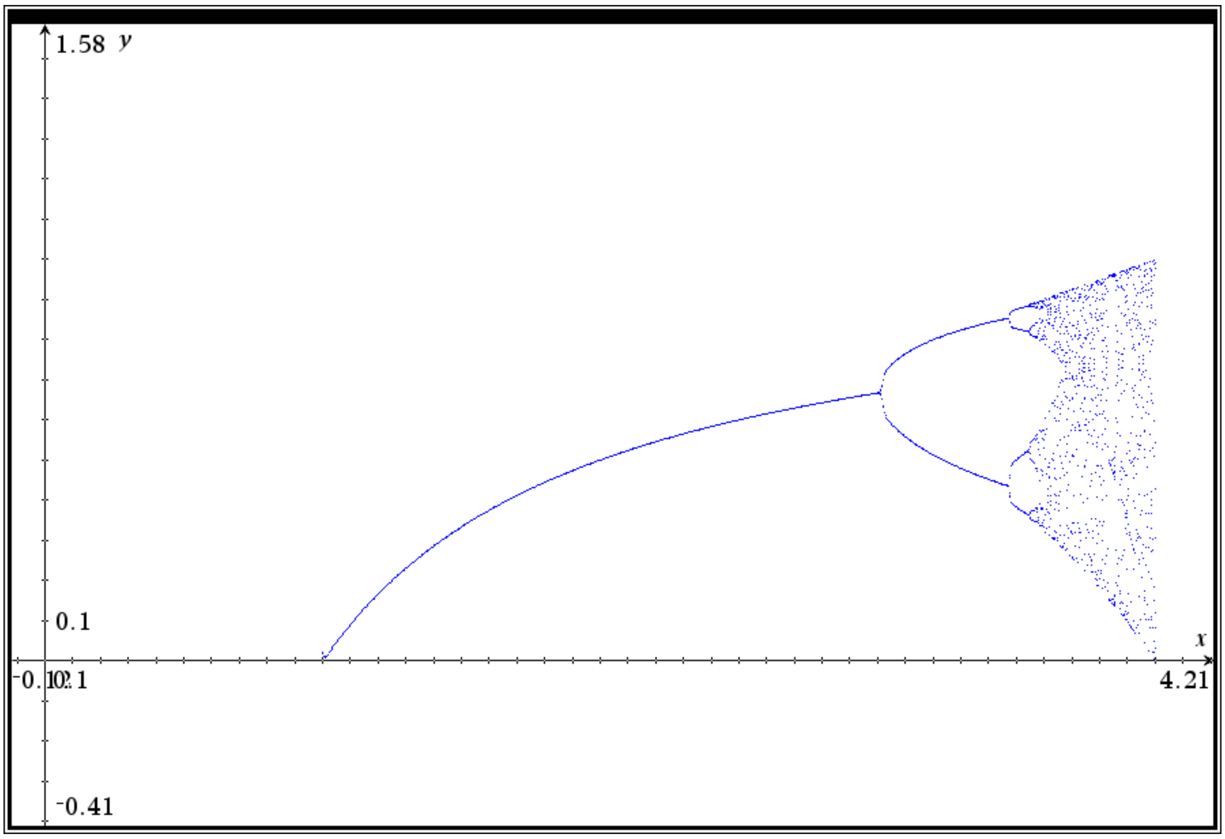
The question is: Can we produce the Feigenbaum diagram on the TI-NspireCAS screen, too. To answer with Barack Obama: “Yes, we can!”.



```
feigb(0.3,50,10,4,0.004)
                    Transfer rv and se to a scatter plot
                                                    Done
dim(rv)                                             7510
```

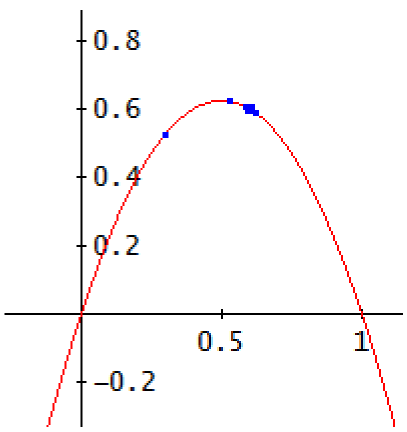
```
feigb                                              16/16
Define feigb(x0,n,k,re,inc)=
Prgm
Local r,i,x
{□}→rv:{□}→se
r:=1
While r≤re
  i:=1:x:=x0
  While i≤n
     x:=r· x0·(1−x0)
     If i≥n−k+1 Then
       rv:=augment(rv,{r})
       se:=augment(se,{x})
     EndIf
  x0:=x:  i:=i+1
  EndWhile
   r:=r+inc
EndWhile
Disp "Transfer rv and se to a scatter plot "
EndPrgm
```

Program feigb with the same syntax as DERIVE `feigb` generates lists rv (*r*-values) and se (sequence elements) which can be placed into the Scatter Plot Graph Entry. Gives a pretty plot!

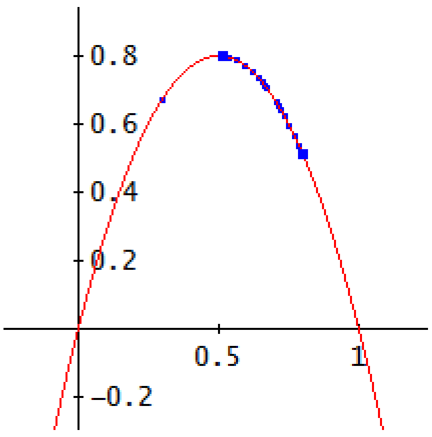


Before proceeding to the next chapter, I would like to describe one more possibility visualizing the iteration process together with the attractors.

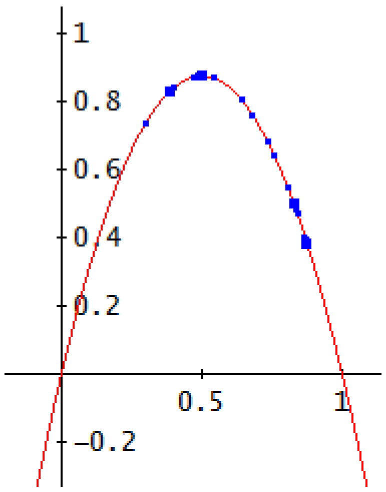
$\mathrm{lp}(r,\ x0,\ n) := \mathrm{ITERATES}\left(\left[v_2,\ f(r,\ v_2)\right],\ v,\ [x0,\ f(r,\ x0)],\ n\right)$



r = 2.5, 1000 iterated points



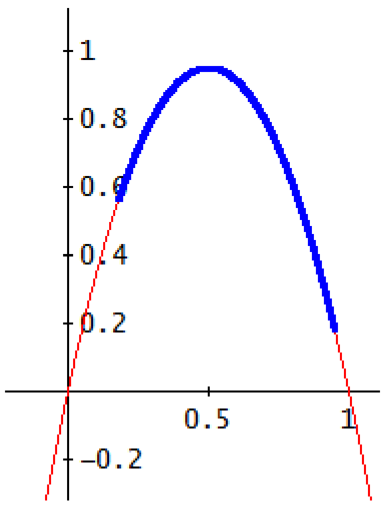
r = 3.2, 1000 points



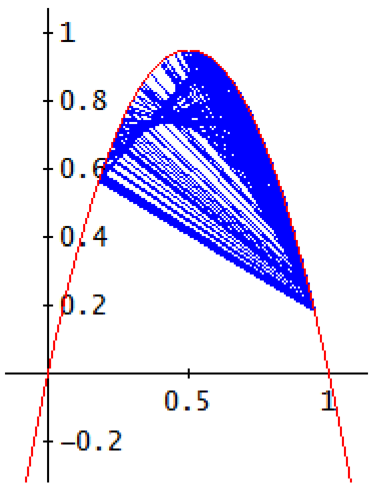
r = 3.5, 1000 points

I plot the logistic parabola (red) and the first 1000 or 10 000 points (x_i, x_{i+1}) in blue (medium size). We see that all points are lying on the parabola. The attractor points are also in blue but in large size. There are no attractors for *r* = 3.8.

Once I forgot to deactivate the Points >Connect>No box and found the family of chords plotted. I wonder if there is an envelope of the chords – and how to find it.



r = 3.8, 10 000 points

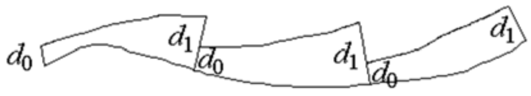


200 points, Connected

2 The Lyapunov-exponent

As we are approaching chaos now we need a measure to test chaotic behavior. Chaotic processes are extremely sensitive to initial conditions. Reading this you might remember the famous "Butterfly Effect".

The general idea is to follow two nearby orbits and to calculate their average logarithmic rate of separation. The Lyapunov exponent λ is the power of 2 by which the difference between two nearly equal *x*-values changes on average for each iteration i.e. the difference changes by an average of 2^λ for each iteration. (This is a definition for a one-dimensional iteration. It is adapted in case of more dimensional systems.)



Start with any point on the attractor and choose a nearby point separated by d_0. Iterate both values and find their distance d_1. Now calculate $\log_2\left|\frac{d_1}{d_0}\right|$. Readjust one orbit so its separation is d_0 in the same direction as d_1 and repeat this procedure many times and then calculate the average of the logarithms [3].

If $\lambda < 0$ then the solutions approach another $\rightarrow$ we receive a point attractor
but
if $\lambda > 0$ then the process is sensitive to initial conditions $\rightarrow$ we can expect chaotic behavior $\rightarrow$ we receive a **strange attractor**.

This is another interpretation: The Lyapunov-exponent λ (LE) is the rate at which information is lost when a map is iterated.

In case of one dimensional iterations we can use means of calculus to find the LE.

$$x_{i+1} = f(x_i)$$

x_i and neighbour $\tilde{x}_i = x_i + \Delta_i$

$$\lim_{\Delta_i \to 0} \frac{\Delta_{i+1}}{\Delta_i} = \lim_{\Delta_i \to 0} \frac{\tilde{x}_{i+1} - x_{i+1}}{\Delta_i} = \frac{f(x_i + \Delta_i) - f(x_i)}{\Delta_i} =$$

$$= \left| f'(x_i) \right|$$

$$\lambda = \lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{N} \ln\left| f'(x_i) \right|$$

Applying this formula on the logistic equation we receive:

$$x_{i+1} = r\,x_i(1-x_i) = r\,x_i - r\,x_i^2 = f(x_i)$$

$$\left| f'(x_i) \right| = \left| r - 2r\,x_i \right|$$

$$\lambda = \frac{1}{N} \sum_{i=1}^{N} \log_2 \left| r - 2r\,x_i \right|$$

I wrote a short DERIVE program for testing the Lyapunov-exponent:

```
lyaptest(cf, its, t_, xn, x0, o, f, f1, i, l) :=
  Prog
    o := DIM(cf) - 1
    f := cf·VECTOR(x^k, k, 0, o)
    f1 := ∂(f, x)
    [i := 1, l := 0, x0 := 0.05]
    x0 := ITERATE(f, x, x0, 14 - 2·o)
    Loop
      If ABS(x0) > 100 exit
      If i > its
         RETURN l/its
      xn := LIM(f, x, x0)
      l := l + LOG(ABS(LIM(f1, x, xn)), 2)
      x0 := xn
      i :+ 1
```

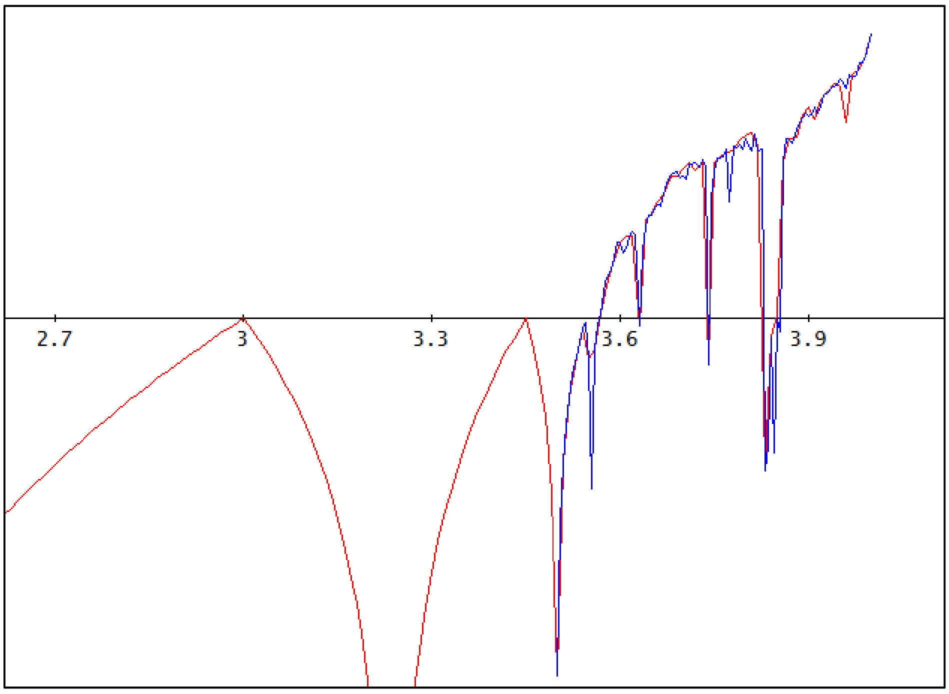
Let us check the LEs of the logistic parabolas from above.

You can see that for $r = 3.8$ the LE turns out to be positive.

```
lyaptest([0, 2.5, -2.5], 2000) = -1.0000
lyaptest([0, 3.2, -3.2], 2000) = -1.3219
lyaptest([0, 3.5, -3.5], 2000) = -1.2587
lyaptest([0, 3.8, -3.8], 2000) = 0.61736
```

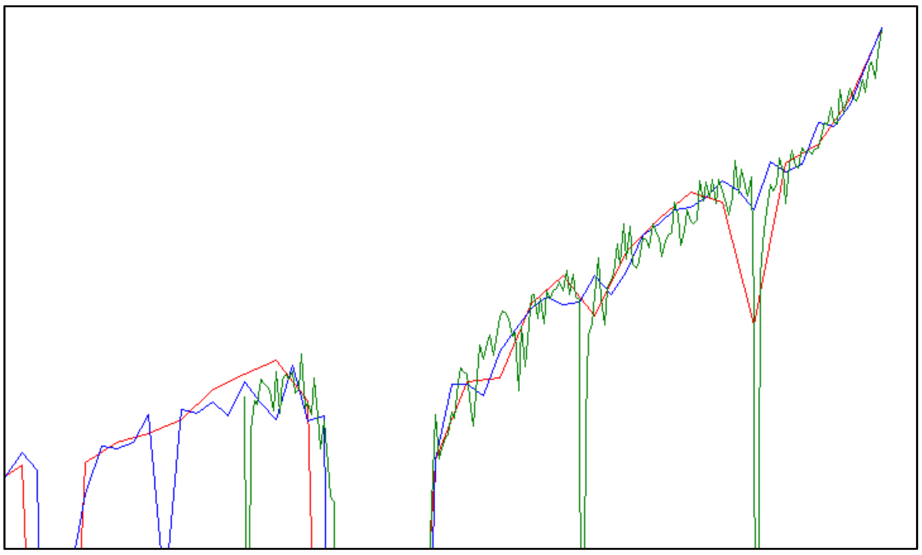
`cf` are the coefficients of the parabola in ascending order, e.g. [0, 3.2, -3.2] for $3.2x\cdot(1 - x) = 3.2x - 3.2x^2$. `its` is the number of iterations – which is N in the formulas given above. We will use this program for polynomials of higher order, too.#

In [3] I found the graph of the LE depending on r for the logistic parabolas with $2 \le r \le 4$. Applying the basics of the program from above we can produce this interesting graph using program `lyape_f(iterations, r_start, r_end, r_inc)`.



`lyape_f(1000,2.5,4,0.01)` (red) and `lyape_f(1000,3.5,4,0.005)` (blue)

The zeros of the graph indicate the position of the bifurcations. Zooming into the right upper corner shows the fractal structure of this graphic representation (`lyape_f(1000,3.8,4,0.001)` in green).



3 The first Investigation

We will try to extend the logistic parabola to ordinary parabolas and further to parabolas (polynomials) of higher order. At the moment, we have the only chance to find quadratic parabolas with a positive LE by try and error:

My first try:

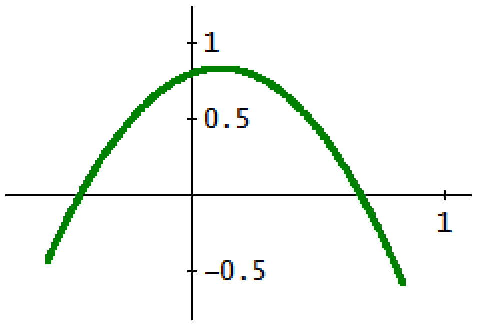
```
            2
f(x) := 3.2·x  + 1.2·x - 0.1

lyaptest([-0.1, 1.2, 3.2], 1000) = -2.7474
```

$$\text{ITERATES}\left(\left[v_2,\ f(v_2)\right],\ v,\ [0.1,\ f(0.1)],\ 5\right) = \begin{bmatrix} 0.1 & 0.052 \\ 0.052 & -0.028947 \\ -0.028947 & -0.13205 \\ -0.13205 & -0.20266 \\ -0.20266 & -0.21176 \\ -0.21176 & -0.21061 \end{bmatrix}$$

This was not very successful. We have a point attractor, which is not surprising because the LE is negative. Let’s try another one:

```
             2
g(x) := - 2.7·x  + 0.6·x + 0.8

lyaptest([0.8, 0.6, -2.7], 1000) = 0.86524
```

$$\text{ITERATES}\left(\left[v_2,\ g(v_2)\right],\ v,\ [0.1,\ g(0.1)],\ 1000\right)$$



The positive LE is promising. We plot x_{n+1} vs x_n.

It is boring and frustrating finding “attractive” parabolas and polynomials just by trying sets of coefficients. Very soon we will provide a tool for organizing this search which will guarantee success.

How to obtain “attractive” polynomials of higher order? Once found a parabola like $g(x)$ this is an easy job. We plot x_{n+k} (k steps later iteration) versus x_n with $k \geq 1$.

The next program gives this plot with `lits` = k with `lits = 1` by default.

```
pol_map(cf, its, x0, lits := 1, f, xn, xn_, vals, i) :=
  Prog
    vals := []
    i := 1
    f := cf·VECTOR(x^k, k, 0, DIM(cf) - 1)
    Loop
      If i > its
         RETURN vals
      xn := LIM(f, x, x0)
      xn_ := ITERATE(f, x, x0, lits)
      vals := APPEND(vals, [[x0, xn_]])
      x0 := xn
      i :+ 1
```

The reason for entering the parabola (and later the polynomials) in form of a list of the coefficients - and not as an expression - will be explained in the next paragraph.

We will compare the first generated points changing the last parameter of this function from 1 (by default) to 2 and further to 3.

```
pol_map([0.8, 0.6, -2.7], 5, 0.1)

⎡    0.1       0.833  ⎤
⎢   0.833    -0.57370 ⎥
⎢ -0.57370   -0.43287 ⎥
⎢ -0.43287   0.034342 ⎥
⎣ 0.034342    0.81742 ⎦
```

```
pol_map([0.8, 0.6, -2.7], 5, 0.1, 2)

⎡    0.1     -0.57370 ⎤
⎢   0.833    -0.43287 ⎥
⎢ -0.57370   0.034342 ⎥
⎢ -0.43287    0.81742 ⎥
⎣ 0.034342   -0.51362 ⎦
```

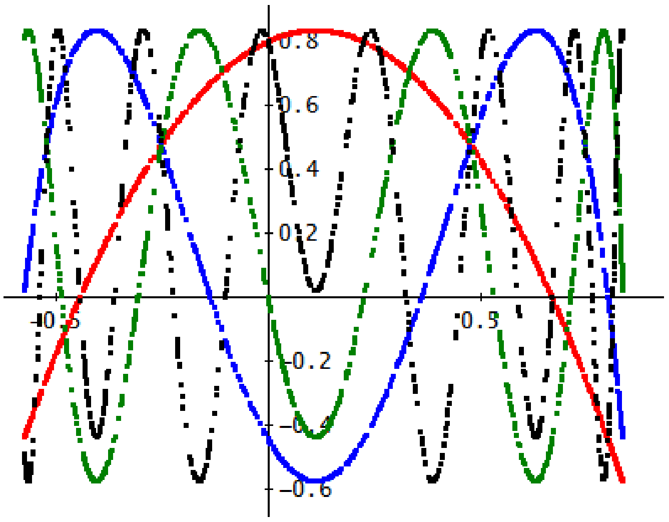
```
pol_map([0.8, 0.6, -2.7], 5, 0.1, 3)

⎡    0.1     -0.43287 ⎤
⎢   0.833    0.034342 ⎥
⎢ -0.57370    0.81742 ⎥
⎢ -0.43287   -0.51362 ⎥
⎣ 0.034342   -0.22046 ⎦
```

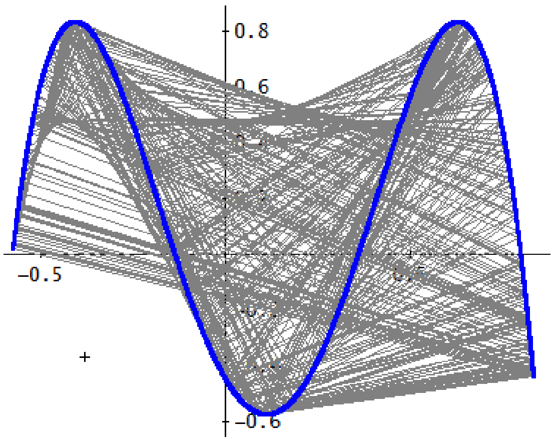
```
EXPAND(ITERATE(g(v), v, x, 2))

          4          3          2
- 19.683·x  + 8.748·x  + 9.072·x  - 2.232·x - 0.448

⎡    0.1     -0.57370 ⎤
⎢ -0.57370   0.034342 ⎥
⎢ 0.034342   -0.51362 ⎥
⎢ -0.51362    0.53648 ⎥
⎣  0.53648    0.68591 ⎦
```

Can you find respective points in this table and in the table above?

We plot $g(x)$ together with the first 1000 points of x_{i+1} vs x_i (red), x_{i+2} vs x_i (blue), x_{i+3} vs x_i (green) and x_{i+4} vs x_i (black) which is a polynomial of order 16.

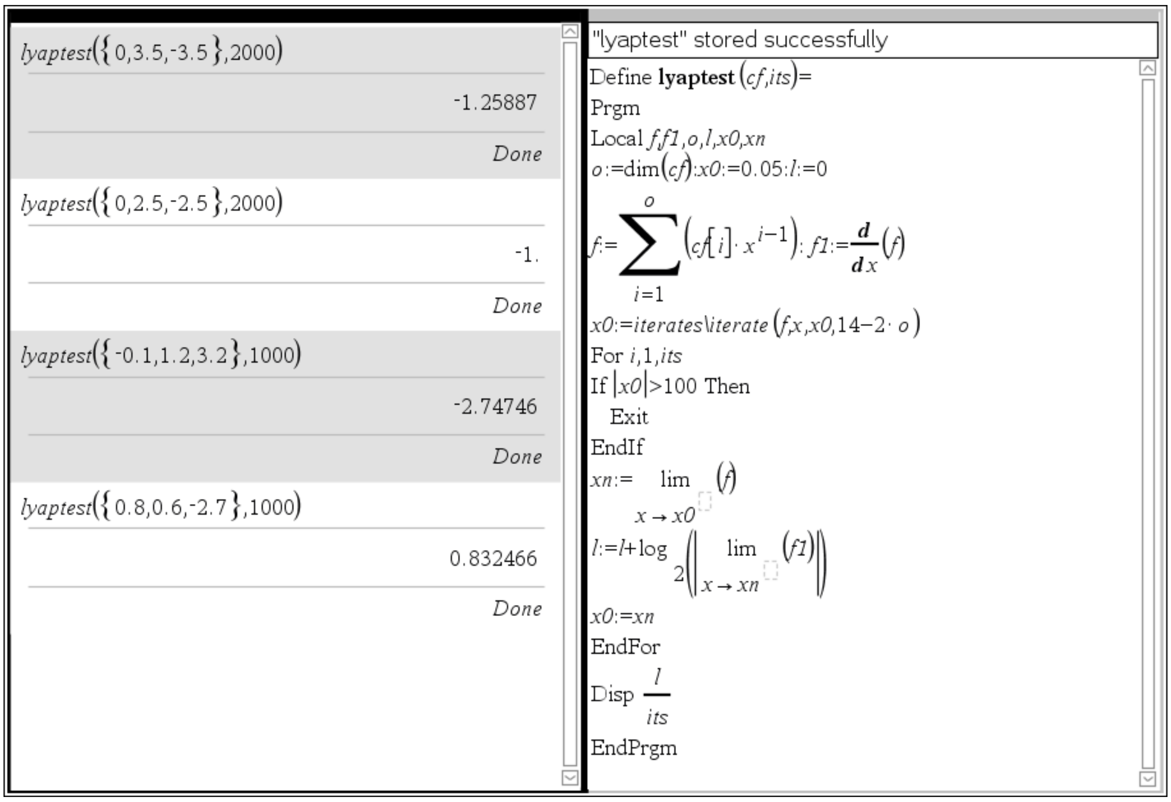
```
pol_map([0.8, 0.6, -2.7], 1000, 0.1)

pol_map([0.8, 0.6, -2.7], 1000, 0.1, 2)

pol_map([0.8, 0.6, -2.7], 1000, 0.1, 3)

pol_map([0.8, 0.6, -2.7], 1000, 0.1, 4)
```





Plotting the points connected gives the family of chords and we again get the impression of a possible envelope.

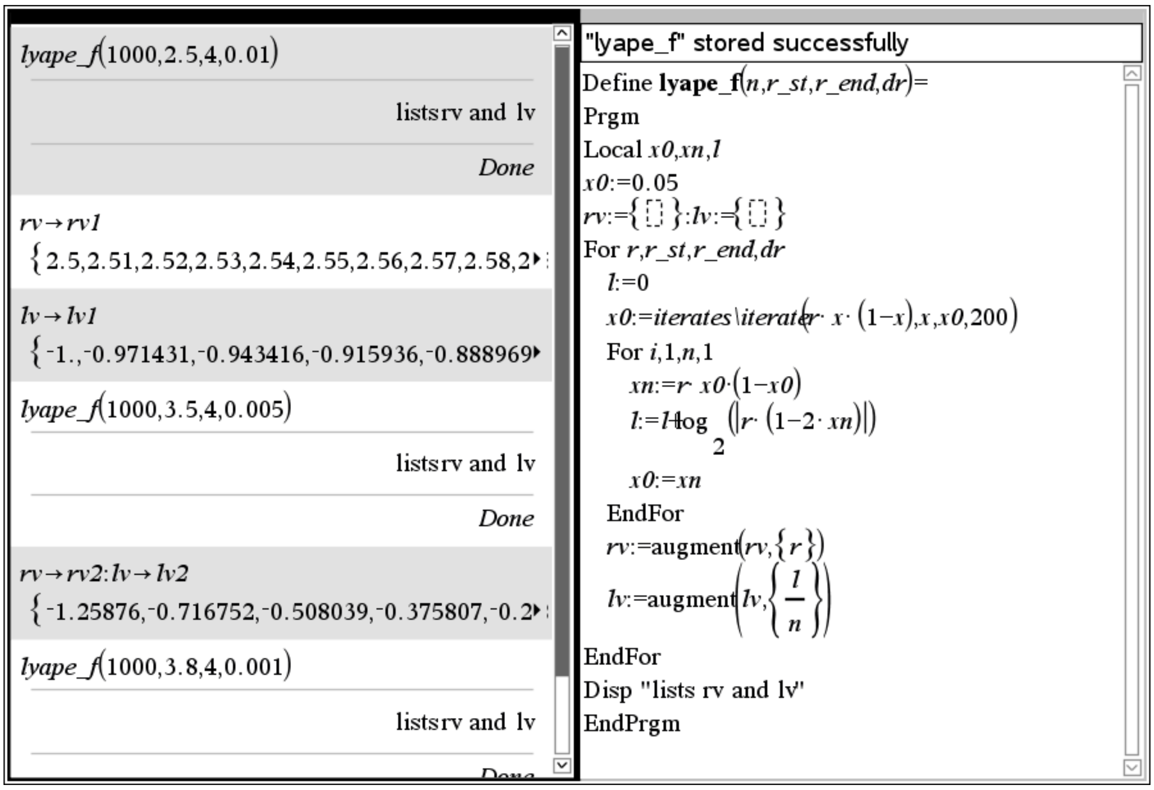
Now let’s turn on the TI-NspireCAS again. Calculation of the Lyapunov exponent follows the DERIVE program from above. VECTOR must be replaced by a sequence and – more important – we have the common for-next loop available.



```
lyaptest({0,3.5,-3.5},2000)
                                   -1.25887
                                       Done
lyaptest({0,2.5,-2.5},2000)
                                        -1.
                                       Done
lyaptest({-0.1,1.2,3.2},1000)
                                   -2.74746
                                       Done
lyaptest({0.8,0.6,-2.7},1000)
                                   0.832466
                                       Done
```

```
"lyaptest" stored successfully
Define lyaptest(cf,its)=
Prgm
Local f,f1,o,l,x0,xn
o:=dim(cf):x0:=0.05:l:=0
$f:=\sum_{i=1}^{o}\left(cf[i]\cdot x^{i-1}\right):f1:=\frac{d}{dx}(f)$
x0:=iterates\iterate(f,x,x0,14−2·o)
For i,1,its
If |x0|>100 Then
  Exit
EndIf
$xn:=\lim_{x\to x0}(f)$
$l:=l+\log_2\left(\left|\lim_{x\to xn}(f1)\right|\right)$
x0:=xn
EndFor
$\text{Disp } \frac{l}{its}$
EndPrgm
```

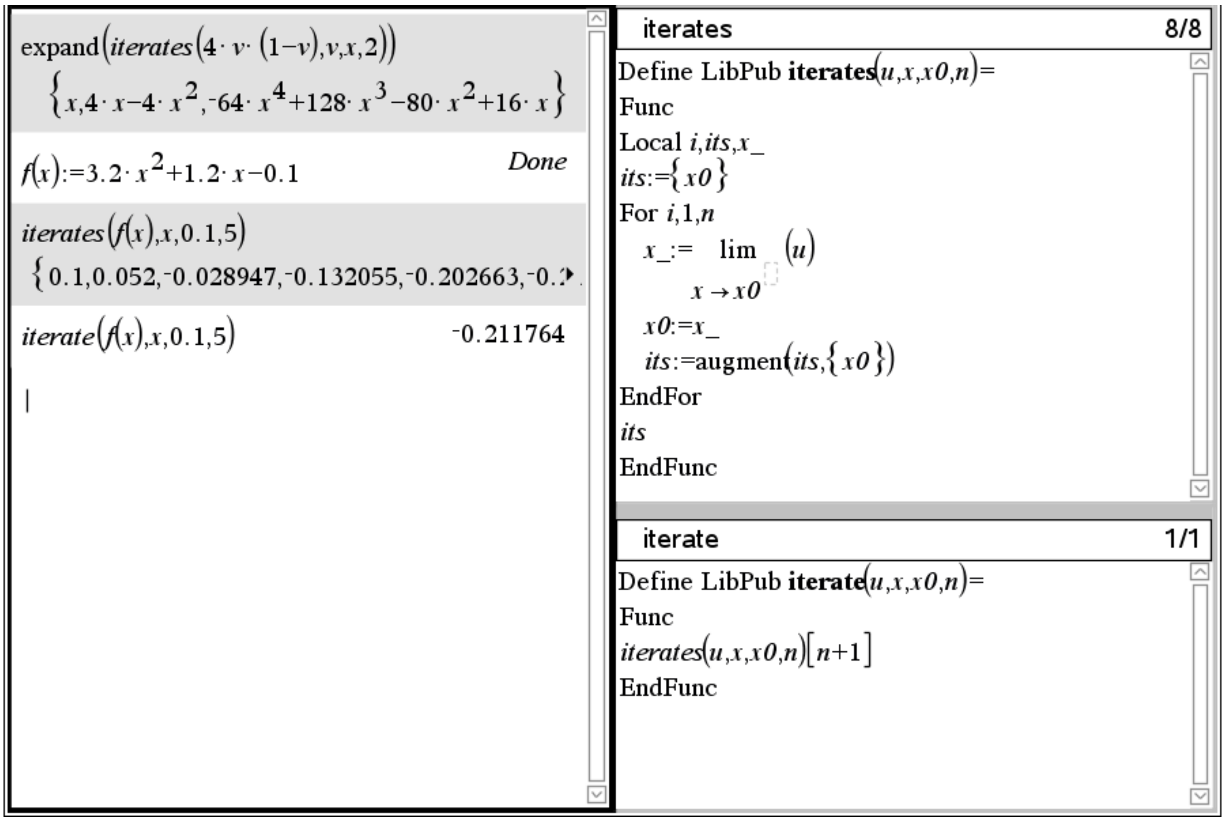
You will notice *iterate* for TI-NspireCAS. It is not implemented, so we define our own *iterate*-function and *iterates* as well. *Michel Beaudin* provides a more powerful *iterate* (and *iterates* as well) in his TI-Nspire toolbox but this one is sufficient enough for our purpose.



```
lyape_f(1000,2.5,4,0.01)
                               listsrv and lv
                                         Done
rv→rv1
 {2.5,2.51,2.52,2.53,2.54,2.55,2.56,2.57,2.58,2▸
lv→lv1
 {-1.,-0.971431,-0.943416,-0.915936,-0.888969▸
lyape_f(1000,3.5,4,0.005)
                               listsrv and lv
                                         Done
rv→rv2:lv→lv2
 {-1.25876,-0.716752,-0.508039,-0.375807,-0.2▸
lyape_f(1000,3.8,4,0.001)
                               listsrv and lv
```

```
"lyape_f" stored successfully
Define lyape_f(n,r_st,r_end,dr)=
Prgm
Local x0,xn,l
x0:=0.05
rv:={ }:lv:={ }
For r,r_st,r_end,dr
  l:=0
  x0:=iterates\iterate(r·x·(1−x),x,x0,200)
  For i,1,n,1
    xn:=r·x0·(1−x0)
    $l:=l+\log_2(|r\cdot(1-2\cdot xn)|)$
    x0:=xn
  EndFor
  rv:=augment(rv,{r})
  $lv:=\text{augment}\left(lv,\left\{\frac{l}{n}\right\}\right)$
EndFor
Disp "lists rv and lv"
EndPrgm
```

This is our *iterate* for TI-Nspire. (I put both function in a public library in order to have it available at any occasion.)

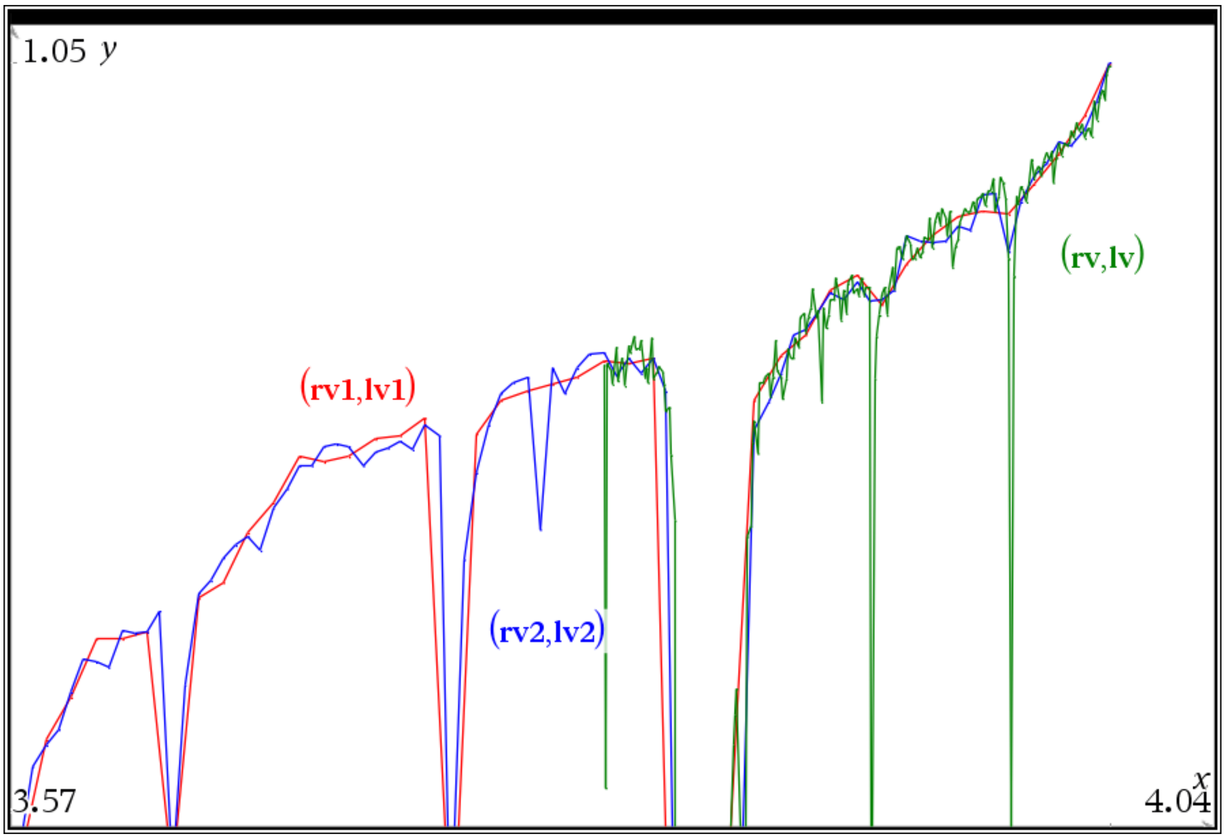


```
expand(iterates(4·v·(1−v),v,x,2))
   {x,4·x−4·x^2,-64·x^4+128·x^3−80·x^2+16·x}
f(x):=3.2·x^2+1.2·x−0.1                          Done
iterates(f(x),x,0.1,5)
 {0.1,0.052,-0.028947,-0.132055,-0.202663,-0.▸
iterate(f(x),x,0.1,5)                      -0.211764
```

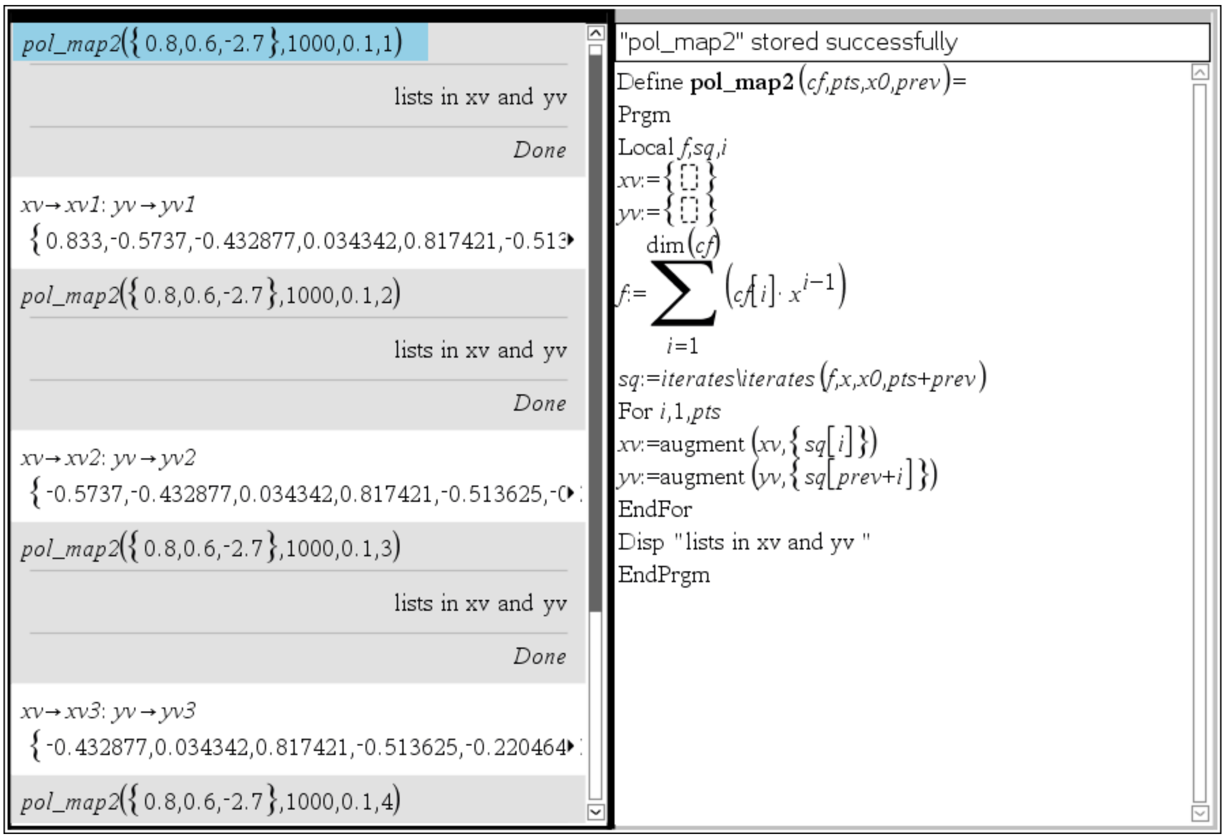
```
iterates                                          8/8
Define LibPub iterates(u,x,x0,n)=
Func
Local i,its,x_
its:={x0}
For i,1,n
  x_:=  lim   (u)
       x→x0
  x0:=x_
  its:=augment(its,{x0})
EndFor
its
EndFunc
```

```
iterate                                           1/1
Define LibPub iterate(u,x,x0,n)=
Func
iterates(u,x,x0,n)[n+1]
EndFunc
```

You are invited to compare the plot of the Lyapunov-exponent function.



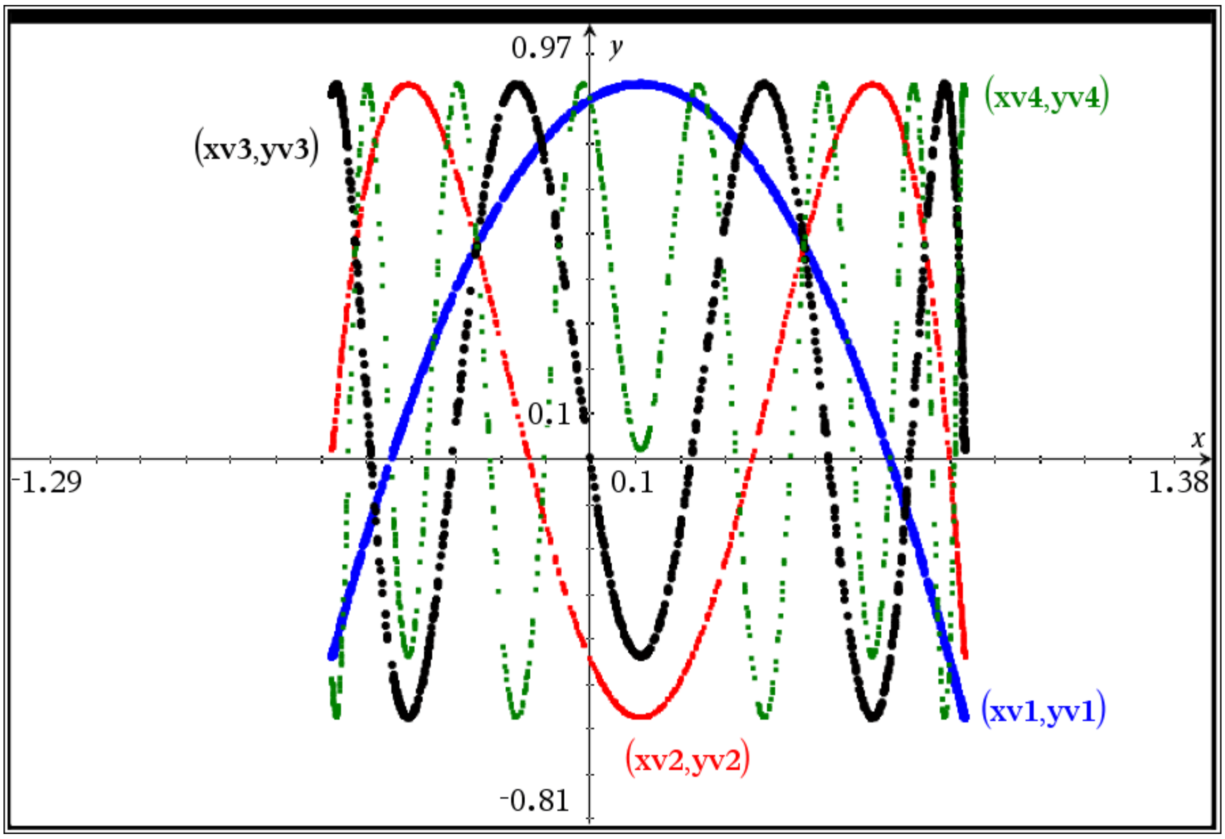
I produced a program *polmap* as a “translation” from DERIVE-syntax to TI-Nspire syntax. Here I show another way to plot the attractor, generating the whole sequence of iterations first followed by gathering the elements for the two lists which are needed for the scatter plot, compare with `polmap` from above.



```
pol_map2({0.8,0.6,-2.7},1000,0.1,1)
                                        lists in xv and yv
                                                      Done
xv→xv1: yv→yv1
 {0.833,-0.5737,-0.432877,0.034342,0.817421,-0.513▸
pol_map2({0.8,0.6,-2.7},1000,0.1,2)
                                        lists in xv and yv
                                                      Done
xv→xv2: yv→yv2
 {-0.5737,-0.432877,0.034342,0.817421,-0.513625,-0▸
pol_map2({0.8,0.6,-2.7},1000,0.1,3)
                                        lists in xv and yv
                                                      Done
xv→xv3: yv→yv3
 {-0.432877,0.034342,0.817421,-0.513625,-0.220464▸
pol_map2({0.8,0.6,-2.7},1000,0.1,4)
```

"pol_map2" stored successfully

```
Define pol_map2(cf,pts,x0,prev)=
Prgm
Local f,sq,i
xv:={[]}
yv:={[]}
f:= Σ(i=1 to dim(cf)) (cf[i]·x^(i-1))
sq:=iterates\iterates(f,x,x0,pts+prev)
For i,1,pts
xv:=augment(xv,{sq[i]})
yv:=augment(yv,{sq[prev+i]})
EndFor
Disp "lists in xv and yv "
EndPrgm
```



4 Motivation and Inspiration as well

My motivation and inspiration as well to deal with strange attractors was a book written by Julien C. Sprott:

Strange Attractors:
Creating Patterns in Chaos

by
Julien C. Sprott

and his incredible search for strange attractors [3]. This book can be downloaded as a pdf-file free of charge together with program SA.EXE (**S**earch **A**ttractors). This program is a compiled BASIC program (10 pages code). Among the downloadable files are numerous examples for wonderful strange attractors. Sprott's search machine is based on a random choice of coefficients for the polynomials needed for one and more dimensional systems. He encodes the coefficients from -4.5 to 5.0 (with increment = 0.1) with characters (from " " (ASCII 32) to "_" (ASCII 127)). Together with a leading character associated with the type of iteration each attractor can be described in a unique way. This is one example:

$$\text{decode(cd)} := \text{VECTOR}\left(\frac{c - 77}{10},\ c,\ \text{REST(NAME\_TO\_CODES(cd))}\right)$$

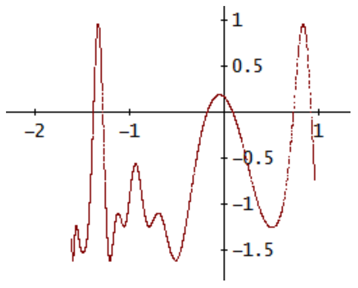
```
decode(CBLCTX) = [-1.1, -0.1, -1, 0.7, 1.1]

decode(EWM?MPMMWMMMM) = [1, 0, -1.4, 0, 0.3, 0, 0, 1, 0, 0, 0, 0]
```

"CBLCTX" describes a one-dimensional quartic map.

```
decode("CBLCTX") = [-1.1, -0.1, -1, 0.7, 1.1]
```

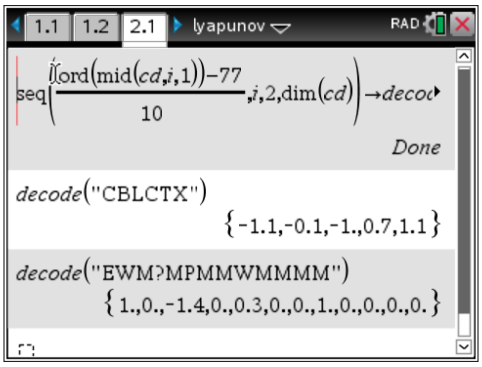
C is the code for a quartic, which reads as $f(x) = -1.1 - 0.1x - x^2 + 0.7x^3 + 1.1x^4$.

```
pol_map([-1.1, -0.1, -1, 0.7, 1.1], 10000, 0.05, 3)
```



will give the plot of the of x_{i+3} vs x_i plot of this attractor.

The second one is for a two-dimensional quadratic mapping (will be presented very soon).



```
seq((ord(mid(cd,i,1))-77)/10,i,2,dim(cd))→decod▸
                                         Done
decode("CBLCTX")
                        {-1.1,-0.1,-1.,0.7,1.1}
decode("EWM?MPMMWMMMM")
              {1.,0.,-1.4,0.,0.3,0.,0.,1.,0.,0.,0.,0.}
```

Sprott's codes for the coefficients of the iteration functions go up to 540 characters!!

The "decoding"-function for the TI-NspireCAS is given on the handheld screen.

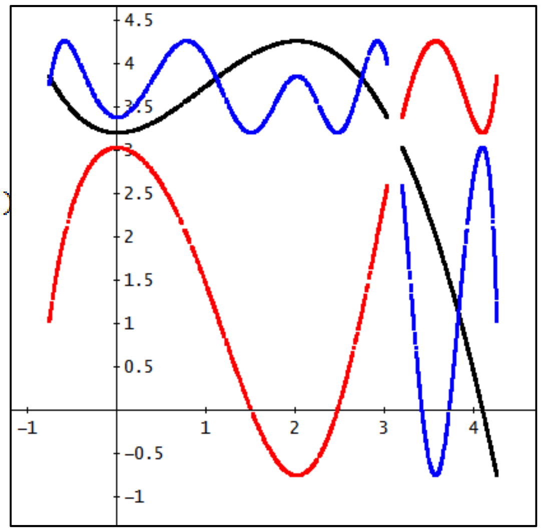
We could be satisfied to reproduce the numerous wonderful plots depicted in Sprott's book together with so many others which are given by their codes only. `pol_map` and some other similar programs would do this up to strange attractors in 3D and 4D space.

This is not sufficient for us and was not sufficient for Julien C. Sprott. He had the idea – and this is the core message of his book – to find own "self-made" strange attractors which had never been seen before.

5 Search for Polynomial Attractors

The trick is to create for polynomials of order *n* sets of randomly chosen coefficients between -5 and +5, compose the respective polynomials, interpret them as iteration function and then calculate their Lyapunov exponent. All sets of coefficients which give a positive LE are connected with an attractor.

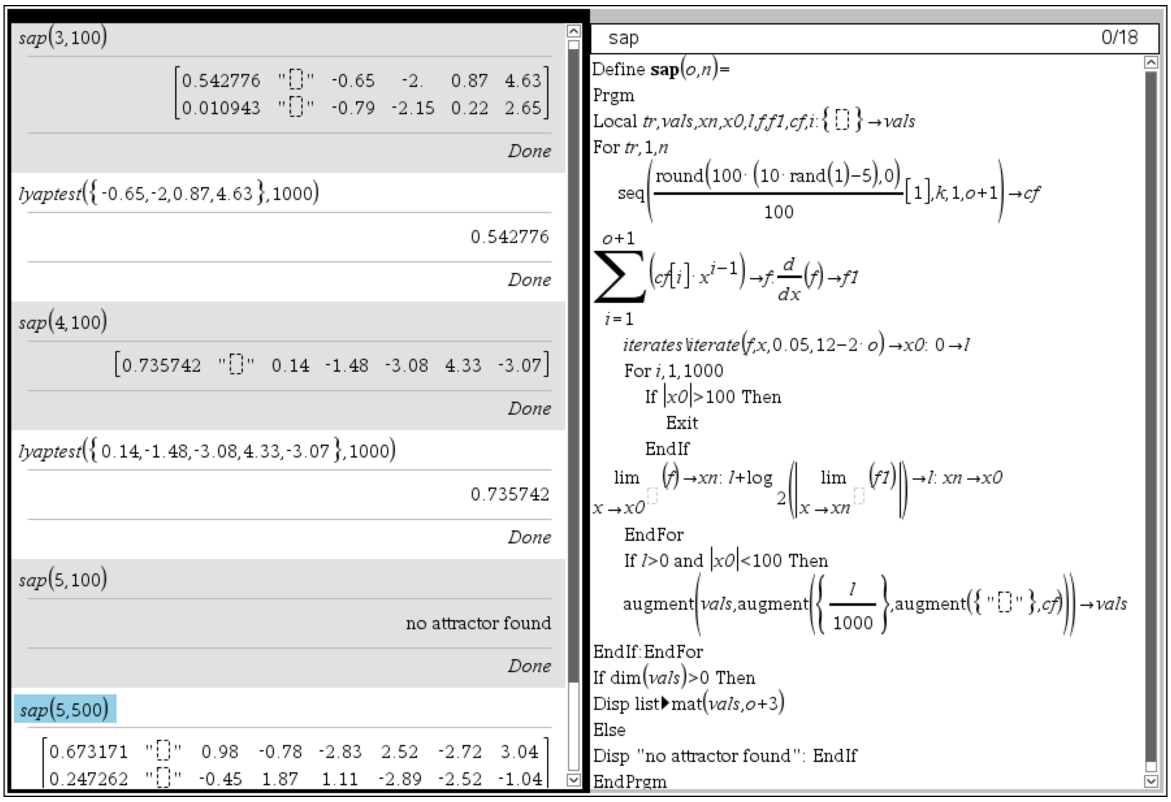
```
sap(o, n, tr, t_, xn, x0, f, f1, cf, vals, i, s, dummy, l) :=
  Prog
    [dummy := RANDOM(0), tr := 1, vals := []]
    Loop
      WRITE([tr, DIM(vals)])
      If tr > n exit
      cf := VECTOR(ROUND(100·(9.5·RANDOM(1) - 4.5))/100, k, o + 1)
      f := cf·VECTOR(x^k, k, 0, o)
      f1 := ∂(f, x)
      [i := 1, l := 0]
      x0 := ITERATE(f, x, 0.05, 14 - 2·o)
      Loop
        If ABS(x0) > 100 exit
        If i > 2000 ∧ l > 0
           vals := APPEND(vals, [[l/2000, cf]])
        If i > 2000 exit
        xn := LIM(f, x, x0)
        l := l + LOG(ABS(LIM(f1, x, xn)), 2)
        x0 := xn
        i :+ 1
      tr :+ 1
    RETURN vals
```



```
sap(4, 200)

⎡ 0.59115   [3.2, -0.01, 0.87, -0.34, 0.02]  ⎤
⎢                                            ⎥
⎢  0.341   [-0.58, 1.6, 3.94, -3.79, -1.29]  ⎥
⎢                                            ⎥
⎢  1.137    [-0.5, 1.91, 2.16, -1.79, 0.14]  ⎥
⎢                                            ⎥
⎢ 0.7173     [0.92, -2, -1.17, 0.97, 0.47]   ⎥
⎢                                            ⎥
⎢ 0.53857  [0.35, 1.51, -1.97, -3.85, 0.37]  ⎥
⎢                                            ⎥
⎣ 0.83091  [-1.34, -2.95, -0.49, 0.91, 0.23] ⎦

pol_map([3.2, -0.01, 0.87, -0.34, 0.02], 2000, 0.05)

pol_map([3.2, -0.01, 0.87, -0.34, 0.02], 2000, 0.05, 2)

pol_map([3.2, -0.01, 0.87, -0.34, 0.02], 2000, 0.05, 3)
```

We start iteration with $x_0 = 0.05$ and perform 2000 iterations. Only coefficient sets with a positive LE are listed. Please notice the `write`-command. You can follow the calculation progress in the status line at left bottom.

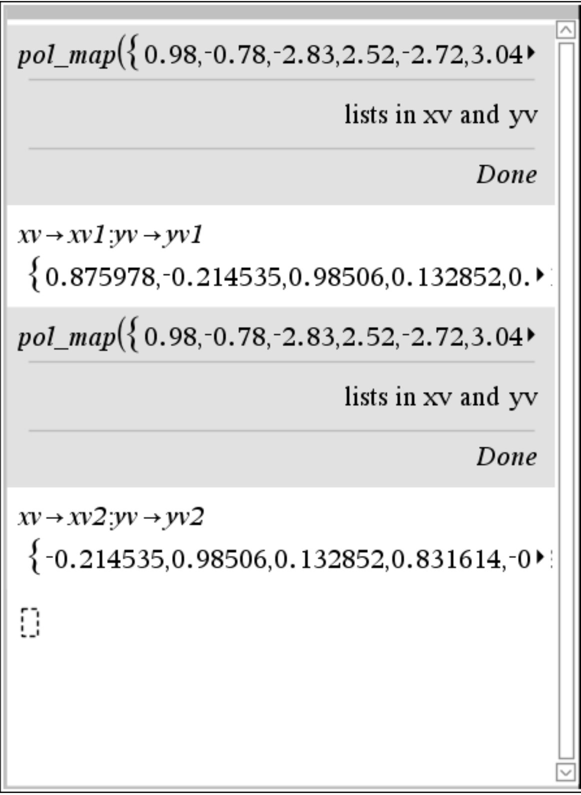
Let’s finish this first part of our tour by searching for polynomial attractors with TI-Nspire CAS.



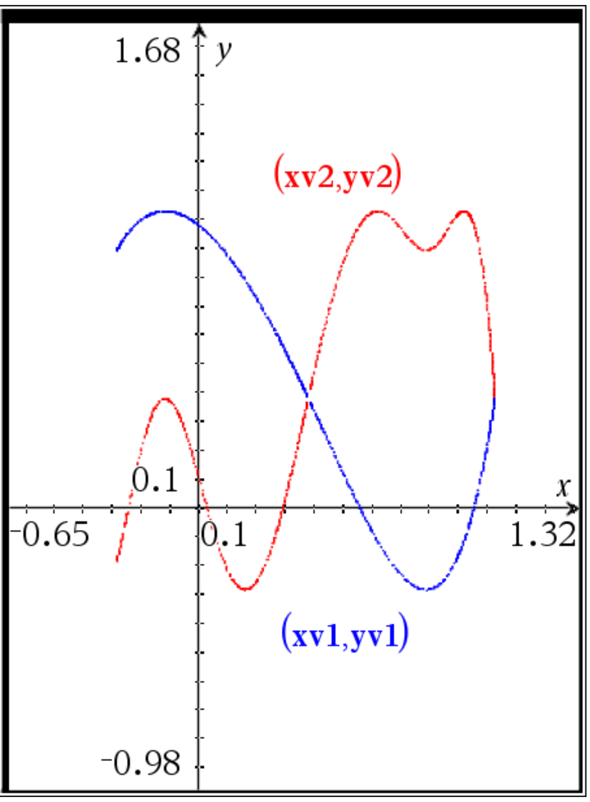
```
sap(3,100)
            [0.542776  "[]"  -0.65  -2.    0.87  4.63]
            [0.010943  "[]"  -0.79  -2.15  0.22  2.65]
                                                  Done
lyaptest({-0.65,-2,0.87,4.63},1000)
                                              0.542776
                                                  Done
sap(4,100)
       [0.735742  "[]"  0.14  -1.48  -3.08  4.33  -3.07]
                                                  Done
lyaptest({0.14,-1.48,-3.08,4.33,-3.07},1000)
                                              0.735742
                                                  Done
sap(5,100)
                                    no attractor found
                                                  Done
sap(5,500)
  [0.673171  "[]"   0.98  -0.78  -2.83   2.52  -2.72   3.04]
  [0.247262  "[]"  -0.45   1.87   1.11  -2.89  -2.52  -1.04]
```

```
sap                                                    0/18
Define sap(o,n)=
Prgm
Local tr,vals,xn,x0,l,f,f1,cf,i:{[]}→vals
For tr,1,n
  seq((round(100·(10·rand(1)−5),0))/100[1],k,1,o+1)→cf
  Σ(i=1,o+1)(cf[i]·x^(i−1))→f: d/dx(f)→f1
   iterates\iterate(f,x,0.05,12−2·o)→x0: 0→l
   For i,1,1000
     If |x0|>100 Then
       Exit
     EndIf
  lim(x→x0)(f)→xn: l+log_2(|lim(x→xn)(f1)|)→l: xn→x0
   EndFor
   If l>0 and |x0|<100 Then
   augment(vals,augment({l/1000},augment({"[]"},cf)))→vals
EndIf: EndFor
If dim(vals)>0 Then
Disp list▸mat(vals,o+3)
Else
Disp "no attractor found": EndIf
EndPrgm
```

It needs some – not too difficult – tricks to obtain an output similar to the DERIVE output. Finally, we convert the resulting list into a matrix with the Les in the first column. The rest of the rows present the coefficients in the same order as before.



```
pol_map({0.98,-0.78,-2.83,2.52,-2.72,3.04▸
                                lists in xv and yv
                                              Done
xv→xv1:yv→yv1
 {0.875978,-0.214535,0.98506,0.132852,0.▸
pol_map({0.98,-0.78,-2.83,2.52,-2.72,3.04▸
                                lists in xv and yv
                                              Done
xv→xv2:yv→yv2
 {-0.214535,0.98506,0.132852,0.831614,-0▸
[]
```

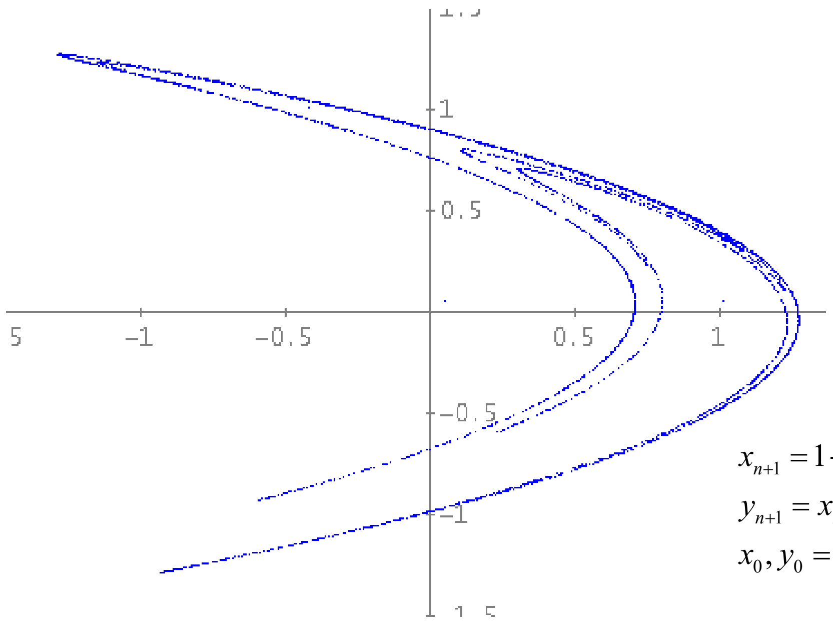


The real exciting adventure will start in the next part of the tour. Promised!!

6 A famous two-dimensional quadratic mapping

Let’s have a look to Julien C. Sprott. The attractor I have in mind is hidden behind his code “EWM?MPMMWMMMM”.

```
decode(EWM?MPMMWMMMM) = [1, 0, -1.4, 0, 0.3, 0, 0, 1, 0, 0, 0, 0]

                    qmap_2(EWM?MPMMWMMMM, 4000)
```

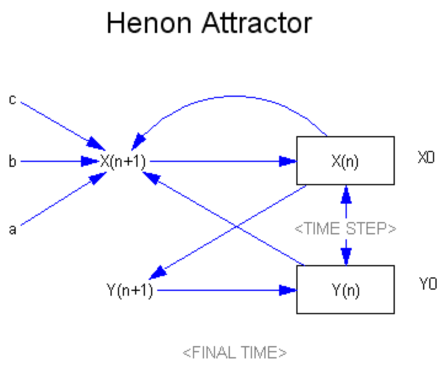


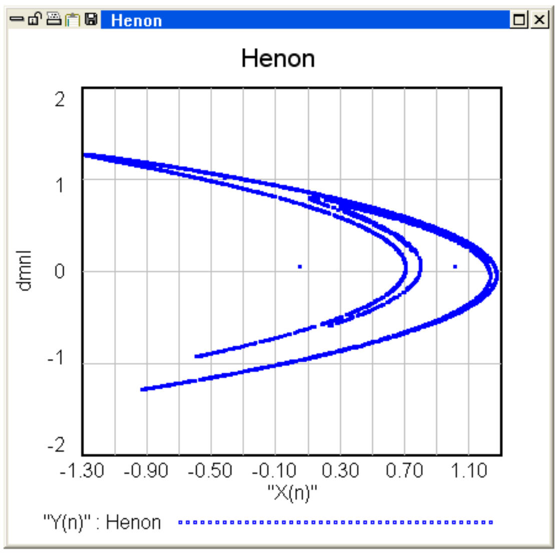
$$x_{n+1} = 1 - 1.4 x_n^{\;2} + 0.3 y_n$$
$$y_{n+1} = x_n$$
$$x_0, y_0 = 0.05$$

You will surely recognize the famous Hénon-Attractor [3, 5, 7].

Please allow a side step. I use a program for modelling dynamic systems -VENSIM PLE [4]- ([http://vensim.com/vensim-software/)](http://vensim.com/vensim-software/) for plotting the Hénon-attractor:

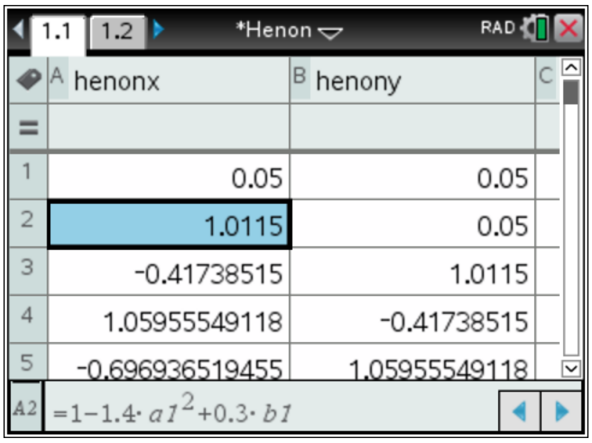
$$x_{n+1} = 1 + a\,x_n^{\;2} + b\,y_n$$
$$y_{n+1} = x_n$$
$$x_0, y_0 \;\text{ given}$$





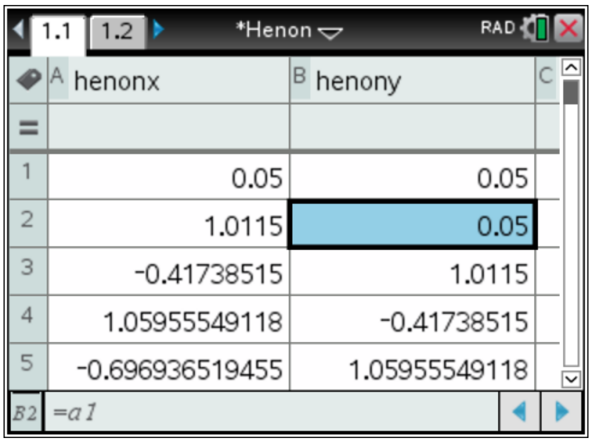
Hénon-Attractor generated with VENSIM PLE

The easiest way to create the Hénon-attractor on a TI-Nspire screen is defining the two recursive equations in the spreadsheet application as shown below.



	A henonx	B henony
=		
1	0.05	0.05
2	1.0115	0.05
3	-0.41738515	1.0115
4	1.05955549118	-0.41738515
5	-0.696936519455	1.05955549118

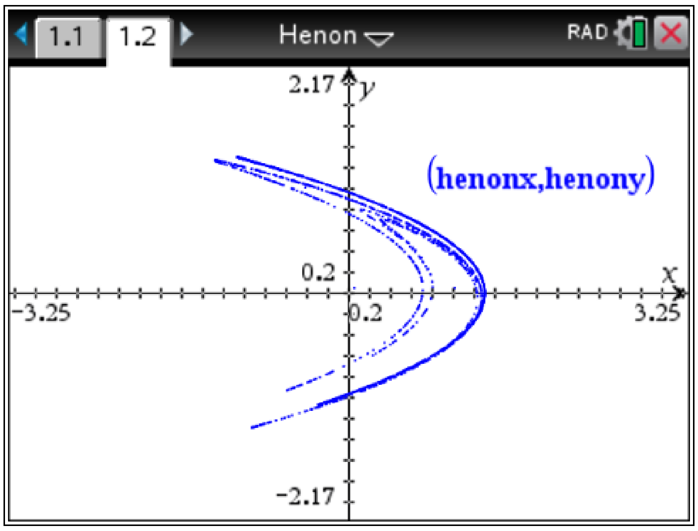
A2 $=1-1.4\cdot a1^2+0.3\cdot b1$



	A henonx	B henony
=		
1	0.05	0.05
2	1.0115	0.05
3	-0.41738515	1.0115
4	1.05955549118	-0.41738515
5	-0.696936519455	1.05955549118

B2 $=a1$

We can copy down 2500 rows maximum (with the software) and then plot the scatter diagram. We recognize that the graph looks a little bit different (maybe some inaccuracy?).

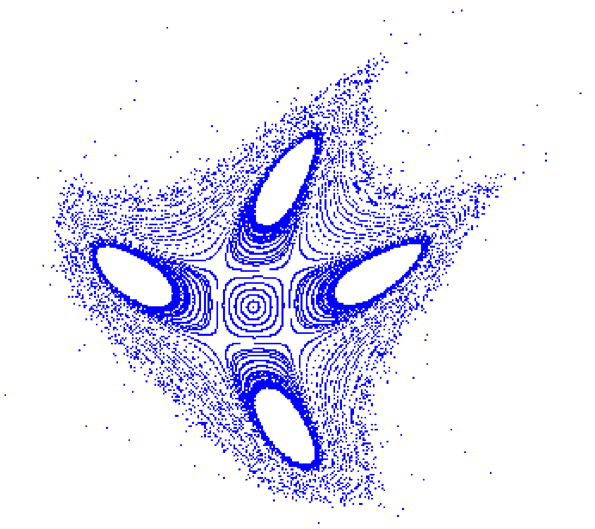


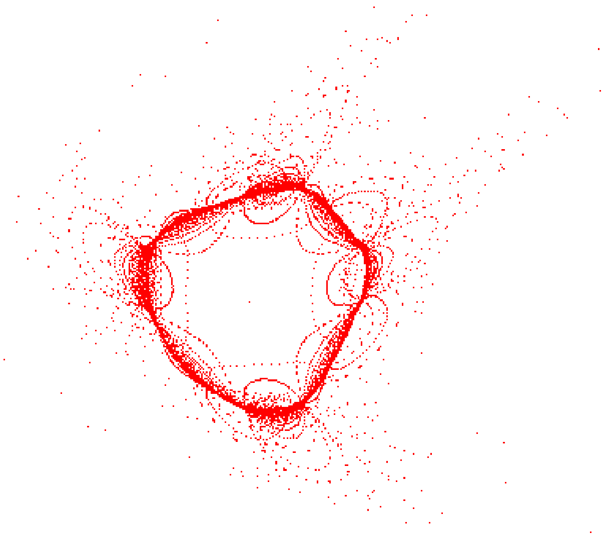
Before showing other quadratic mappings, I'd like to present other Hénon-systems.

```
henon(start, α, n, list := [], i, xn, yn, xo, yo) :=
  Prog
    i := 1
    [xn := start↓1, yn := start↓2]
    Loop
      If i > n
         RETURN list
      list := APPEND(list, [[xn, yn]])
      [xo := xn, yo := yn]
      xn := xo·COS(α) - (yo - xo^2)·SIN(α)
      yn := xo·SIN(α) + (yo - xo^2)·COS(α)
      If ABS(xn) > 10000
         xn := 0
      If ABS(yn) > 10000
         yn := 0
      i :+ 1
```

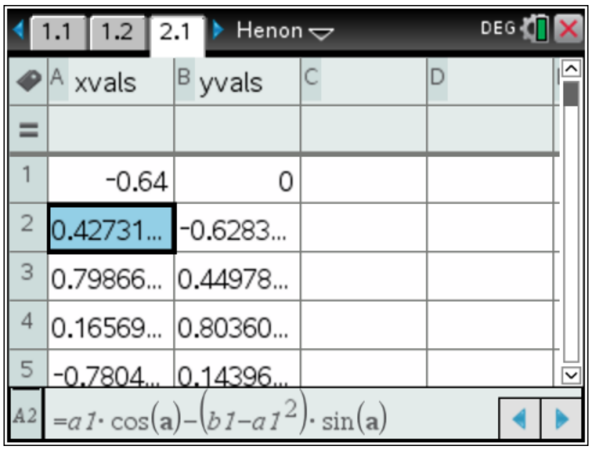
These are not attractors but variations of the initial point result in exciting graphs of chaotic distributed points in the plane forming unexpected patters.

```
VECTOR(henon([k, 0], 90.57·1°, 400), k, -0.7, 0.7, 0.025)

VECTOR(henon([0.5, 0], k·1°, 400), k, 75, 95, 0.5)
```



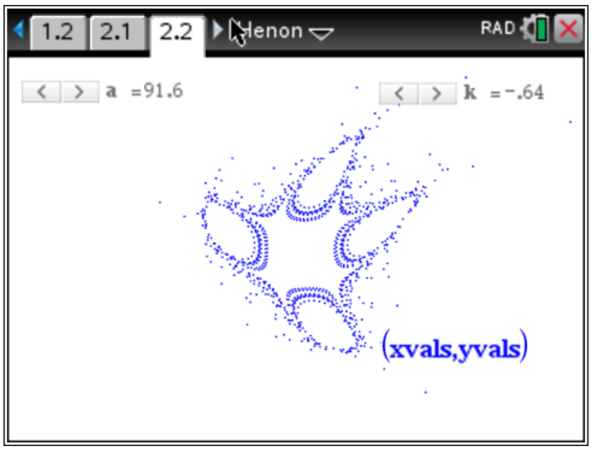


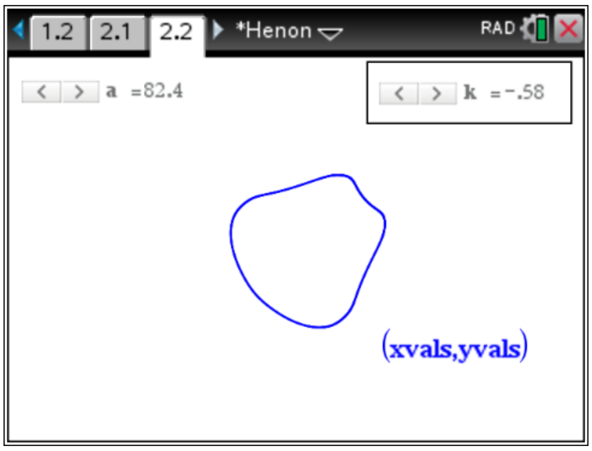
We can vary the initial point (left) or the angle (right). Unfortunately, we cannot produce the families of scatter diagrams with the TI-Nspire-environment. But what we can do is varying the parameters by a slider and observe how the "members of the family" are changing.

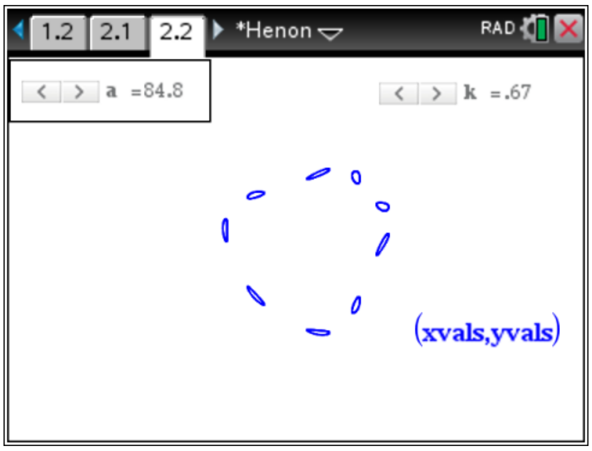


	A xvals	B yvals	C	D
=				
1	-0.64	0		
2	0.42731...	-0.6283...		
3	0.79866...	0.44978...		
4	0.16569...	0.80360...		
5	-0.7804...	0.14396...		

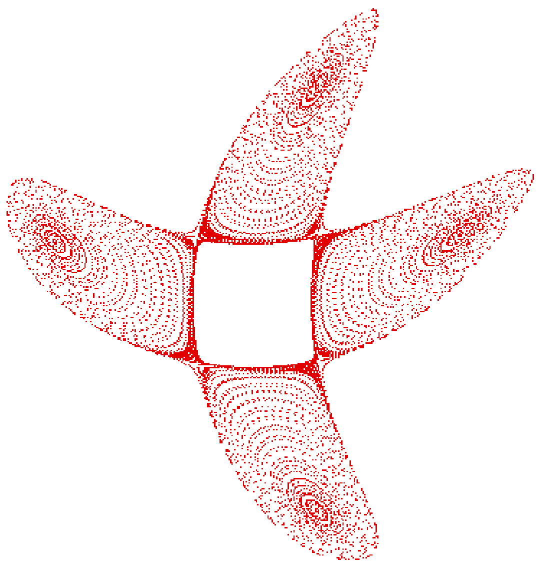
A2 $=a1\cdot\cos(a)-(b1-a1^2)\cdot\sin(a)$

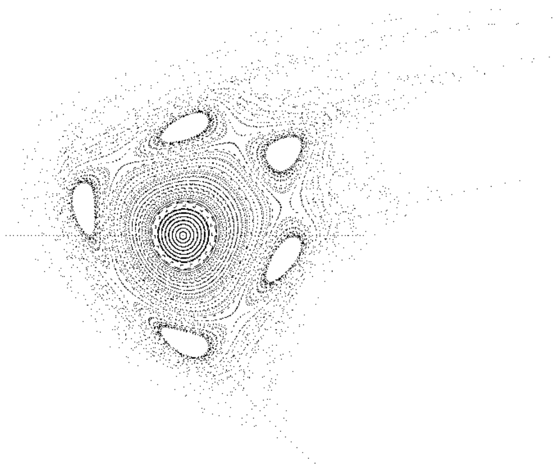


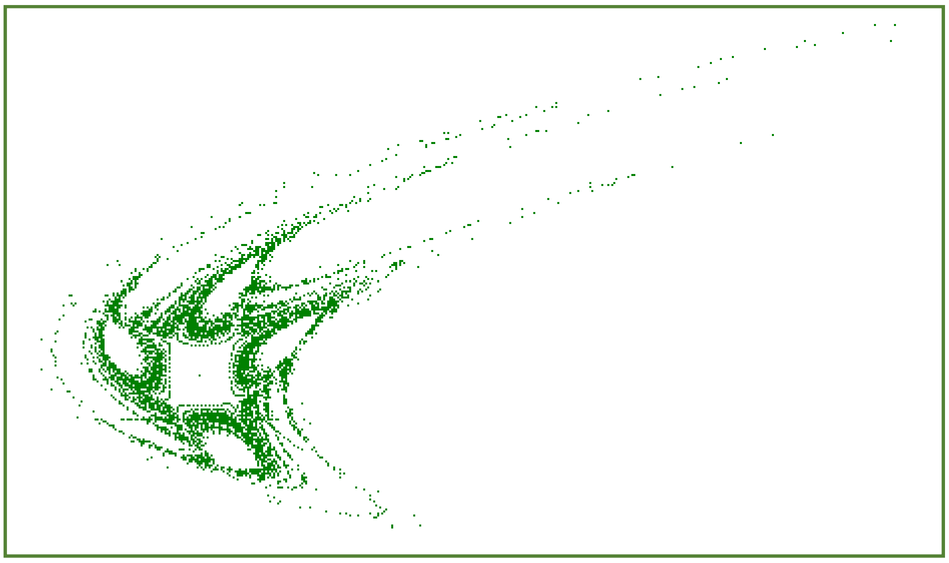


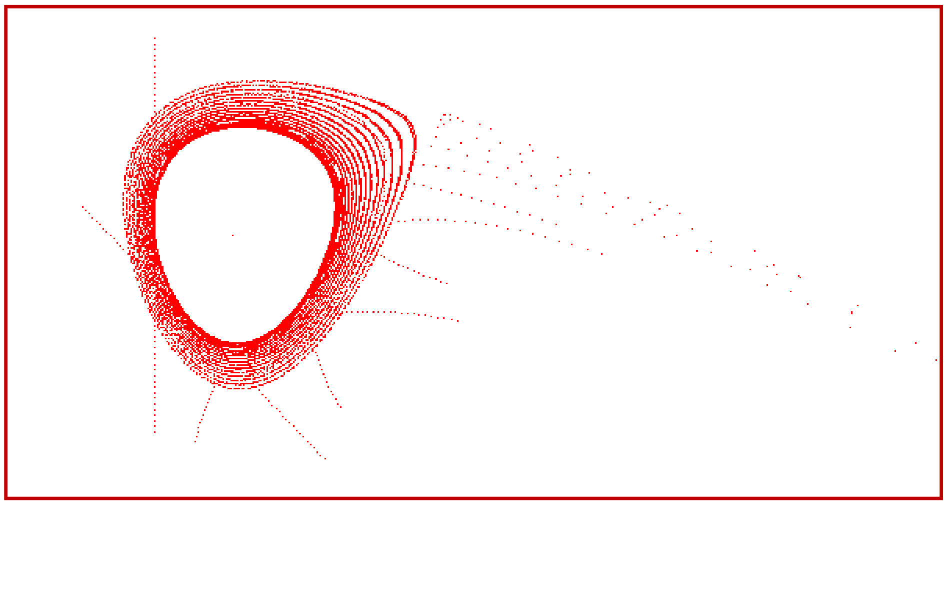


More nice *DERIVE*-generated Hénon-families:



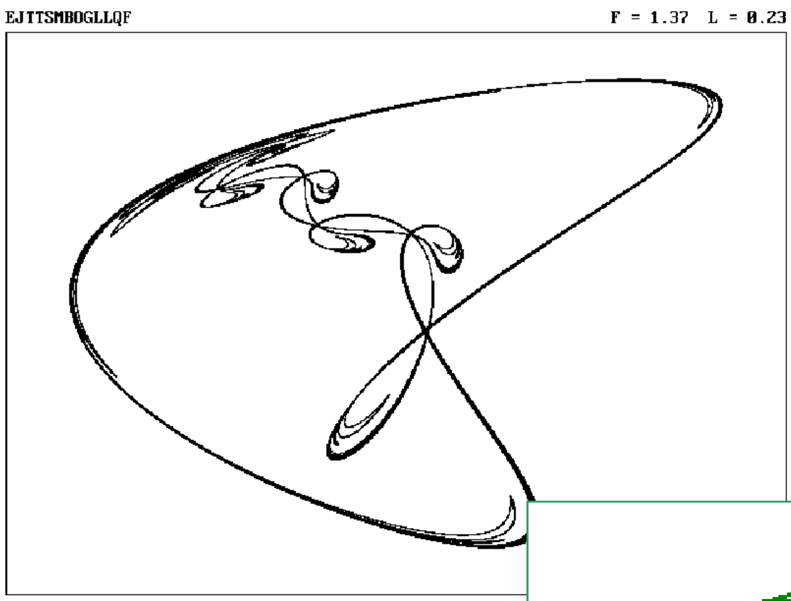


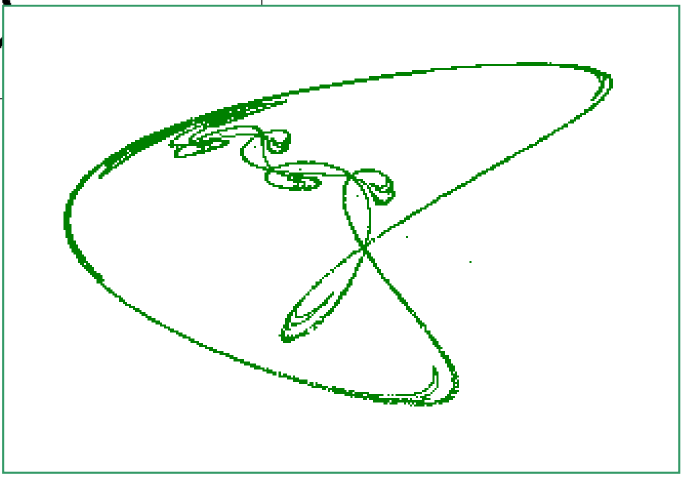




7 Generalized Polynomial Maps

The next program (*DERIVE* and TI-Nspire) is to plot polynomial maps. As a reference for my program `map` I used one of the many graphs in Sprott's book.





`map(EJTTSMBOGLLQF, 20000)`

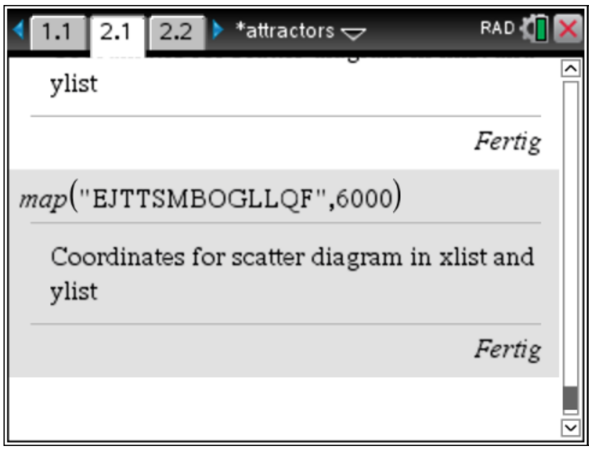
My graph produced with my program `map` looks quite the same (20 000 points!).

Sprott's program returns the Lyapunov-exponent and the fractal dimension (which I didn't include in my investigations).

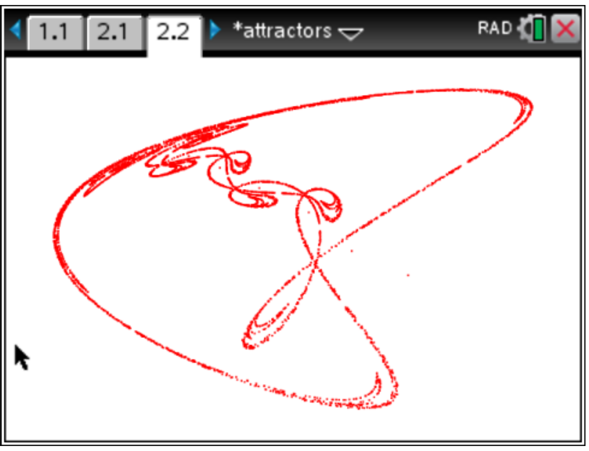
His L =0.23 is confirmed by my `ljap_e("EJTTSMBOGLLQF",1000)`.

`ljap_e(EJTTSMBOGLLQF, 1000) = 0.24121`

TI_Nspire's *map*() cannon plot 20000 points, but as you can see 6000 points can do it also.



```
ylist
                                   Fertig
map("EJTTSMBOGLLQF",6000)
Coordinates for scatter diagram in xlist and
ylist
                                   Fertig
```



It is possible to produce the plots using the spreadsheet app, too. But here we can only produce 2500 points maximum – and it might be that we would miss some accuracy. I prefer the program.

Both map-programs understand Sprott's codes and lists of coefficients as well.

Sprott lists all his maps in an appendix. The two-dimensional cubic (case F) map needs 20 coefficients.

Case F: D = 2, O = 3, M = 20

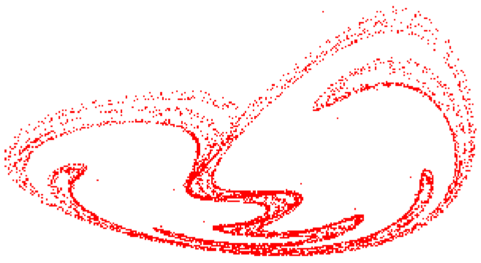
$$X = a_1 + a_2X + a_3X^2 + a_4X^3 + a_5X^2Y + a_6XY + a_7XY^2 + a_8Y + a_9Y^2 + a_{10}Y^3$$

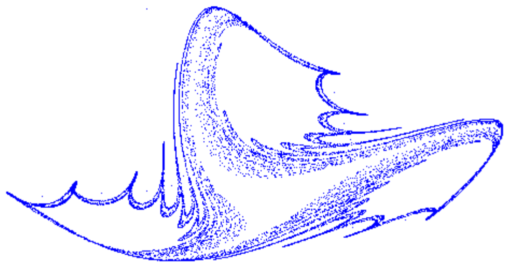
$$Y = a_{11} + a_{12}X + a_{13}X^2 + a_{14}X^3 + a_{15}X^2Y + a_{16}XY + a_{17}XY^2 + a_{18}Y + a_{19}Y^2 + a_{20}Y^3$$

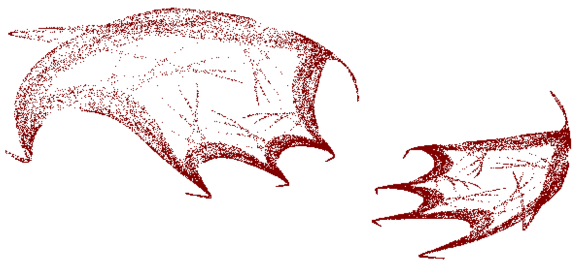
The list of coefficients reads in the order $[a_1, a_2, \ldots, a_{20}]$.

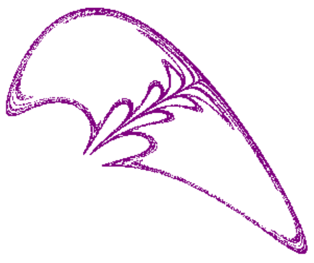
Some examples from Sprott's collection

Cubic maps



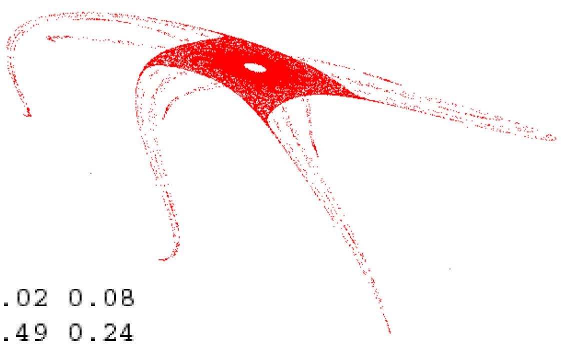






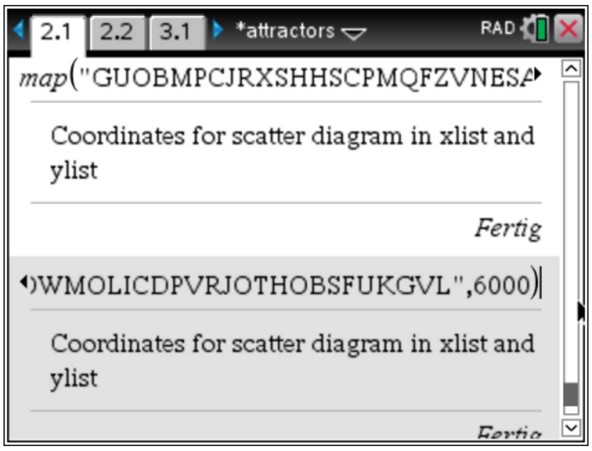
Quartic and Quintic maps

30 000 points with DERIVE

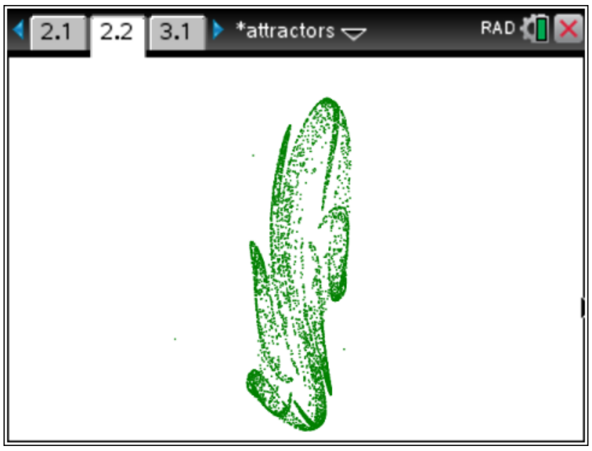


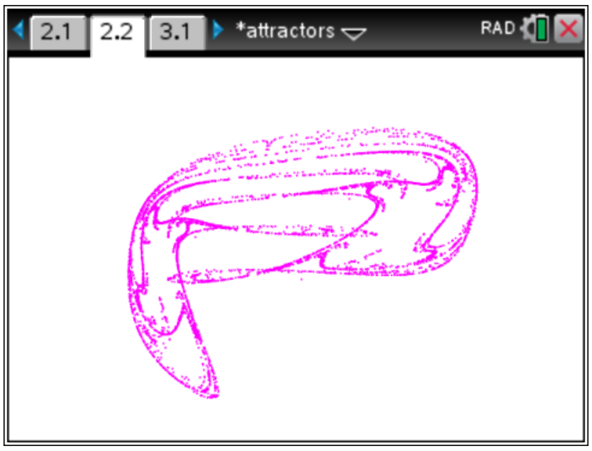
```
GUMMMMMMMMMMMMMWEODFMMMMMMMMMMMM 1.02 0.08
GUOBMPCJRXSHHSCPMQFZVNESALEKOHY 1.49 0.24
GXEMONYFKDJMDTPNSLGHQLHOOTOQBUN 1.20 0.18
HCJBKUPMMMMMMMMMMMWMMMMICMMMMMMMMMMMMMMMMMMM 1.39 0.07
HHANQRENHONYATQYPTNXKNMNQEGDWKYPNSMMMODAOBC 1.13 0.01
HIFZLMPJUBERQBKLRRDOWMOLICDPVRJOTHOBSFUKGVL 1.39 0.24
```

See on the next page the 6000 points plot of the second and the last code from above (GUOBM … and HIFZ …)



```
map("GUOBMPCJRXSHHSCPMQFZVNESA▸
      Coordinates for scatter diagram in xlist and
      ylist
                                          Fertig
◂)WMOLICDPVRJOTHOBSFUKGVL",6000)
      Coordinates for scatter diagram in xlist and
      ylist
                                          Fertig
```





You can see that we can obtain satisfying results with TI-Nspire, too.

I don't want to print the code of the `map`-program – it needs too much space. Instead of this I'll present the next program which is much more demanding and interesting.

7 Exciting Searches for "attractive" Polynomial Maps

It is great to try the many promising codes and then admire the fantastic figures appearing on the screen. Then it is clear that one has the strong desire to find his own attractors. Just taking any numbers as coefficients is not very promising. So let's apply our `sa(order,number of tries)` – program.

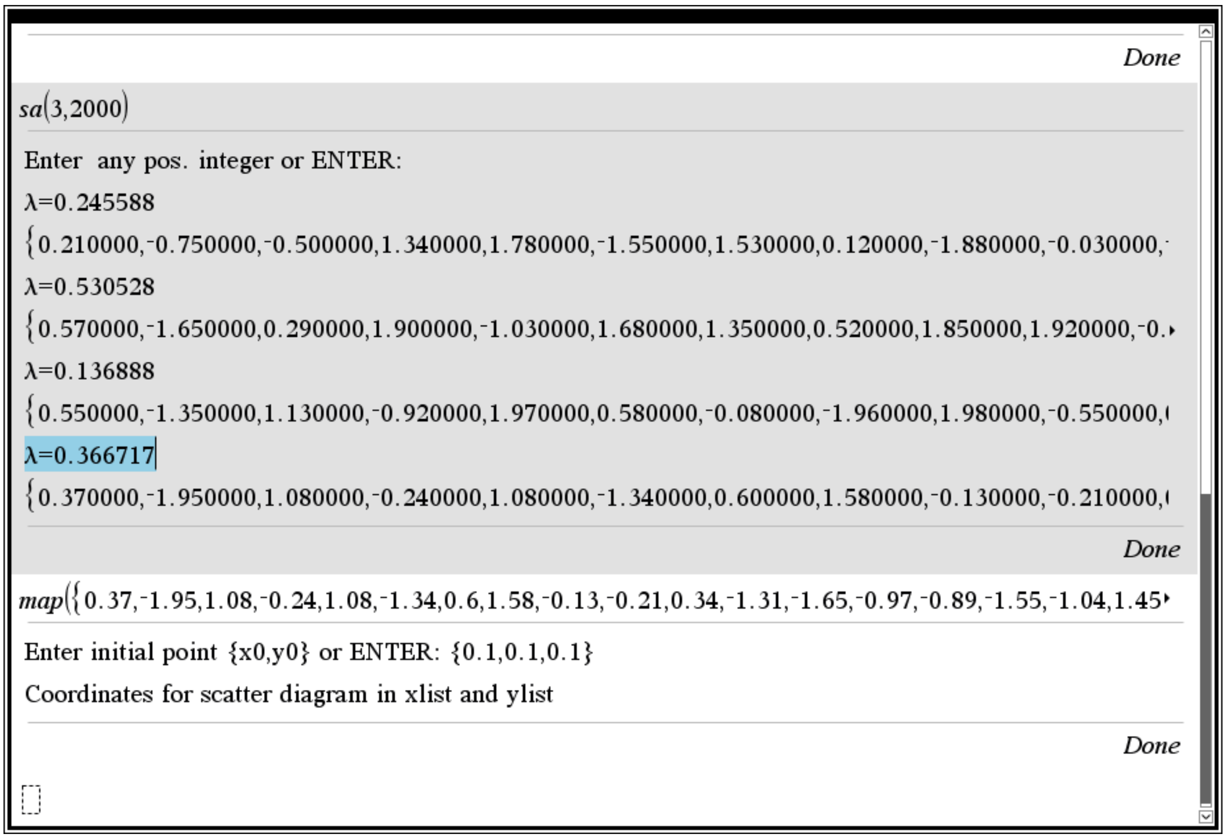
On the next page you can find a demonstration how it works:

I present the TI-Nspire procedure followed by one *DERIVE* search.

With the Nspire it is recommended to enter a randseed number (at least for the first run) in order to start new investigations. With only ENTER the next random number will be generated based on the built-in randseed.

The default initial point for searching and plotting as well is [0.05,0.05]. If you want to choose another point (be careful!) then you can add this point as third argument in the *DERIVE* program, but not in TI-Nspire. (Here you will be asked to enter another initial point than the default one!)

The last important difference lies in the fact that you don't need any work around for plotting the scatter diagram in *DERIVE* and not to forget, we can generate and plot much more points!

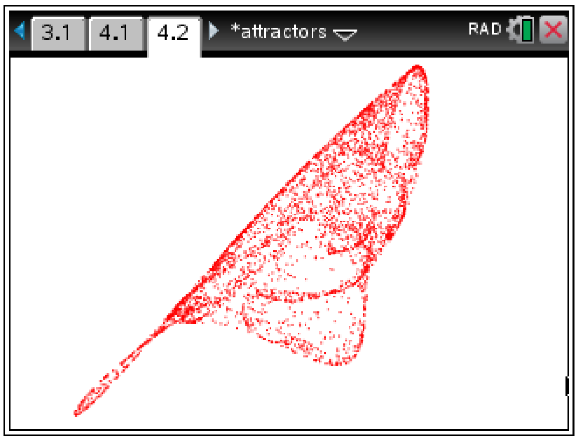


```
Done
sa(3,2000)
Enter  any pos. integer or ENTER:
λ=0.245588
{0.210000,-0.750000,-0.500000,1.340000,1.780000,-1.550000,1.530000,0.120000,-1.880000,-0.030000,
λ=0.530528
{0.570000,-1.650000,0.290000,1.900000,-1.030000,1.680000,1.350000,0.520000,1.850000,1.920000,-0.▸
λ=0.136888
{0.550000,-1.350000,1.130000,-0.920000,1.970000,0.580000,-0.080000,-1.960000,1.980000,-0.550000,
λ=0.366717
{0.370000,-1.950000,1.080000,-0.240000,1.080000,-1.340000,0.600000,1.580000,-0.130000,-0.210000,
Done
map({0.37,-1.95,1.08,-0.24,1.08,-1.34,0.6,1.58,-0.13,-0.21,0.34,-1.31,-1.65,-0.97,-0.89,-1.55,-1.04,1.45▸
Enter initial point {x0,y0} or ENTER: {0.1,0.1}
Coordinates for scatter diagram in xlist and ylist
Done
```

It is comfortable to copy and paste the list of coefficients into the map(list, number of points) call.

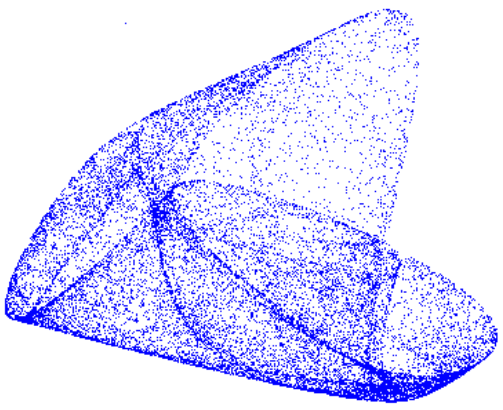
This is the final version showing only attractors with positive Lyapunov-exponents – they are the "attractive ones".

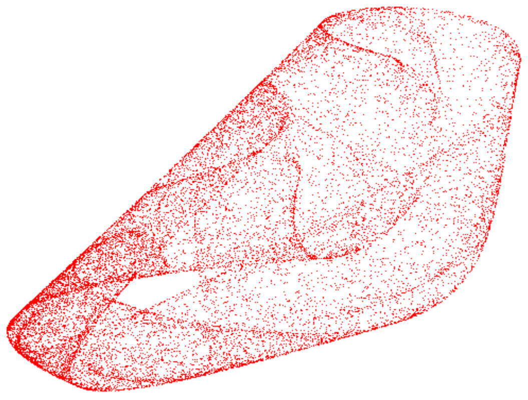
See the Nspire Calculator screen (software) and the respective polynomial map of order 3. The positive LE indicate "strange attractors".

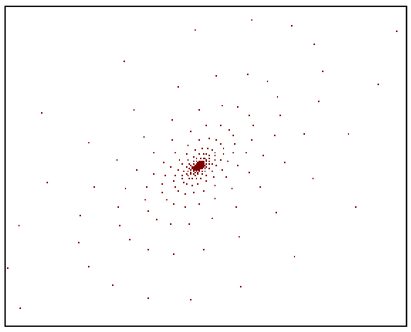


Let's do the same with *DERIVE*: I was lucky and found two "suspicious" Lyapunov-exponents and plotted both quadratic maps (20000 iteration points).

```
sa(2, 200)

⎡  0.42153    [0.4, -2.59, 1.37, -3.28, -2.65, -4.31, -0.24, -1.52, 3.13, 0.07, 1.61, 1.67] ⎤
⎢  0.24404     [-0.02, 0.72, 0.2, -1.42, 0.71, 3.33, -0.25, 2.4, 0.56, 0.21, -0.62, -0.77]  ⎥
⎢ -0.073991  [1.57, -1.46, 1.19, 0.49, -1.36, -0.13, 1.49, -0.83, 0.14, 0.82, -0.28, -0.27] ⎥
⎢ -0.026602   [-0.37, 1.2, 4.09, -4.9, -1.79, -1.98, -0.45, -1.15, 0.05, -0.42, 1.39, 1.84] ⎥
⎢ -0.038797   [-0.81, 0.27, 1.69, -0.03, 0.58, -0.54, -0.3, 0.38, 1.02, 0.61, -0.39, 0.65]  ⎥
⎢ -0.94608    [0.13, 1.19, 0.28, 1.11, -1.21, -1.07, 0.41, 1.52, -0.51, 1.34, -1.84, 0.37]  ⎥
⎣ -0.28225      [0.21, 0.9, 0.4, 0.56, 0.55, 0.59, -0.57, -1.66, 0.18, 1.25, -1.18, 0.07]   ⎦

map([0.4, -2.59, 1.37, -3.28, -2.65, -4.31, -0.24, -1.52, 3.13, 0.07, 1.61, 1.67], 2000)

map([-0.02, 0.72, 0.2, -1.42, 0.71, 3.33, -0.25, 2.4, 0.56, 0.21, -0.62, -0.77], 2000)
```







Nice graphs, aren't they?

I plot also 10000 points of one list having a negative LE λ = -0.0266. As you can see the graph is not very appealing – it is a point attractor.

We can search for attractors generated by two-dimensional polynomial systems up to order `o` = 4. `cf` is the list of the randomly generated coefficients from (-2, +2) and its number `cfs` depends on the order `o` of the system. The distance to the separated points which is necessary for calculating the Lyapunov exponent is fixed with 10^{-6}, and the initial point is fixed at [0.05, 0.05] – but it can be changed.

The result is a list of the polynomial coefficients together with the respective Lyapunov exponents. All other combinations of numbers are tries leading to unbounded iterations.

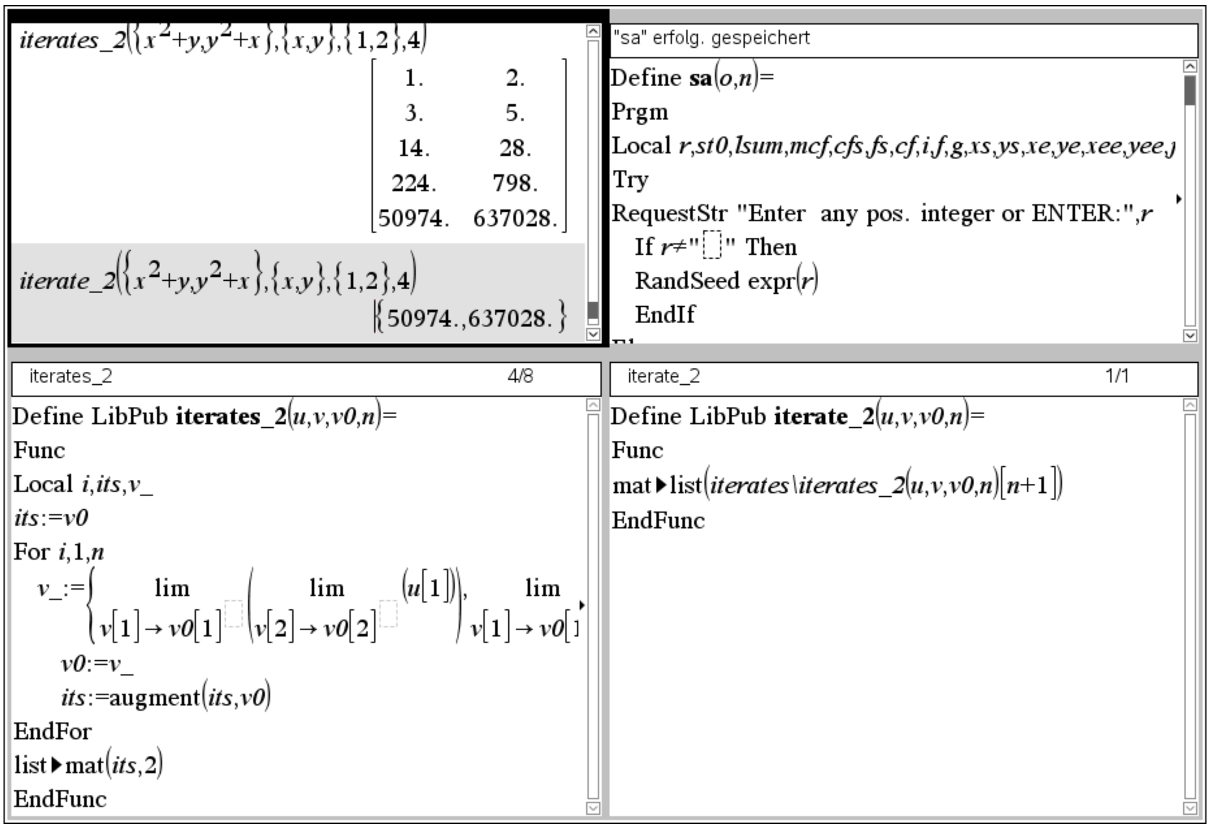
This is the moment when discovering begins and the experiment becomes exciting. Make some thousand tries and wait for the results. Feel happy if there are positive LEs appearing, because they promise STRANGE ATTRACTORS. Then plot some thousand iteration points, lean back and enjoy the moment when it is very likely that you are the very first person seeing this object appearing.

For our "search attractor"-program we will use ITERATE as we did earlier. As you might know, unfortunately ITERATE(S) is not implemented in the TI-Nspire tool box. So, I produced iterates_2() and iterate_2() for TI-NspireCAS. Please compare DERIVE's ITERATE(S) and the TI-Nspire's one.

$$\mathrm{ITERATES}\left(\left[x^2 + y,\ y^2 + x\right],\ [x,\ y],\ [1,\ 2],\ 4\right)$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 5 \\ 14 & 28 \\ 224 & 798 \\ 50974 & 637028 \end{bmatrix}$$

$$\mathrm{ITERATE}\left(\left[x^2 + y,\ y^2 + x\right],\ [x,\ y],\ [1,\ 2],\ 4\right) = [50974,\ 637028]$$



```
iterates_2({x^2+y,y^2+x},{x,y},{1,2},4)
                              [1.      2.    ]
                              [3.      5.    ]
                              [14.     28.   ]
                              [224.    798.  ]
                              [50974.  637028.]
iterate_2({x^2+y,y^2+x},{x,y},{1,2},4)
                              {50974.,637028.}
```

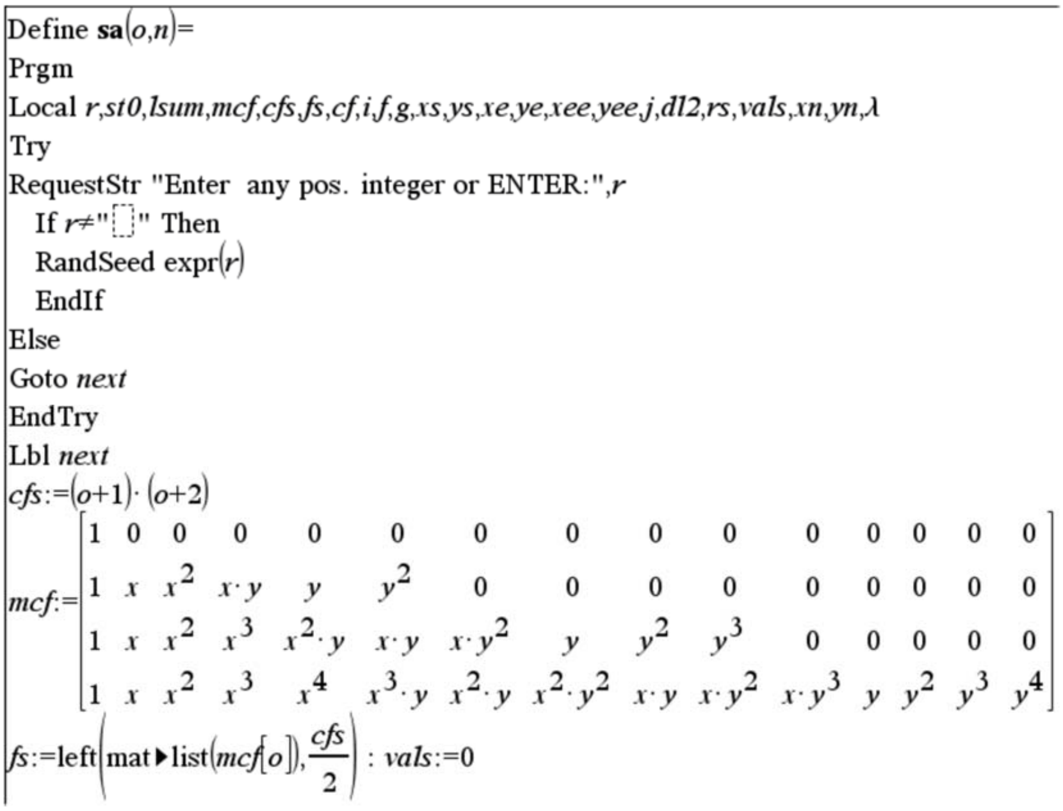
```
"sa" erfolg. gespeichert
Define sa(o,n)=
Prgm
Local r,st0,lsum,mcf,cfs,fs,cf,i,f,g,xs,ys,xe,ye,xee,yee,j
Try
RequestStr "Enter  any pos. integer or ENTER:",r
  If r≠"" Then
  RandSeed expr(r)
  EndIf
```

```
iterates_2                                   4/8
Define LibPub iterates_2(u,v,v0,n)=
Func
Local i,its,v_
its:=v0
For i,1,n
  v_:={lim(v[1]→v0[1]) lim(v[2]→v0[2]) (u[1]), lim(v[1]→v0[1]
    v0:=v_
    its:=augment(its,v0)
EndFor
list▶mat(its,2)
EndFunc
```

```
iterate_2                                    1/1
Define LibPub iterate_2(u,v,v0,n)=
Func
mat▶list(iterates\iterates_2(u,v,v0,n)[n+1])
EndFunc
```

I put the “iterates” into a library in order to have it available at any time.

I reprint sa(o,n) for TI-Nspire. It is “inspired” by my *DERIVE*-program which was “inspired” by a huge BASIC program provided by Julien C. Sprott. The *DERIVE* version of the map-program will not be presented here. You are invited to inspect (and use!) it (can be found in attr.zip).



```
Define sa(o,n)=
Prgm
Local r,st0,lsum,mcf,cfs,fs,cf,i,f,g,xs,ys,xe,ye,xee,yee,j,dl2,rs,vals,xn,yn,λ
Try
RequestStr "Enter  any pos. integer or ENTER:",r
  If r≠"" Then
  RandSeed expr(r)
  EndIf
Else
Goto next
EndTry
Lbl next
cfs:=(o+1)·(o+2)
```

$$mcf:=\begin{bmatrix}1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 1 & x & x^2 & x\cdot y & y & y^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 1 & x & x^2 & x^3 & x^2\cdot y & x\cdot y & x\cdot y^2 & y & y^2 & y^3 & 0 & 0 & 0 & 0 & 0\\ 1 & x & x^2 & x^3 & x^4 & x^3\cdot y & x^2\cdot y & x^2\cdot y^2 & x\cdot y & x\cdot y^2 & x\cdot y^3 & y & y^2 & y^3 & y^4\end{bmatrix}$$

$fs:=\text{left}\left(\text{mat▶list}(mcf[o]),\frac{cfs}{2}\right)$: $vals:=0$

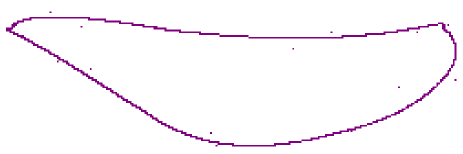
```
For $i$,1,$n$
$cf:=\frac{\text{floor}(100\cdot(4\cdot\text{rand}(cfs)-2))}{100}$
$f:=\text{sum}\left(\text{left}\left(cf,\frac{cfs}{2}\right)\cdot fs\right)$: $g:=\text{sum}\left(\text{right}\left(cf,\frac{cfs}{2}\right)\cdot fs\right)$
$lsum:=0$
$st0:=iterate\_2(\{f,g\},\{x,y\},\{0.05,0.05\},5)$
$\square$
$xs:=st0[1]:ys:=st0[2]$
$xe:=xs+1.\text{E}^{-}6:ye:=ys$
For $j$,1,2000
    If $|xs|>10$ or $|ys|>10$ Then
    Goto $nexttry$
    EndIf
    $xn:=\lim_{y\to ys}\left(\lim_{x\to xs}(f)\right):yn:=\lim_{y\to ys}\left(\lim_{x\to xs}(g)\right)$
    $xee:=\lim_{y\to ye}\left(\lim_{x\to xe}(f)\right):yee:=\lim_{y\to ye}\left(\lim_{x\to xe}(g)\right)$
    $xe:=xee:ye:=yee$
    $dl2:=(xn-xe)^2+(yn-ye)^2$
```

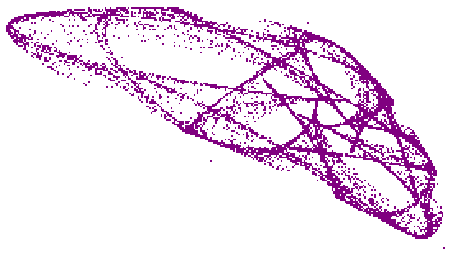
```
    $rs:=\frac{1}{10^{6}\cdot\sqrt{dl2}}$
    $xs:=xn:ys:=yn$
    $xe:=xs+rs\cdot(xe-xs):ye:=ys+rs\cdot(ye-ys)$
    $lsum:=lsum+\ln(10^{12}\cdot dl2)$
EndFor
$\lambda:=\frac{\frac{1}{2\cdot\ln(2)}\cdot lsum}{j}$
If $\lambda>0$ Then
   $vals:=1$
   Disp "λ="&string($\lambda$)
   Disp $cf$
EndIf
Lbl $nexttry$
EndFor
If $vals=0$ Then
  Disp "no attractor found"
EndIf
EndPrgm
```

Let’s have another run searching for cubic maps (I changed the program, in order to give only attractors with positive Lyapunov-exponents):

```
sa(3, 1000)

⎡ 0.087146   [-0.01, 1.61, 0.1, -1.93, 1.93, 1.31, -0.11, 0.34, -0.82, -0.08, 0.38, -1.
⎢  0.20434     [0.49, -0.76, -1.39, 1.74, 1.74, -1.18, 0.24, 0.67, 1.31, 0.57, 0.46, -1
⎢ 0.0014233  [0.27, 1.04, -0.77, -0.69, 1.52, -0.07, -0.7, -0.27, -1.05, -0.93, 0.39, -
⎣ 0.041903     [-0.66, 0.07, 0.75, 0.64, -1.32, 0.83, -2, -0.24, 1.74, 1.47, 0.17, -1.
```

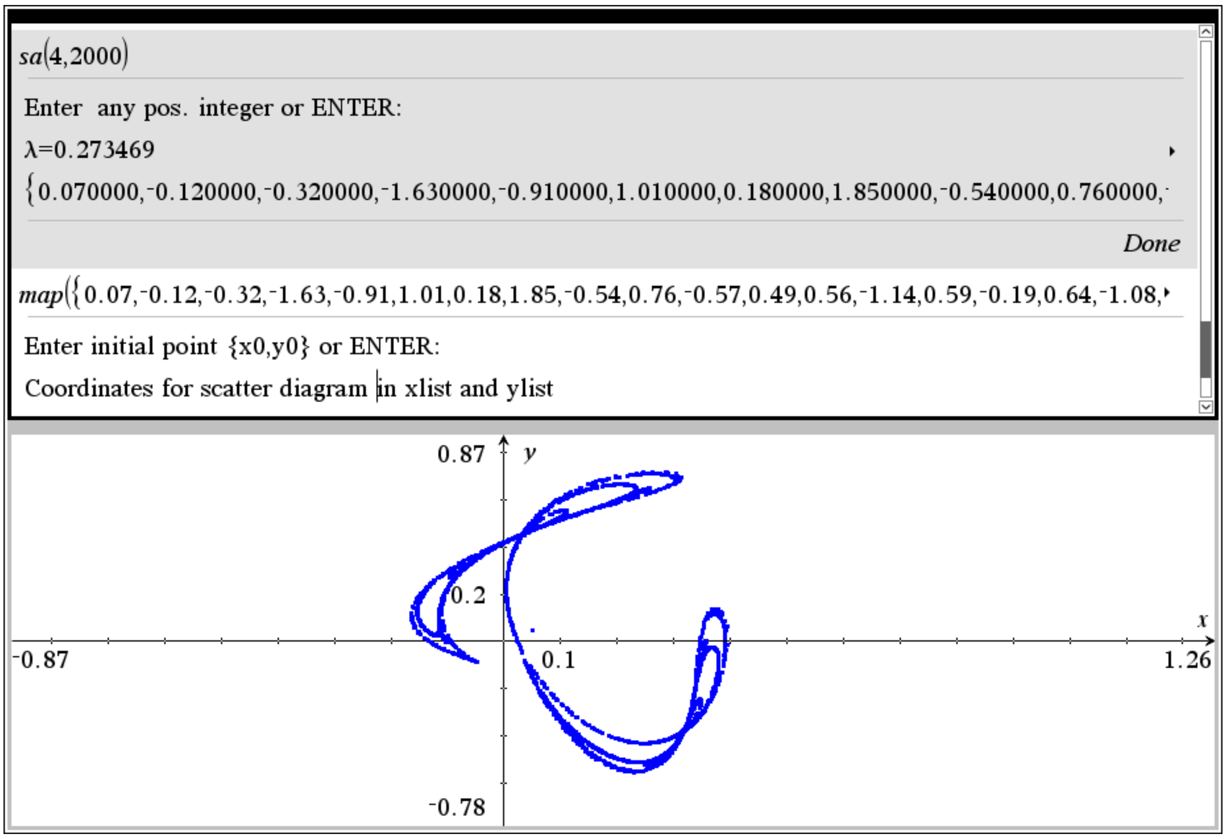
I plot one of the maps with a positive LE (right) and then the one map with its LE very close to zero. LE < 0 indicates a point attractor and LE = 0 announces a curve attractor. The left graph is a confirmation of this.





Browsing in Sprott’s book one gets an impression about his tremendous and time-consuming work collecting all his wonderful attractors … and this is just the beginning of our real “Attractors’ Tour”.

Last search for now: We’ll find a quartic map with TI-Nspire and plot 4000 points of it.



```
sa(4,2000)
Enter  any pos. integer or ENTER:
λ=0.273469
{0.070000,-0.120000,-0.320000,-1.630000,-0.910000,1.010000,0.180000,1.850000,-0.540000,0.760000,
                                                                                          Done
map({0.07,-0.12,-0.32,-1.63,-0.91,1.01,0.18,1.85,-0.54,0.76,-0.57,0.49,0.56,-1.14,0.59,-0.19,0.64,-1.08,
Enter initial point {x0,y0} or ENTER:
Coordinates for scatter diagram in xlist and ylist
```

9 Two special polynomial maps (and attractors, of course)

Some special structures of the polynomial functions result in special forms of attractors. Sprott called two of them "*Symplectic Attractors*" and "*Plaid Attractors*".

Symplectic Attractors

$$x_{n+1} = a + b x_n + c x_n^2 + d x_n^3 + e x_n^4 + f x_n^5 \pm y_n$$
$$y_{n+1} = g \mp x_n$$

Plaid Attractors

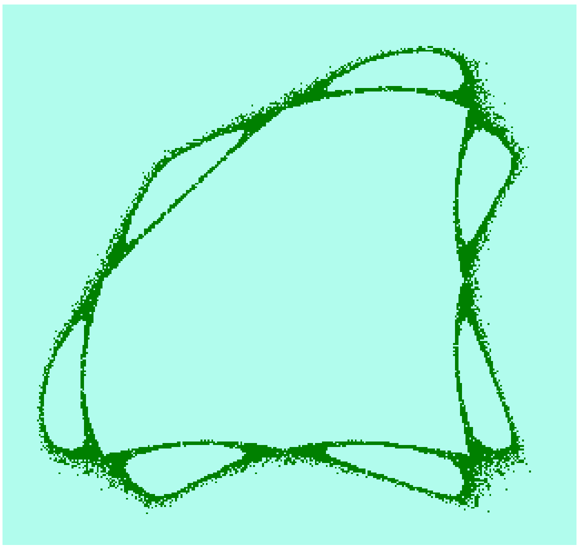
$$x_{n+1} = a + b y_n + c y_n^2 + d y_n^3 + e y_n^4$$
$$y_{n+1} = f + g x_n + h x_n^2 + i x_n^3 + k x_n^4$$

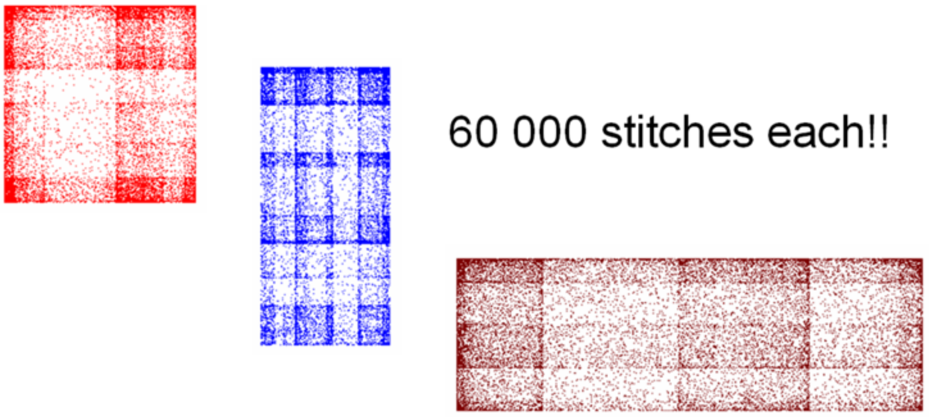
We will search for both types of attractors and then admire the results. (I changed the program to only return the coefficients giving a positive LE.)

```
sa_symp(3, 5000)

[  0.0107637     [-0.36, -0.8, 0.73, 0.42, 1, 0.67, -1]  ]
[ 0.00957944     [0.17, 1.55, -2.1, -2.88, -1, -0.04, 1] ]
[  0.0132139     [0.63, 0.71, -0.1, -0.95, -1, 0.53, 1]  ]
[  0.0117269     [-0.14, 0.07, 1.09, 0.24, 1, -1.15, -1] ]
[  0.0491747     [-3.64, -0.44, 0.2, 0.01, 1, 2.69, -1]  ]
```

```
sa_plaid(4, 10000)

[ 0.287029   [-0.42, 1.87, -0.88, -2.33, 3.78, -0.09, -0.39, 0.57, -2.65, -3.18] ]
[  0.28711    [0.5, 0.39, -2.01, -3.63, -2.2, -0.52, 1.63, -0.94, 4.06, -4.12]  ]
[ 0.250974    [0.05, -1.16, 3.49, -1.12, -2.39, 0.24, -3.9, 4.85, -2.81, -1.27] ]
[ 0.294624    [0.16, 0.55, -1.72, -3.12, 1.47, -0.25, 2.62, 2.01, -2.15, -0.71] ]
[ 0.0995453   [-0.49, -0.38, 1.45, -2.04, 1.19, 0.05, -3.2, -1.26, 2.98, 3.03]  ]
```

```
map_symp([-3.64, -0.44, 0.2, 0.01, 1, 2.69, -1], 20000)
```

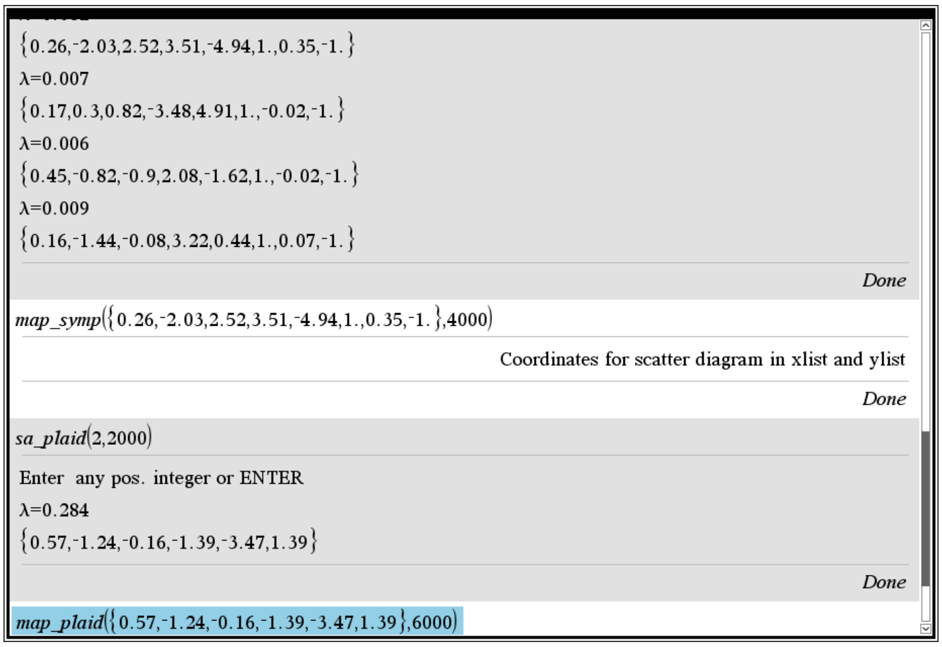




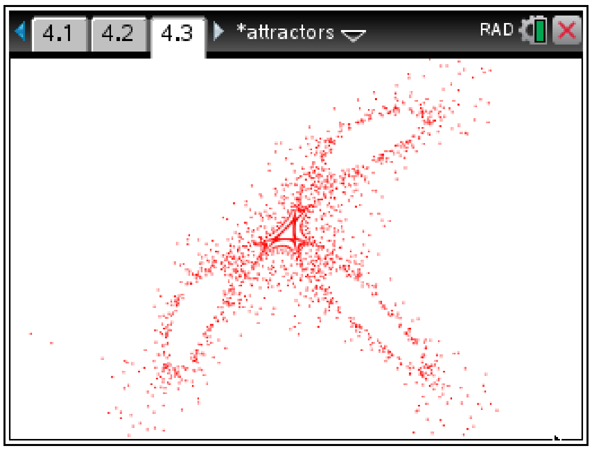
You will understand why Sprott called them “Plaid-attractors”. I must admit that I don’t have any idea for “symplectic”. I found this attribute in Mathematics Encyclopedias but I couldn’t find any connection to the iteration formulae – silly me!?

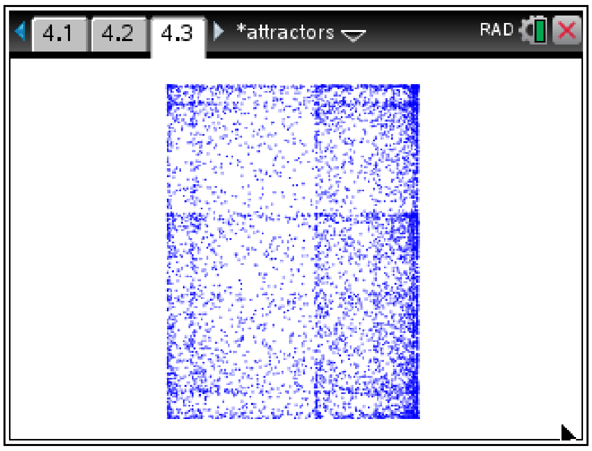
Julien C. Sprott included all his attractors (generating and plotting as well) into one huge menu-guided BASIS program. I liked to split it up. So, I separated the “symplectics” and the “plaids” from the other generalized polynomial maps.

What about them on the TI-Nspire screen?

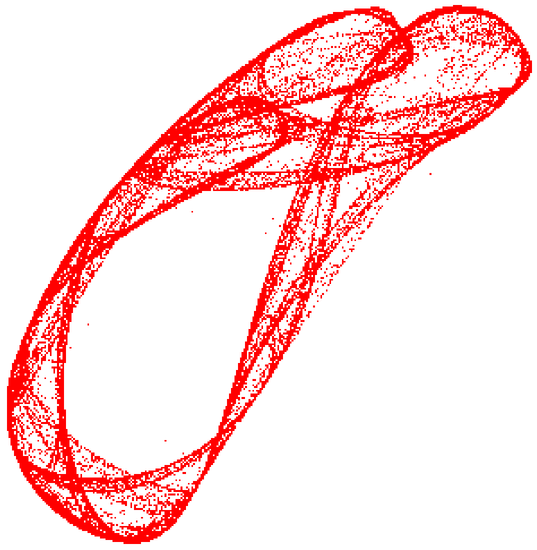
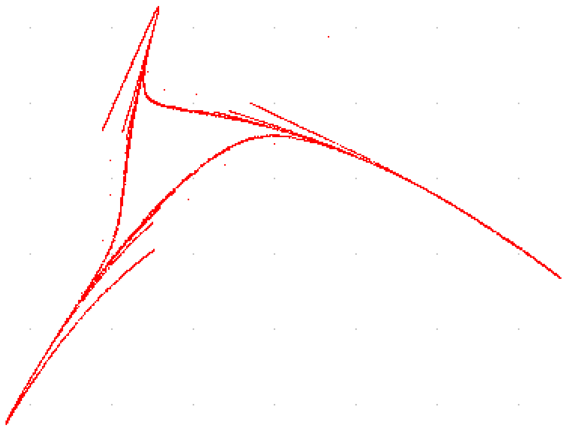
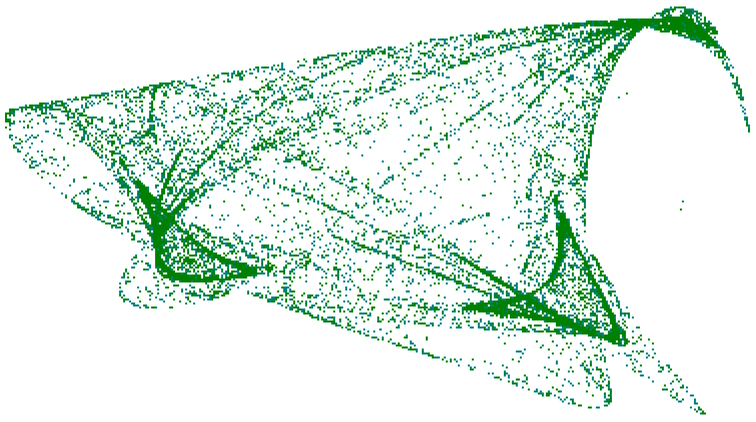


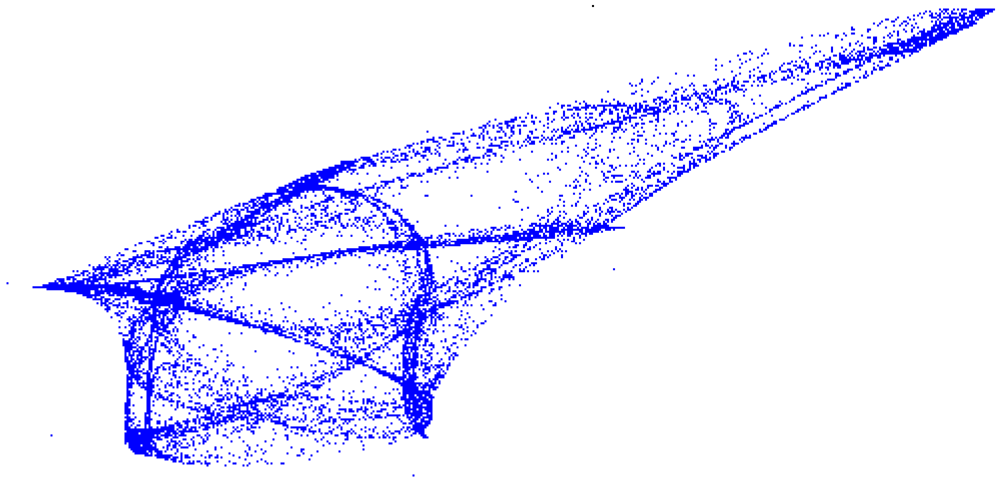
```
{0.26,-2.03,2.52,3.51,-4.94,1.,0.35,-1.}
λ=0.007
{0.17,0.3,0.82,-3.48,4.91,1.,-0.02,-1.}
λ=0.006
{0.45,-0.82,-0.9,2.08,-1.62,1.,-0.02,-1.}
λ=0.009
{0.16,-1.44,-0.08,3.22,0.44,1.,0.07,-1.}
                                                        Done
map_symp({0.26,-2.03,2.52,3.51,-4.94,1.,0.35,-1.},4000)
                      Coordinates for scatter diagram in xlist and ylist
                                                        Done
sa_plaid(2,2000)
Enter  any pos. integer or ENTER
λ=0.284
{0.57,-1.24,-0.16,-1.39,-3.47,1.39}
                                                        Done
map_plaid({0.57,-1.24,-0.16,-1.39,-3.47,1.39},6000)
```





Before exploring attractors in 3D space find some beautiful 2D-attractors:



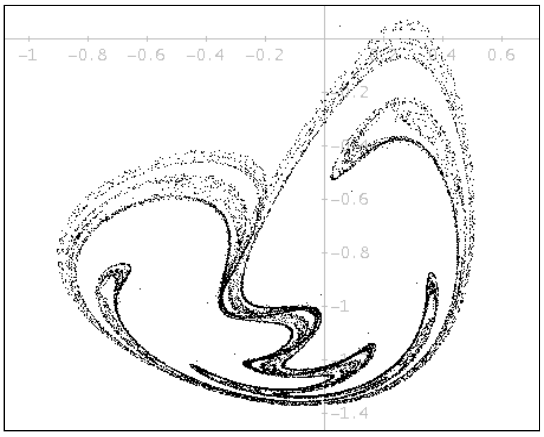
10 Special Projections

Julien C. Sprott added a nice feature to his menu driven program: One can call various projections of the 2D-maps. He offers spherical, vertical and horizontal cylindrical and toroidal projections i.e. the graphs are projected onto the front side of the respective solids.

The challenge is to reproduce this with DERIVE! See how it works.

We would like to project the given attractor (20000 points) onto the solids mentioned above.

This is one of Sprott's cubic attractors:
Code: FIRPGVTFIDGCSXMFPKIDJ



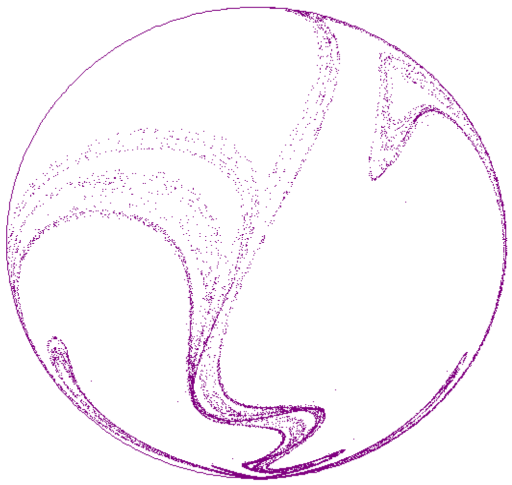
Attractor to be projected

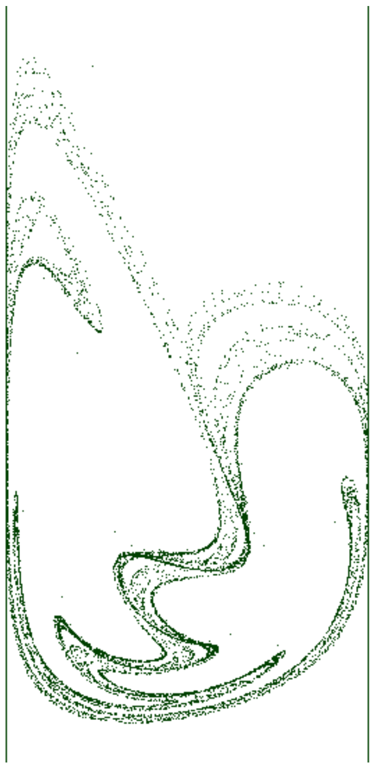
The DERIVE program is not too complicated. It is more or less the DERIVE-version of the respective part in Sprott's huge BASIC program:

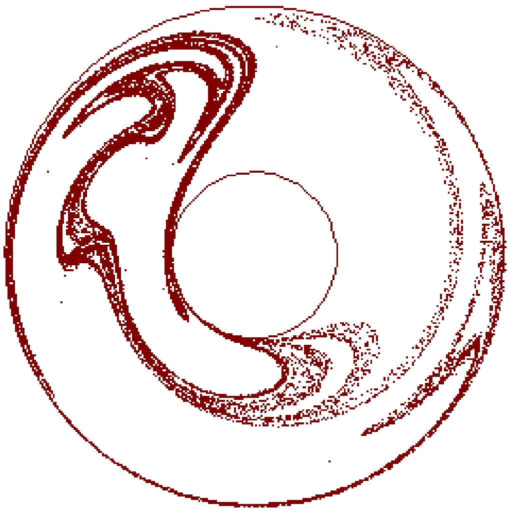
```
sph_proj(llist, list, d, i, pl_list, pt, xmin, xmax, ymin, ymax, th, ph) :=
  Prog
    [list := llist, pl_list := [x^2 + y^2 = 1]]
    d := DIM(list)
    [xmax := MAX(-1000, MAX(list↓↓1)), xmin := MIN(1000, MIN(list↓↓1))]
    [ymax := MAX(-1000, MAX(list↓↓2)), ymin := MIN(1000, MIN(list↓↓2))]
    i := 1
    Loop
      If i > d
         RETURN pl_list
      pt := list↓i
      th := π/(xmax - xmin)·(xmax - pt↓1)
      ph := π/(ymax - ymin)·(ymax - pt↓2)
      pl_list := APPEND(pl_list, [[COS(th)·SIN(ph), COS(ph)]])
      i :+ 1
```

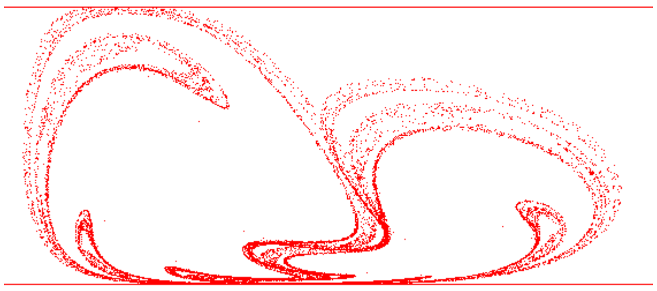
Exchanging the following lines leads to projection on a horizontal cylinder a vertical cylinder and on a torus:

```
pl_list := [-1, 1]
pl_list := APPEND(pl_list,[[(xmax - xmin)·(xmax - pt↓1),COS(ph)]])

pl_list := [x = -1, x = 1]
pl_list := APPEND(pl_list,[[-COS(th),-(ymax - ymin)·(ymax - pt↓2)]])

pl_list := [x^2+y^2 = 1.5^2, x^2+y^2 = 0.5^2]
pl_list := APPEND(pl_list,[[(1+0.5·COS(th))·SIN(ph),(1+0.5·COS(th))·COS(ph)]])
```

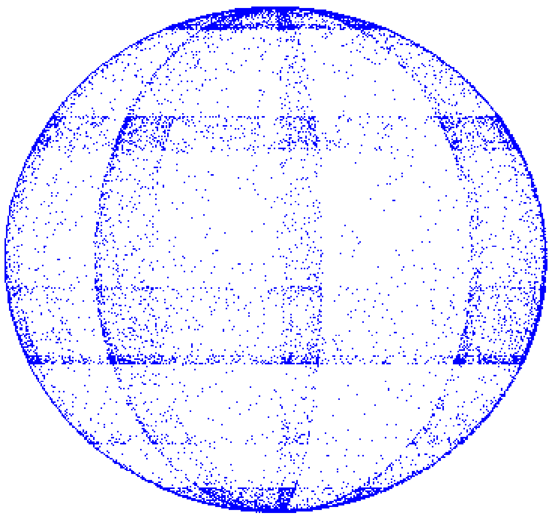

Projection on a sphere




Projection on a torus

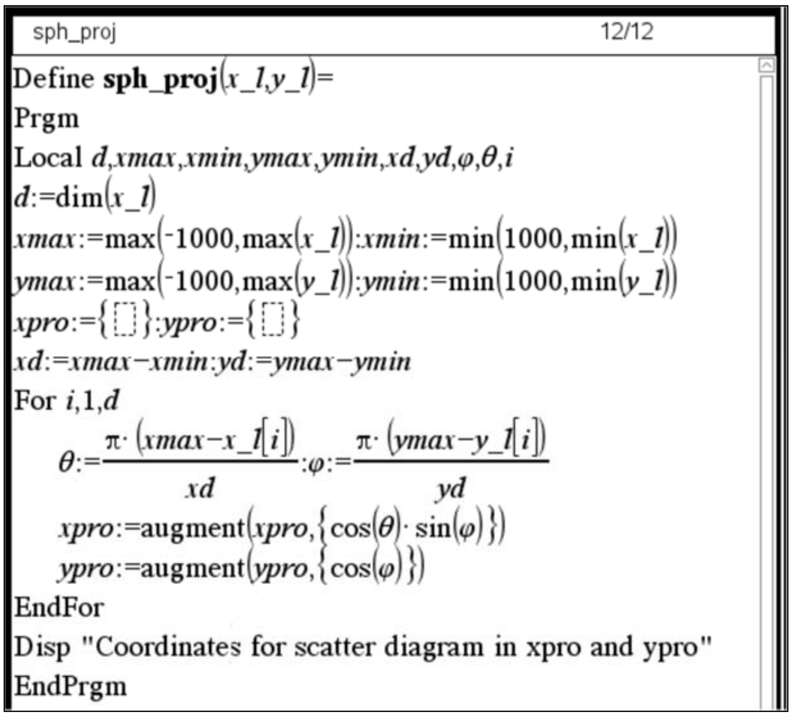


Projection on a vertical cylinder (above)
and on a horizontal cylinder (below)


Spherical projection of a plaid

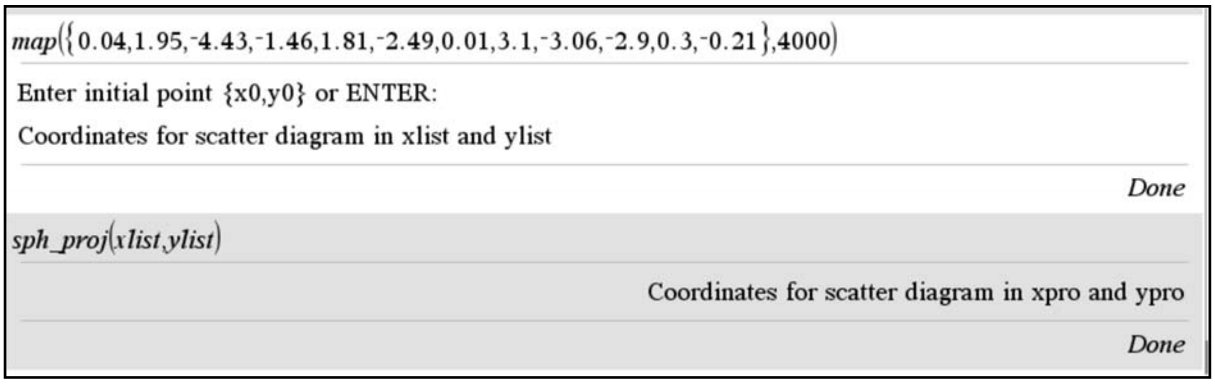
The plots are a little bit time consuming! But it is not too bad, just be patient.

Taking in account that we cannot plot 30000 or 40000 points and that we are satisfied with only some thousand points we can transfer the projection routines on the TI-NspireCAS without any problems:



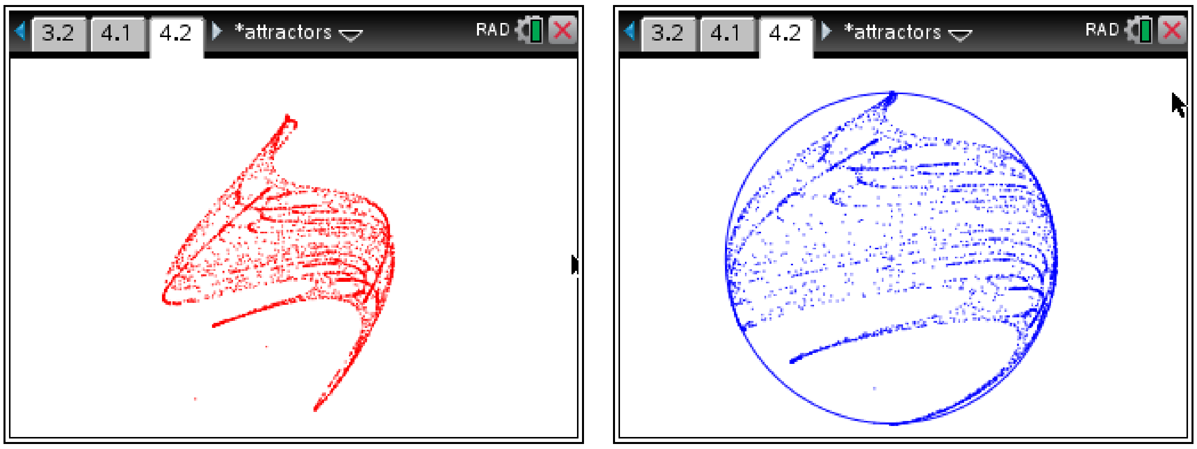
```
sph_proj                                         12/12
Define sph_proj(x_1,y_1)=
Prgm
Local d,xmax,xmin,ymax,ymin,xd,yd,φ,θ,i
d:=dim(x_1)
xmax:=max(-1000,max(x_1)):xmin:=min(1000,min(x_1))
ymax:=max(-1000,max(y_1)):ymin:=min(1000,min(y_1))
xpro:={□}:ypro:={□}
xd:=xmax-xmin:yd:=ymax-ymin
For i,1,d
  θ:=π·(xmax-x_1[i])/xd:φ:=π·(ymax-y_1[i])/yd
  xpro:=augment(xpro,{cos(θ)·sin(φ)})
  ypro:=augment(ypro,{cos(φ)})
EndFor
Disp "Coordinates for scatter diagram in xpro and ypro"
EndPrgm
```

Start with producing the scatter diagram for the attractor and then use the created lists of coordinates for the intended projections (see the spherical projection above as an example).



```
map({0.04,1.95,-4.43,-1.46,1.81,-2.49,0.01,3.1,-3.06,-2.9,0.3,-0.21},4000)
Enter initial point {x0,y0} or ENTER:
Coordinates for scatter diagram in xlist and ylist
                                                        Done
sph_proj(xlist,ylist)
                    Coordinates for scatter diagram in xpro and ypro
                                                        Done
```

How the attractor and its projection on a sphere appear on the handheld screen:

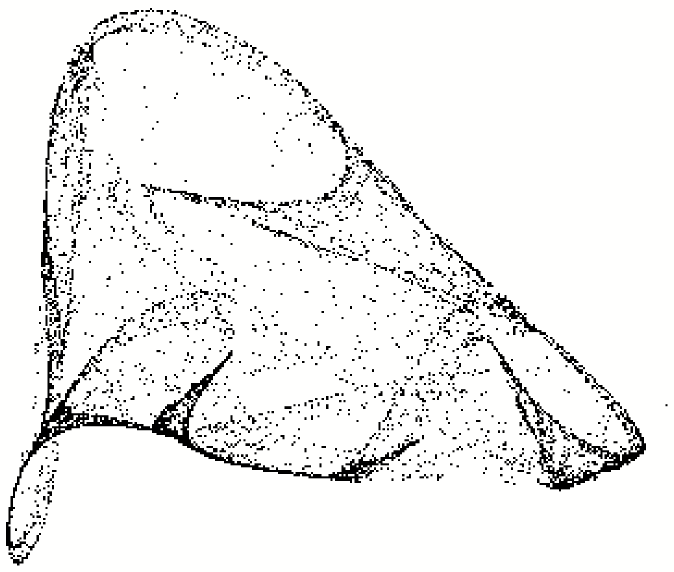


The circle must be plotted additionally in the Graph Application (right).

11 A Step into the 3rd dimension

DERIVE programs make possible to “read” Sprott’s code, to plot the 3D attractor …, and to perform own searches and discoveries.

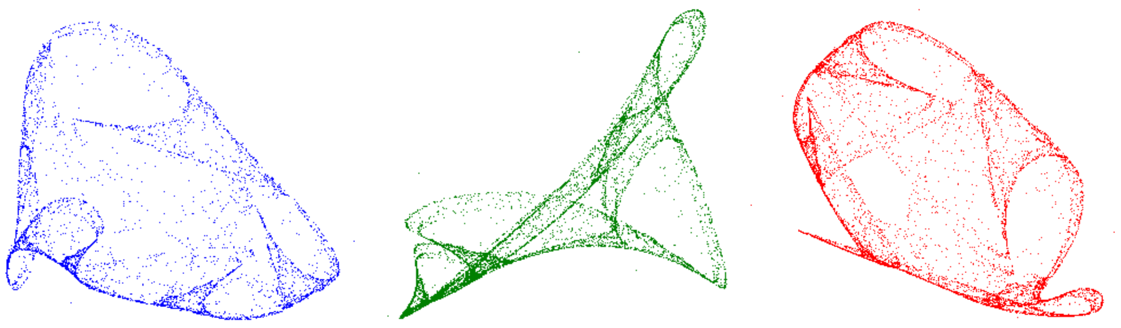
```
 sa3d(2, 500)

┌ 0.056812  [0.598, 0.148, -0.494, -0.582, -0.834, -0.172, 0.806, -0.718, -0.378, 0.604,
└  0.14289        [0.188, 0.7, -0.486, -0.56, 0.698, 0.104, 0.618, -0.42, 0.492, -0.334,

  -0.594, 0.752, 0.592, -0.266, 0.782, 0.012, -0.794, -0.034, 0.408, -0.16, -0.042, -0.3

  0.114, 0.442, 0.02, 0.02, 0.112, -0.244, 0.284, -0.772, 0.056, 0.796, 0.9, -0.546, 0.2

map_3d([0.188, 0.7, -0.486, -0.56, 0.698, 0.104, 0.618, -0.42, 0.492, -0.334, -0.302, -0

  -0.772, 0.056, 0.796, 0.9, -0.546, 0.256, 0.882, -0.48, -0.58, -0.832], 10000)
```



A 3-dimensional attractor ($\lambda = 0.14289$)

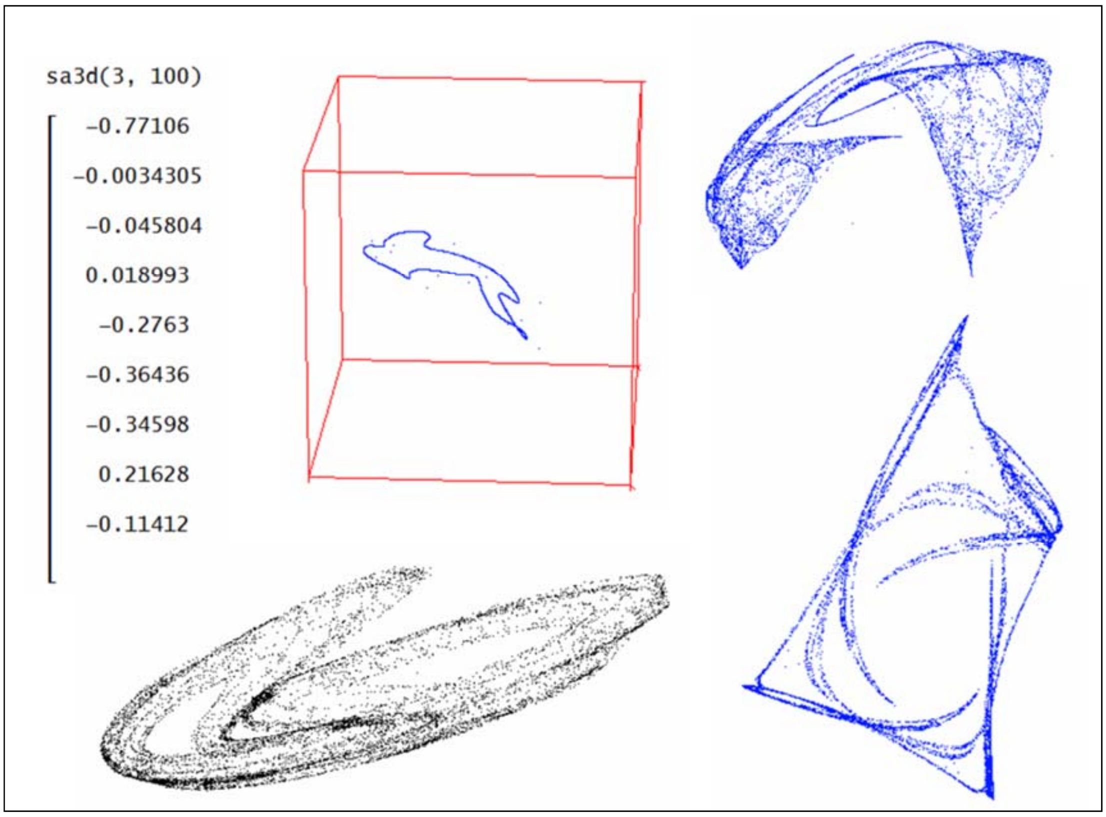
What we can do with DERIVE - and what Sprott’s awesome program is not able to do – is plotting a real 3D-graph which can be rotated. (But his program has many other features which - at least I - am unable to reproduce with DERIVE and TI-Nspire as well!!).

See below the projections of this mysterious veil from above in *x*-, *y*-, and *z*-direction.



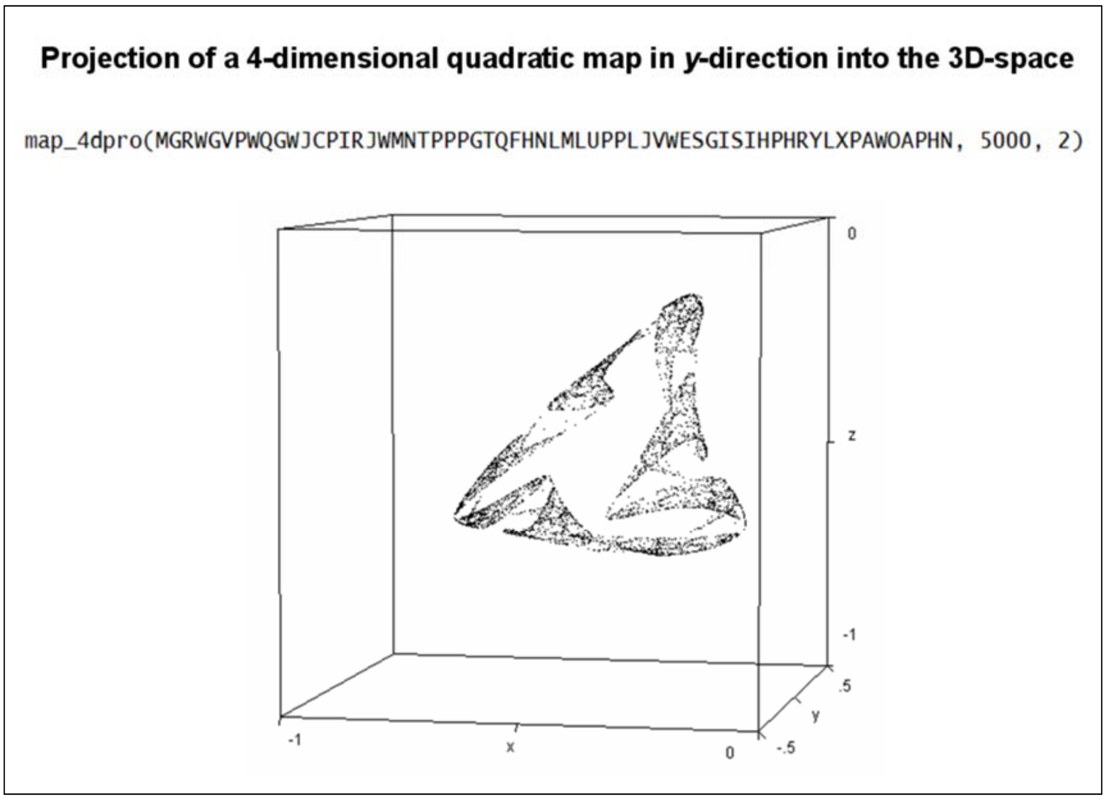
Front-, top-, and side-view of the 3D-attractor

I include two slides from my ppt-presentation:



```
sa3d(3, 100)

[   -0.77106
  -0.0034305
   -0.045804
    0.018993
     -0.2763
    -0.36436
    -0.34598
     0.21628
    -0.11412 ]
```

Some 3-dimensional attractors

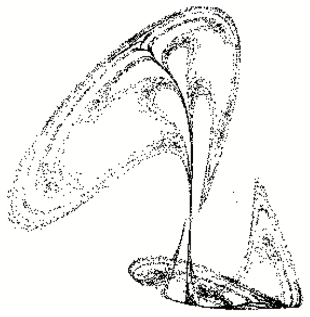
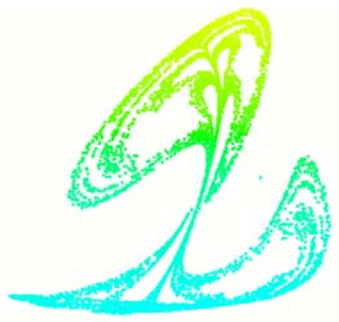


Projection of a 4-dimensional quadratic map in *y*-direction into the 3D-space

```
map_4dpro(MGRWGVPWQGWJCPIRJWMNTPPPGTQFHNLMLUPPLJVWESGISIHPHRYLXPAWOAPHN, 5000, 2)
```

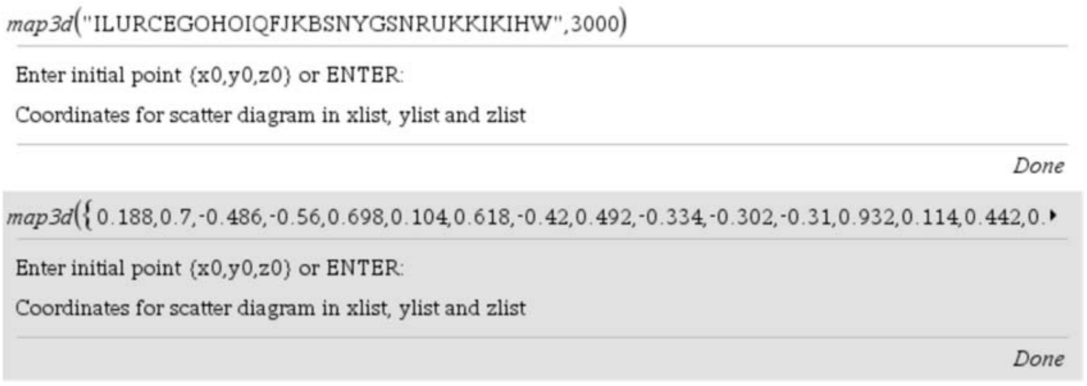
One of Sprott's 4-dimensional attractors in a 3D projection

You can plot tiny points (recommended for a large set of them – more than 10 000) or dots (recommended for a smaller amount of data points). I will describe the difference in programming both ways in a short appendix "*Points and Dots in 3D-plots*" (page 39).

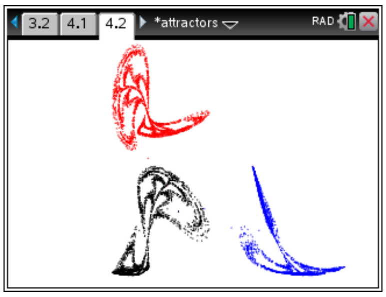
```
map_3d(ILURCEGOHOIQFJKBSNYGSNRUKKIKIHW, 10000)
map_3d_dots(ILURCEGOHOIQFJKBSNYGSNRUKKIKIHW, 10000)
```

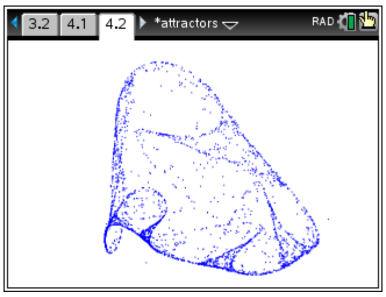
I take it always as a challenge to transfer DERIVE programs to TI-NspireCAS. You can find the programs in files SA_DERIVE_3D (for DERIVE) and attractors.tns (for TI-Nspire).

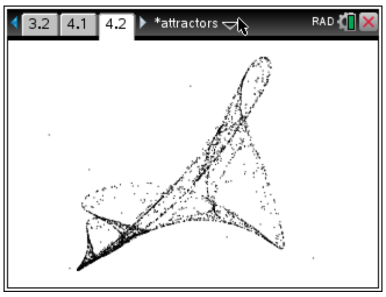


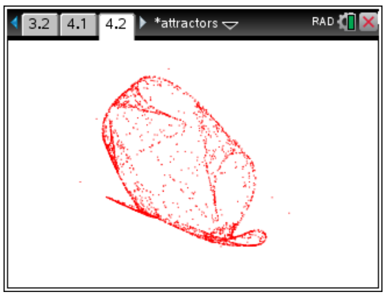
```
map3d("ILURCEGOHOIQFJKBSNYGSNRUKKIKIHW",3000)
Enter initial point {x0,y0,z0} or ENTER:
Coordinates for scatter diagram in xlist, ylist and zlist
                                                          Done
map3d({0.188,0.7,-0.486,-0.56,0.698,0.104,0.618,-0.42,0.492,-0.334,-0.302,-0.31,0.932,0.114,0.442,0.▸
Enter initial point {x0,y0,z0} or ENTER:
Coordinates for scatter diagram in xlist, ylist and zlist
                                                          Done
```

Unfortunately, we cannot plot a 3D scatter diagram with TI-Nspire. So, we must restrict our presentation on projections in direction of the axes. See first all projections on one screen (code: "ILUR…").









Compare the projections on the handheld with the DERIVE plots on page 38.

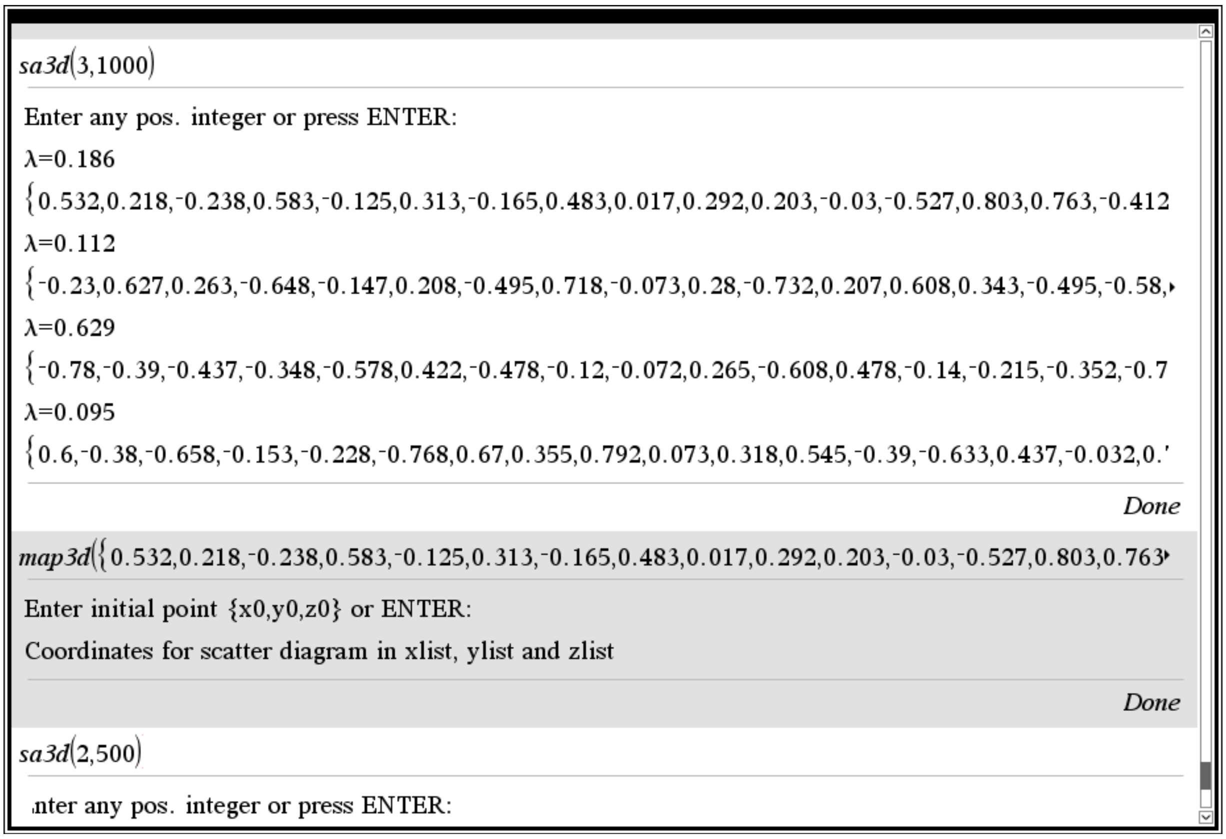
12 Search in 3rd dimension with TI-NspireCAS?

I am quite sure, that you will know, what I was doing – translating the DERIVE Code into TI-Nspire language – and it works (The coefficients in list cf are chosen between -1.3 and +1.3.)

This is the TI-NspireCAS program. Some details will be given.

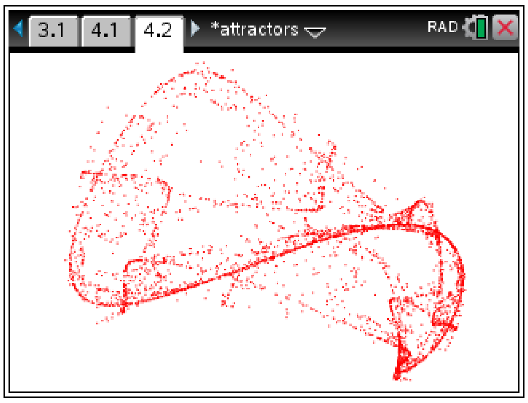
```
Define sa3d(o,n)=
Prgm
:Local r,lsum,st0,cf,cfs,fs,i,f,g,h,xs,ys,zs,xe,ye,ze,xee,yee,zee,j,rs,dl3,vals,xn,yn,zn,λ
:Try
:RequestStr "Enter any pos. integer or press ENTER:",r
:  If r≠"" Then
:    RandSeed expr(r)
:  EndIf
:  Else
:  Goto next
:EndTry
:Lbl next
:If o=2 Then
:  fs:={1,x,x^(2),x*y,x*z,y,y^(2),y*z,z,z^(2)}
:EndIf
:If o=3 Then
:  fs:={1,x,x^(2),x^(3),x^(2)*y,x^(2)*z,x*y,x*y^(2),x*y*z,x*z,x*z^(2),y,y^(2),y^(3),y^(2)*z,
       y*z,y*z^(2),z,z^(2),z^(3)}
:EndIf
:If o=4 Then
:  fs:={1,x,x^(2),x^(3),x^(4),x^(3)*y,x^(3)*z,x^(2)*y,x^(2)*y^(2),x^(2)*y*z,x^(2)*z,x^(2)*z^(2),
       x*y,x*y^(2),x*y^(3),x*y^(2)*z,x*y*z,x*y*z^(2),x*z,x*z^(2),x*z^(3),y,y^(2),y^(3),y^(4),
       y^(3)*z,y^(2)*z,y^(2)*z^(2),y*z,y*z^(2),y*z^(3),z,z^(2),z^(3),z^(4)}
:EndIf
:cfs:=dim(fs) : :vals:=0
:For i,1,n
:cf:=((floor(100*(2.6*rand(3*cfs)−1.3)))/(100))
:f:=sum(left(cf,cfs)*fs):g:=sum(mid(cf,cfs+1,cfs)*fs):h:=sum(right(cf,cfs)*fs)
:lsum:=0
:st0:=iterate_3({f,g,h},{x,y,z},{0.05,0.05,0.05},5)
:xs:=st0[1]:ys:=st0[2]:zs:=st0[3]:xe:=xs+1.ᴇ−6: ye:=ys: ze:=zs
:For j,1,2000
:  If abs(xs)>10 or abs(ys)>10 or abs(zs)>10 Then
:    Goto nexttry
:  EndIf
:  xn:=f|x=xs and y=ys and z=zs:   yn:=g|x=xs and y=ys and z=zs
:  zn:=h|x=xs and y=ys and z=zs:   xee:=f|x=xe and y=ye and z=ze
:  yee:=g|x=xe and y=ye and z=ze:   zee:=h|x=xe and y=ye and z=ze
:  xe:=xee: ye:=yee: ze:=zee
:   dl3:=(xn−xe)^(2)+(yn−ye)^(2)+(zn−ze)^(2):rs:=((1)/(10^(6)*√(dl3)))
:   xs:=xn:ys:=yn:zs:=zn:xe:=xs+rs*(xe−xs):ye:=ys+rs*(ye−ys):ze:=zs+rs*(ze−zs)
:   lsum:=lsum+ln(10^(12)*dl3)
:EndFor
:λ:=((((1)/(2*ln(2)))*lsum)/(j))
:If λ>0.05 Then
:   vals:=1:Disp "λ="&string(λ):Disp cf
:EndIf
:Lbl nexttry
:EndFor
:If vals=0 Then
:   Disp "no attractor found"
:EndIf
:EndPrgm
```

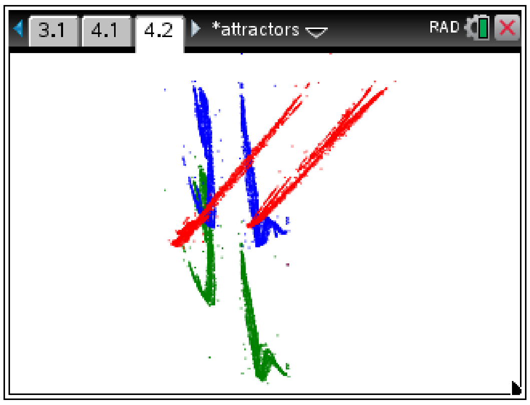
You can use the built-in initial number for the random number generator or you can enter any integer for generating the set of coefficients cf. The important part is calculating the Lyapunov-exponent λ. All sets of coefficients giving λ > 0.05 promise a – more or less – attractive attractor.



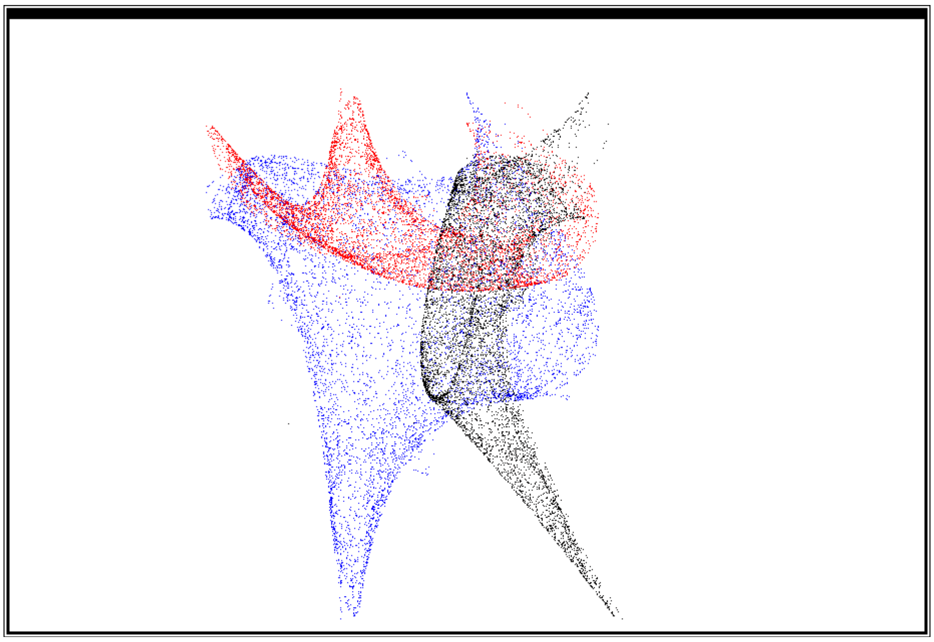
```
sa3d(3,1000)
Enter any pos. integer or press ENTER:
λ=0.186
{0.532,0.218,-0.238,0.583,-0.125,0.313,-0.165,0.483,0.017,0.292,0.203,-0.03,-0.527,0.803,0.763,-0.412
λ=0.112
{-0.23,0.627,0.263,-0.648,-0.147,0.208,-0.495,0.718,-0.073,0.28,-0.732,0.207,0.608,0.343,-0.495,-0.58,▸
λ=0.629
{-0.78,-0.39,-0.437,-0.348,-0.578,0.422,-0.478,-0.12,-0.072,0.265,-0.608,0.478,-0.14,-0.215,-0.352,-0.7
λ=0.095
{0.6,-0.38,-0.658,-0.153,-0.228,-0.768,0.67,0.355,0.792,0.073,0.318,0.545,-0.39,-0.633,0.437,-0.032,0.'
                                                                                              Done
map3d({0.532,0.218,-0.238,0.583,-0.125,0.313,-0.165,0.483,0.017,0.292,0.203,-0.03,-0.527,0.803,0.763▸
Enter initial point {x0,y0,z0} or ENTER:
Coordinates for scatter diagram in xlist, ylist and zlist
                                                                                              Done
sa3d(2,500)
nter any pos. integer or press ENTER:
```

My very first 1000 tries gave four positive L.Es. Then I applied *map3d* in order to receive the lists of coordinates. Thus, I was able to plot a projection on a plane as a scatter diagram:

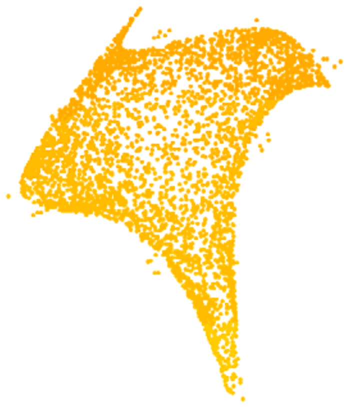


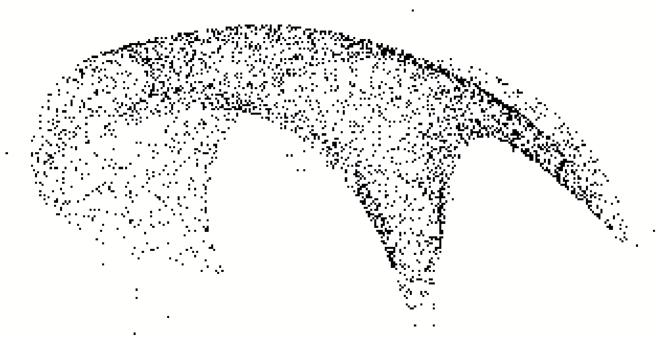


The right plot shows all possible projections of a quadratic 3D-attractor. Another attractive attractor is presented below.



Just for fun I copied the list of coefficients to DERIVE and tried to plot the attractor in form of 30000 dots (left, heat wave) and 40000 points (right). Now I could perform a true 3D-plot and move the cloud of data points around.





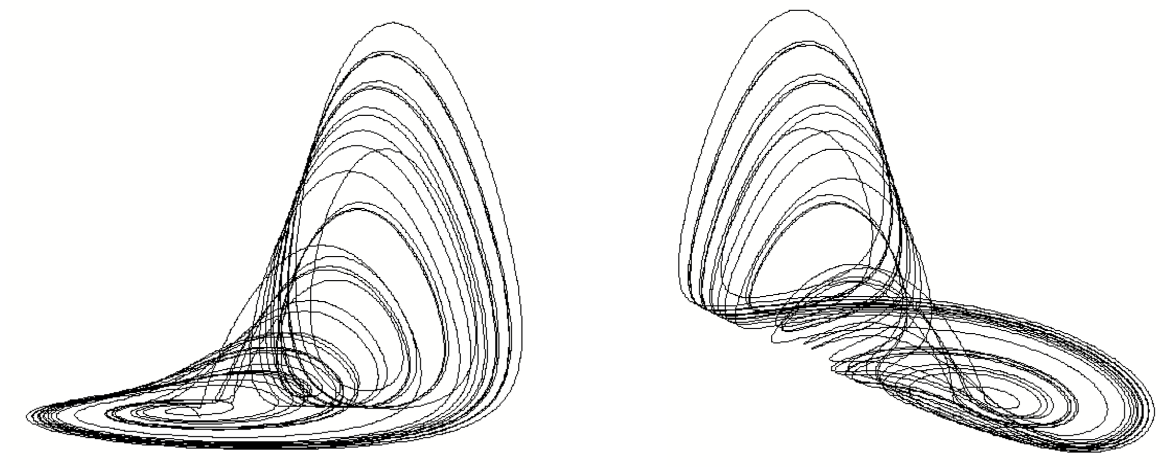
I skip presenting the procedure to create 3D-projections of 4D attractors. The DERIVE program is included in SA_DERIVE_3D.dfw.

13 The Rössler Attractor

Two very famous and well known 3-dimensional attractors are the Lorenz attractor and the Rössler-attractor. We will skip the first one and focus on the latter one.

We receive it when solving the following system of ODEs numerically.

$$\dot{x} = -y - z$$
$$\dot{y} = x + a\,y$$
$$\dot{z} = b + z\,(x - c)$$



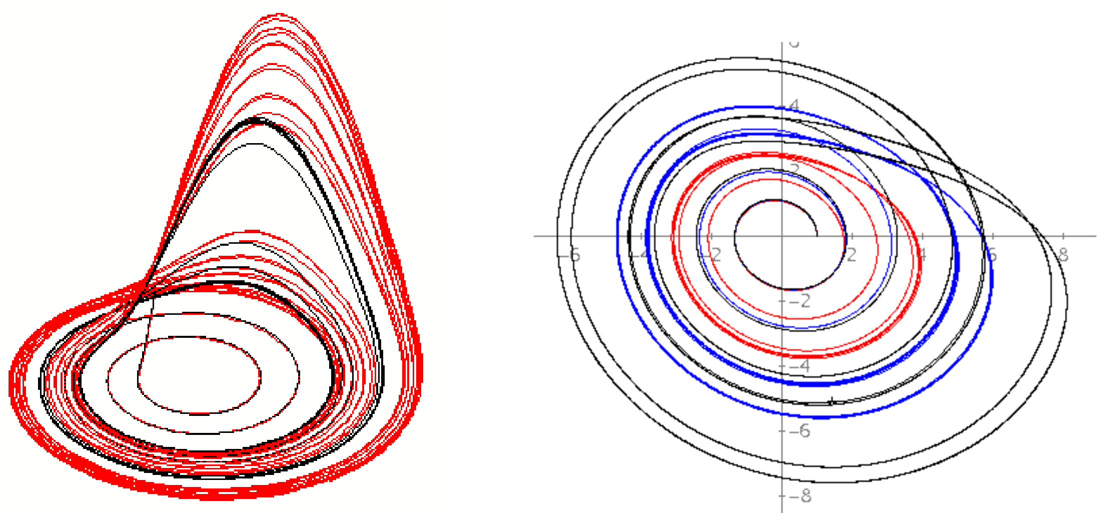
3D graph of the Rössler-attractor (DERIVE made)

I can do this using various software tools.

With DERIVE I will apply the implemented Runge-Kutta-Algorithm:

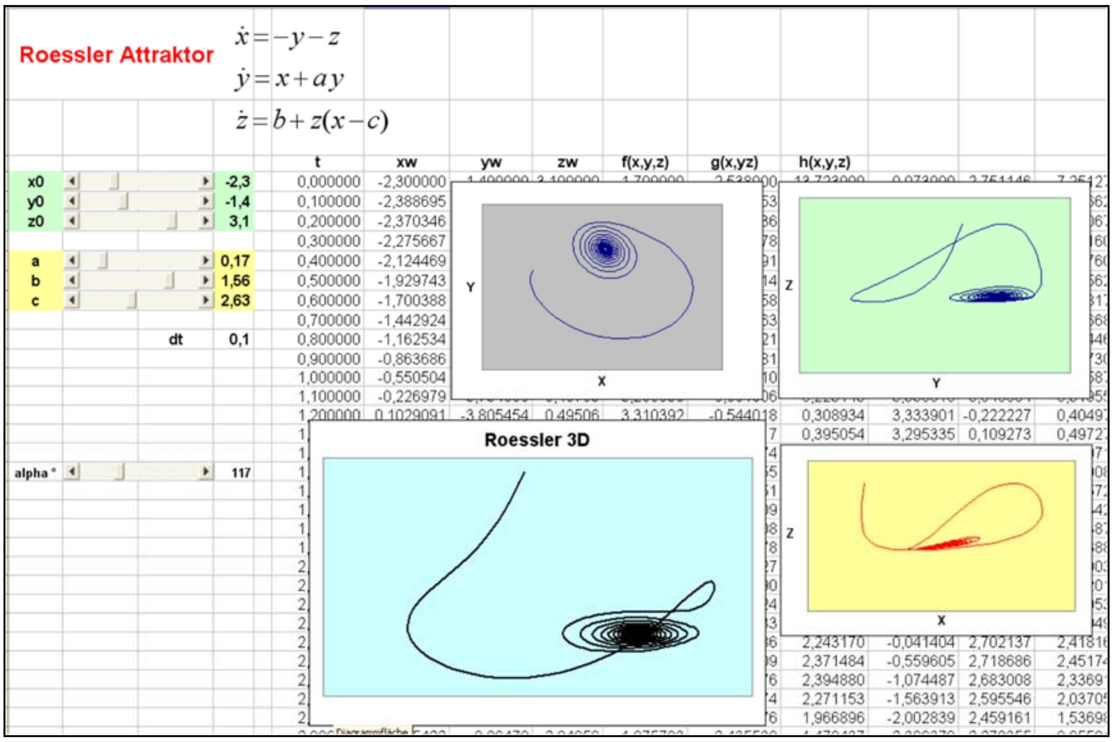
Taking the parameters as follows we have limit cycles for *c* = 2, 3 and 4, which let us recognize the period doubling.

```
VECTOR((RK([-y - z, x + 0.2·y, 0.2 + z·(x - c)], [t, x, y, z], [0, 1, 0, 0],
       0.01, 10000))↓↓[2, 3], c, 2, 4)
```



Three Rössler-attractors for *c* = 2, 3, 4 (3D and top view)

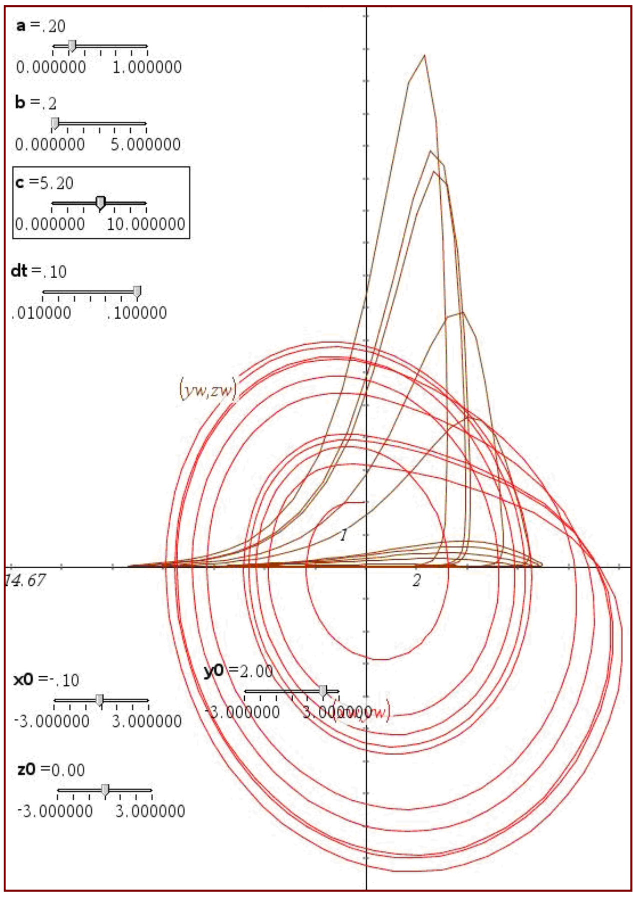
This the representation of the attractor with MS Excel. As you can see I introduced sliders for the parameters *a, b,* and *c* and the initial values as well. The 3D representation is the result of a transformation into the *xy*-plane.



Rössler-attractors in a spreadsheet with sliders

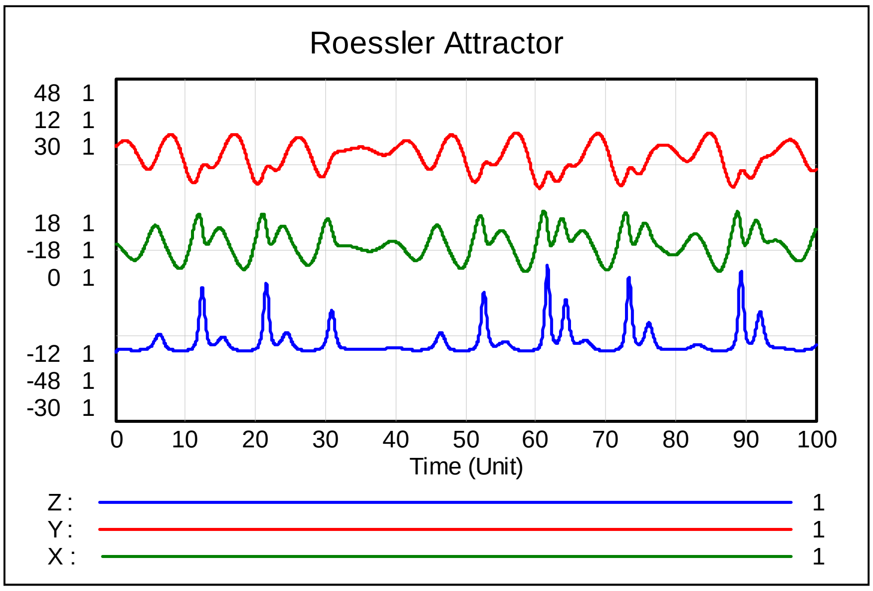
TI-Nspire also allows using sliders.

Here is an additional slider for *dt* added.



Rössler-attractor with TI-NspireCAS

Finally, I'd like to show the oscillations of *x*, *y*, and *z* created with the dynamic systems software VENSIM PLE.

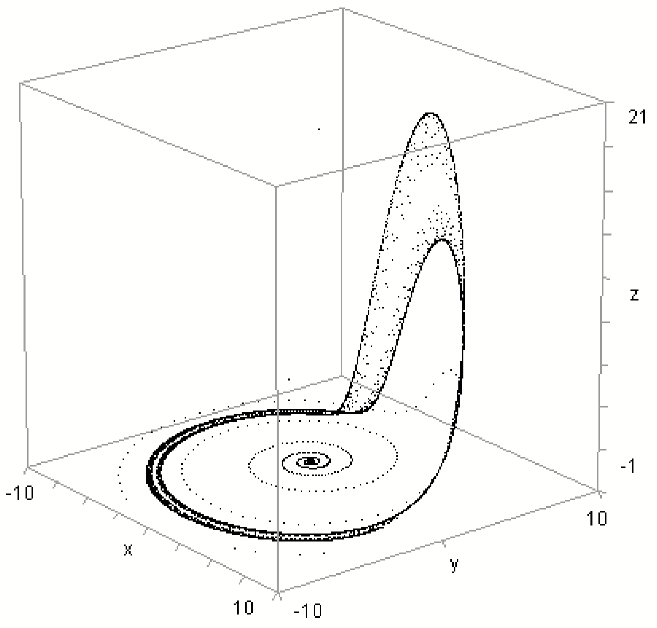


Rössler attractor with VENSIM PLE

13 "Attractive" systems of ODEs

Sprott mentions the Rössler-attractor, too and offers it as one of an infinite number of systems of ODEs generating unexpected graphs. I transferred his program to DERIVE and plotted the attractor without Runge-Kutta as a set of 10000 not connected points.

```
map_ode([0, 0, 0, 0, 0, -1, 0, 0, -1, 0, 0, 1, 0, 0, 0, 0.15, 0, 0, 0, 0,

   0.2, 0, 0, 0, 1, 0, 0, 0, -5, 0], 10000)
```



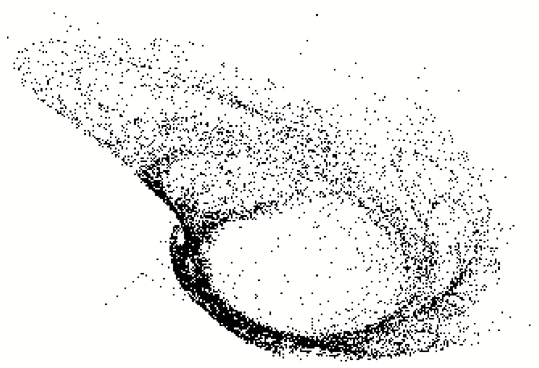
Rössler-attractor as a strange attractor

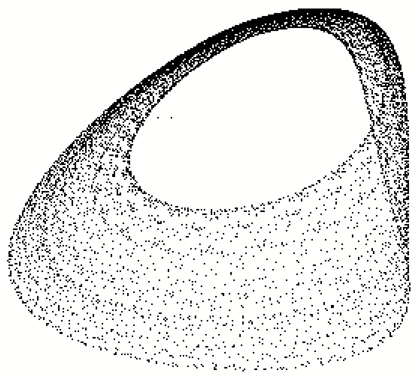
Maybe that you now can imagine that I got an appetite for my own homemade ODE attractors.

```
sa_ode(o, n, tr, t_, xs, ys, zs, xn, yn, zn, xe, ye, ze, xee, yee, zee, f, fs, g,
  Prog
    dummy := RANDOM(0)
    tr := 1
    vals := []
    cfs := (o + 1)·(o + 2)·(o + 3)/6
    fs := [0, [1, x, x^2, x·y, x·z, y, y^2, y·z, z, z^2], [1, x, x^2, x^3, x^2·y, ›
    Loop
      If tr > n exit
      WRITE([tr, DIM(vals)])
      cf := VECTOR(FLOOR(100·(4·RANDOM(1) - 2))/100, k, 3·cfs)
      f := x + 0.1·cf↓[1, ..., cfs]·fs↓o
      g := y + 0.1·cf↓[cfs + 1, ..., 2·cfs]·fs↓o
      h := z + 0.1·cf↓[2·cfs + 1, ..., 3·cfs]·fs↓o
      st0 := [0.05, 0.05, 0.05]
      [lsum := 0, l := 0]
      st0 := ITERATE([f, g, h], [x, y, z], st0, 5)
      [xs := st0↓1, ys := st0↓2, zs := st0↓3]
      [xe := xs + 0.000001, ye := ys, ze := zs]
      i := 1
      Loop
        If ABS(xs) > 10 ∨ ABS(ys) > 10 ∨ ABS(zs) > 10 exit
        If i > 300 ∧ l > 0
           vals := APPEND(vals, [[l, cf]])
        If i > 300 exit
        xn := SUBST(f, [x, y, z], [xs, ys, zs])
        yn := SUBST(g, [x, y, z], [xs, ys, zs])
        zn := SUBST(h, [x, y, z], [xs, ys, zs])
        xee := SUBST(f, [x, y, z], [xe, ye, ze])
        yee := SUBST(g, [x, y, z], [xe, ye, ze])
        zee := SUBST(h, [x, y, z], [xe, ye, ze])
        [xe := xee, ye := yee, ze := zee]
        dl3 := (xn - xe)^2 + (yn - ye)^2 + (zn - ze)^2
        rs := 1/(10^6·√dl3)
        [xs := xn, ys := yn, zs := zn]
        [xe := xs + rs·(xe - xs), ye := ys + rs·(ye - ys), ze := zs + rs·(ze - zs)]
        lsum := lsum + LOG(10^12·dl3)
        l := 0.72134·lsum/(i·0.1)
        i :+ 1
      tr :+ 1
    RETURN vals
```

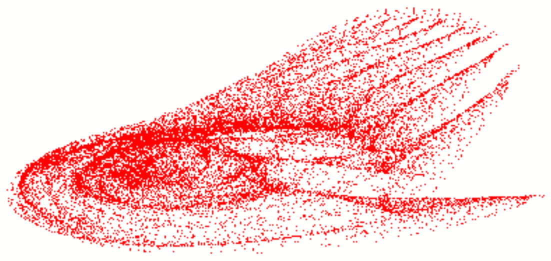
Above is a part of my “Search Attractive ODEs”-program with order $2 \le o \le 5$ testing n sets of randomly generated coefficients between -2 and +2.

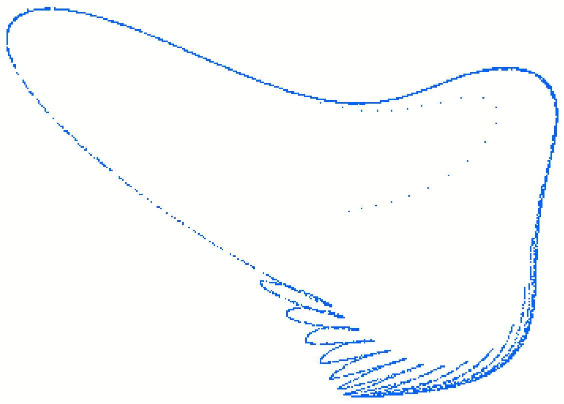
In the following will present some of my findings:

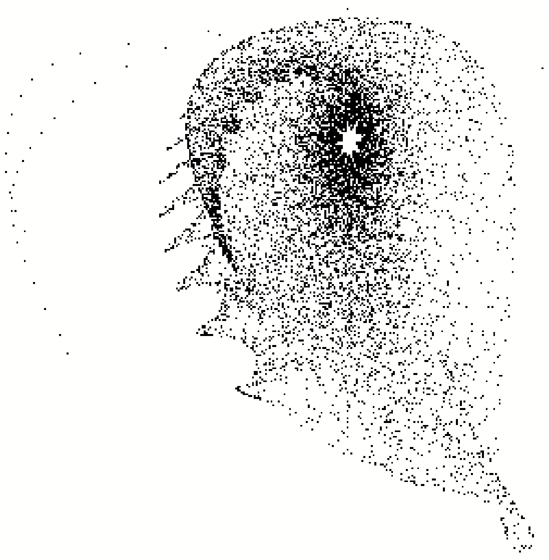

Order 2, $\lambda = 1.59$, 10000 points

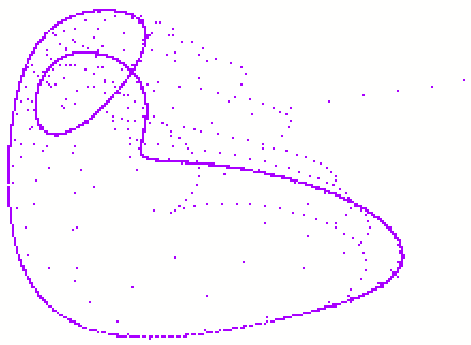

Order 2, $\lambda = 0.09$, 10000 points

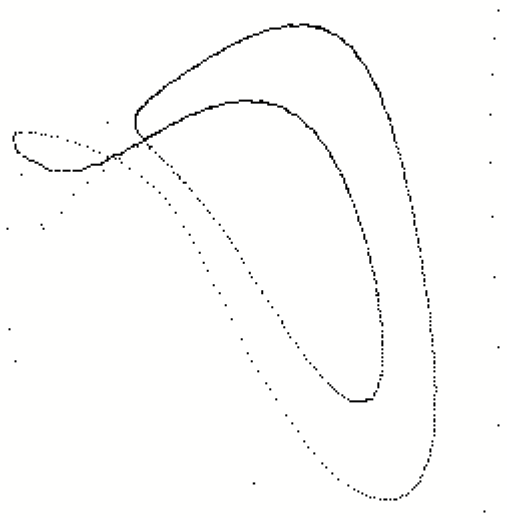
Trajectories of 2nd order ODE systems

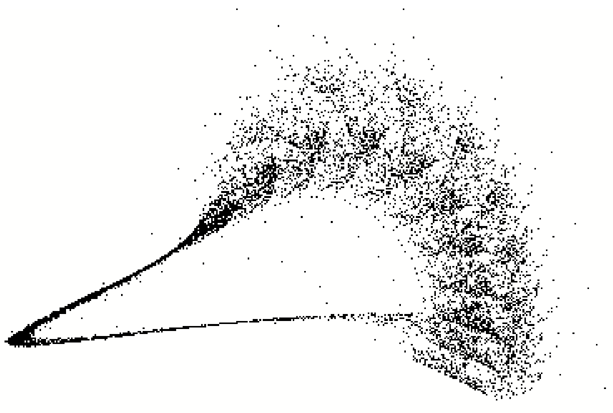

Order 3, λ = 1.30, 20000 points


Order 3, λ = 1.15, 10000 points


Order 4, λ = 0.886, 10000 points


Order 4, λ = 0.26, 10000 points

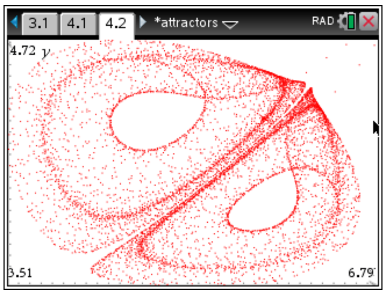

Order 5, λ = 0.09, 1000 points

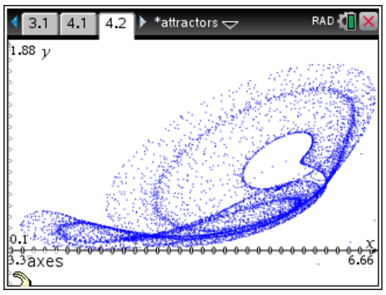

Order 5, λ = 1.43, 10000 points

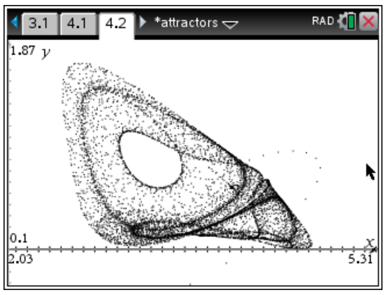
Trajectories of ODE systems of order 3,4 and 5

TI-NspireCAS proudly presents on of Sprott's ODE-attractors:

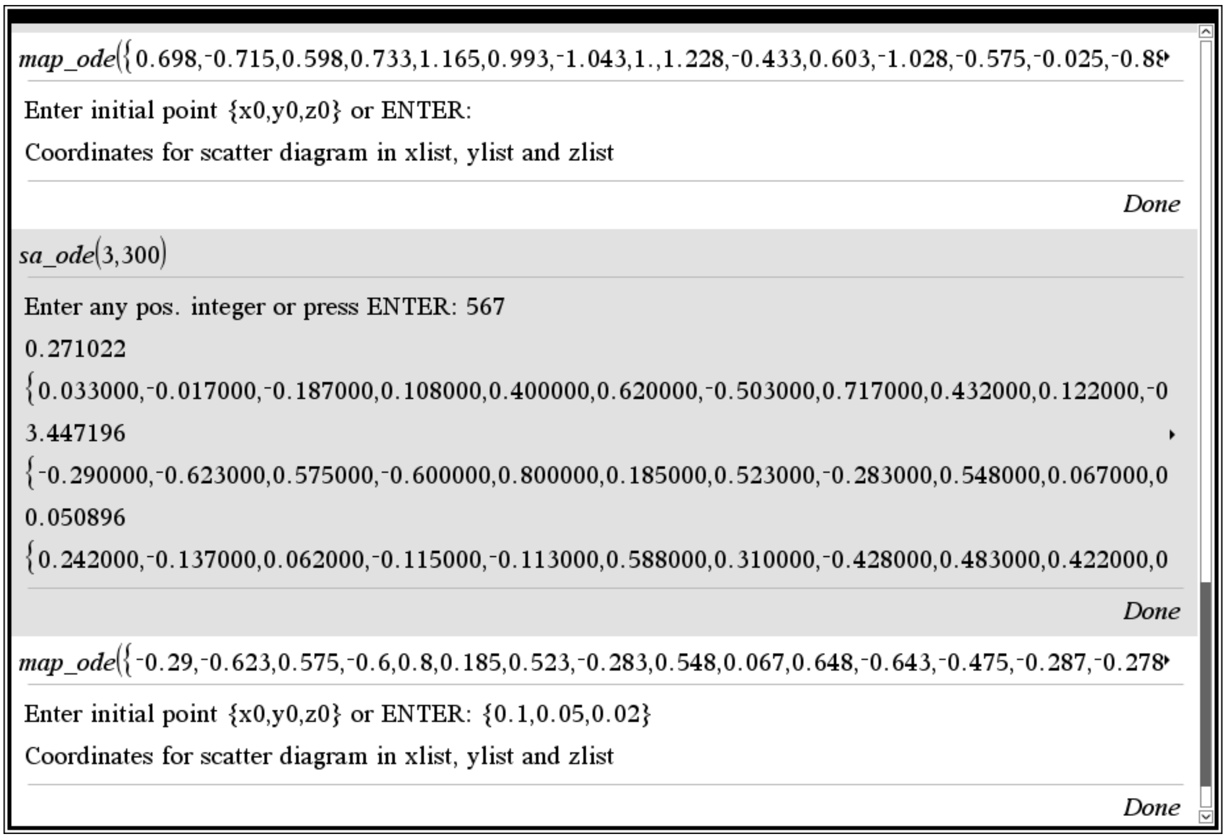
map_ode("QRREQDTWELEQMTMLAAPRGDJJKLPYAFO",7000)





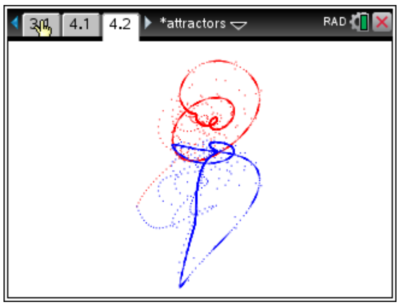


What's about our own ODE-attractors? No problem – see below:

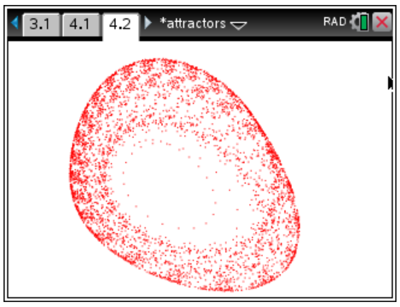


```
map_ode({0.698,⁻0.715,0.598,0.733,1.165,0.993,⁻1.043,1.,1.228,⁻0.433,0.603,⁻1.028,⁻0.575,⁻0.025,⁻0.8▸
 Enter initial point {x0,y0,z0} or ENTER:
 Coordinates for scatter diagram in xlist, ylist and zlist
                                                                                        Done
sa_ode(3,300)
 Enter any pos. integer or press ENTER: 567
 0.271022
 {0.033000,⁻0.017000,⁻0.187000,0.108000,0.400000,0.620000,⁻0.503000,0.717000,0.432000,0.122000,⁻0
 3.447196                                                                                  ▸
 {⁻0.290000,⁻0.623000,0.575000,⁻0.600000,0.800000,0.185000,0.523000,⁻0.283000,0.548000,0.067000,0
 0.050896
 {0.242000,⁻0.137000,0.062000,⁻0.115000,⁻0.113000,0.588000,0.310000,⁻0.428000,0.483000,0.422000,0
                                                                                        Done
map_ode({⁻0.29,⁻0.623,0.575,⁻0.6,0.8,0.185,0.523,⁻0.283,0.548,0.067,0.648,⁻0.643,⁻0.475,⁻0.287,⁻0.278▸
 Enter initial point {x0,y0,z0} or ENTER: {0.1,0.05,0.02}
 Coordinates for scatter diagram in xlist, ylist and zlist
                                                                                        Done
```

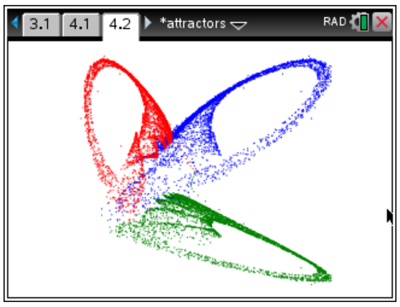
We enter sa_ode(2≤order≤4), number of tries) and receive sets of coefficients together with the corresponding Lyapunov-exponent in two separated lines. Then it is comfortable to copy the list into the map_ode()-program call.



Order 4, λ = 0.16 (top and side view)



Order 2, λ = 0.24



Order 3, λ = 3.45 (all views)

14 Gumowski-Mira-Attractors

You can find many other features offered in Sprott's book and in his SA.EXE program: attractors with shadows, 4-dimensional attractors with the 4th dimension interpreted as colour or as sound, sliced attractors, attractors as analglyphs, and, and, and, … Now we will leave Sprott's wonderful source of inspiration.

I will take the occasion to present some other types of recursive functions which also result in surprising graphs.

Two physicists, I. Gumowski and C. Mira found in the frame of their work at CERN in Geneva in 1980 a new type of attractors [12, 13].

The original model (type 1) is described by

$$\begin{aligned} x_{n+1} &= b\cdot y_n + f(x_n) \\ y_{n+1} &= -x_n + f(x_{n+1}) \end{aligned} \quad \text{with } f(x) = a\cdot x + \frac{2\,(1-a)\,x^2}{1+x^2}; \; a, b \text{ are constants.}$$

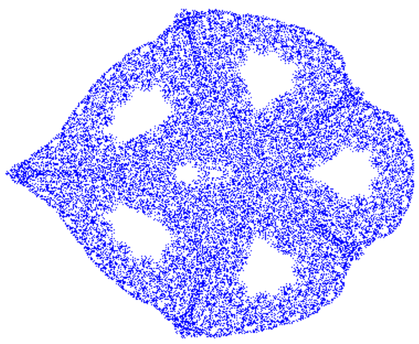
You can find many variations of the original form (see the references). Here is one of them (type 2):

$$\begin{aligned} x_{n+1} &= y_n + a\cdot(1-b\cdot y_n^{\,2})\cdot y_n + f(x_n) \\ y_{n+1} &= -x_n + f(x_{n+1}) \end{aligned} \quad \text{with } f(x) = \mu\cdot x + \frac{2\,(1-\mu)\,x^2}{1+x^2}; \; a, b, \mu \text{ are constants.}$$

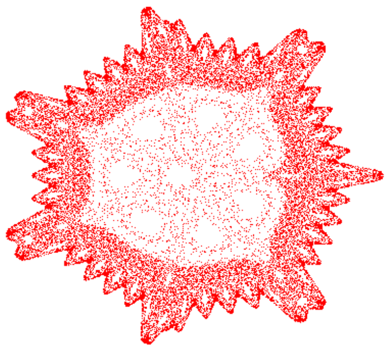
You can also vary function $f(x)$.

I tried to write my DERIVE-function `gm(a, b, f, n, type, x0, y0)` in a form to cope with both types. *type* = 1, *x*0 = *y*0 = 0.1 are the default values.

```
gm(aa, bb, f, n, type := 1, x0 := 0.1, y0 := 0.1, i, pts, xn, yn) :=
  Prog
    pts := [[x0, y0]]
    i := 1
    Loop
      If i > n
         RETURN pts
      xn := ((1 - bb)·(type - 2) + 1)·y0 + (type - 1)·aa·(1 - bb·y0^2)·y0 +
           LIM(LIM(LIM(f, x, x0), a, aa), b, bb)
      yn := -x0 + LIM(LIM(LIM(f, x, xn), a, aa), b, bb)
      pts := APPEND(pts, [[xn, yn]])
      [x0 := xn, y0 := yn]
      i :+ 1
```



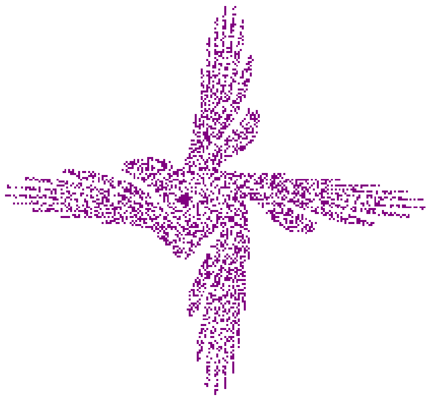
$$\text{gm}\left(0.245,\ 1,\ a\cdot x + \frac{2\cdot(1 - a)\cdot x^2}{1 + x^2},\ 30000,\ 1,\ 1,\ 1\right)$$



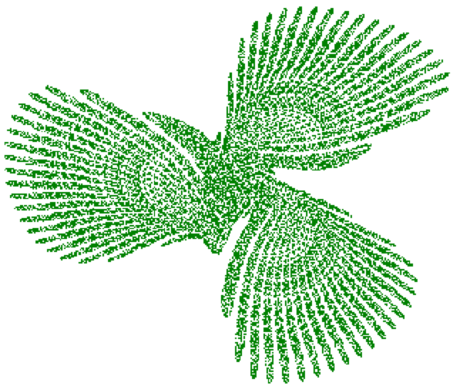
$$\text{gm}\left(-0.245,\ 1,\ a\cdot x + \frac{2\cdot(1 - a)\cdot x^2}{1 + x^2},\ 30000,\ 1,\ 1,\ 1\right)$$

My first Gumowski-Mira attractors

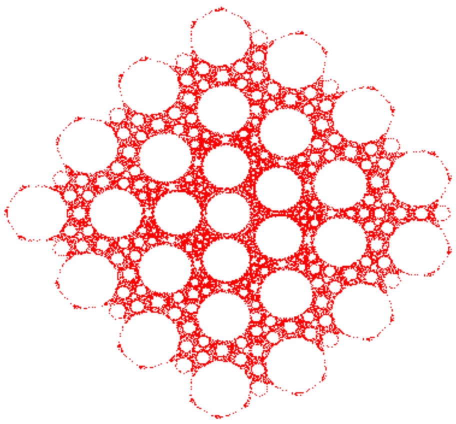
A short visit in my Gumowski-Mira Gallery:



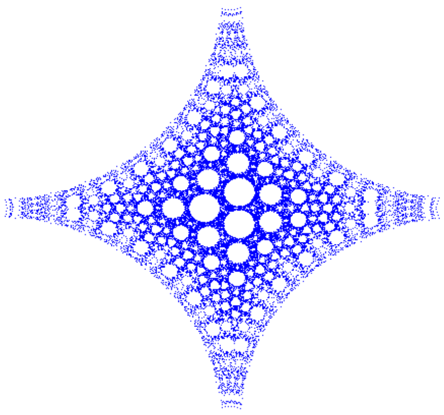
$$\mathrm{gm}\left(0.01,\ 0.978,\ a\cdot x + \frac{2\cdot(1-a)\cdot x^2}{1+x^2},\ 30000,\ 1,\ 1,\ 1\right)$$



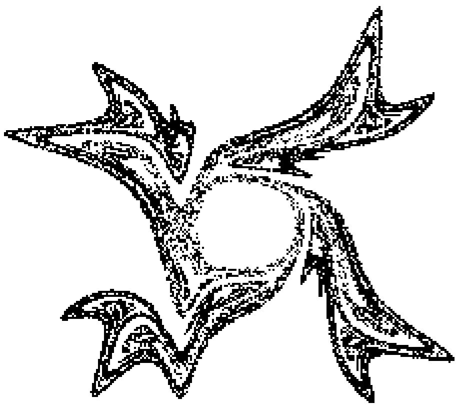
$$\mathrm{gm}\left(-0.48,\ 0.9924,\ a\cdot x + \frac{2\cdot(1-a)\cdot x^2}{1+x^2},\ 30000,\ 1,\ 1,\ 1\right)$$



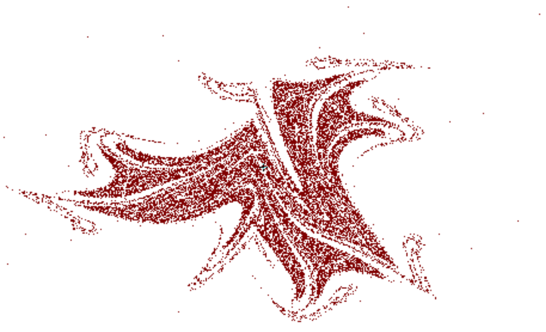
$$\mathrm{gm}\left(0.04,\ 1,\ a\cdot x - \frac{3-a}{a+e^{b\cdot x}},\ 30000,\ 1,\ -2.8,\ 17\right)$$



$$\mathrm{gm}\left(0.04,\ 1,\ a\cdot x + \frac{3-a}{a+e^{b\cdot x}},\ 30000,\ 1,\ -2.8,\ 17\right)$$



$$\mathrm{gm}\left(0.01,\ 0.08,\ -0.105\cdot x + \frac{2.4\cdot 1.7\cdot x^2}{1+x^2},\ 20000,\ 2\right)$$



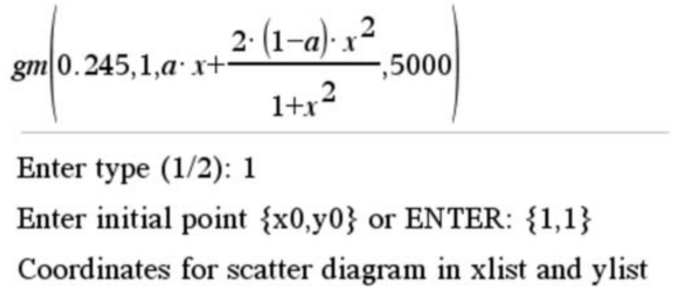
$$\mathrm{gm}\left(-0.96,\ 0.96,\ a\cdot x - \left|\mathrm{ATAN}(a-x) + \frac{b\cdot x^2}{1+x^2}\right|,\ 20000,\ 1,\ 12,\ 9\right)$$

Gumowski-Mira-attractors

And here is the TI-NspireCAS version:

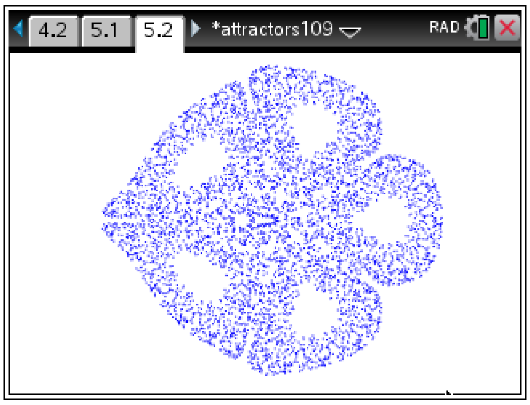
```
Define gm(aa,bb,f,n)=
Prgm
:Local type,start,xo,yo,xn,yn,i
:Lbl typ_
:Request "Enter type (1/2):",type
:If type≠1 and type≠2 Then
:   Goto typ_
:EndIf
:Try
:RequestStr "Enter initial point {x0,y0} or ENTER:",start
:If start≠"" Then
:   start:=expr(start)
:EndIf
:Else
:   start:={0.1,0.1}
:   Goto next
:EndTry
:Lbl next
:xo:=start[1]:yo:=start[2]:xlist:={xo}:ylist:={yo}
:For i,1,n
:If abs(xo)>100 or abs(yo)>100 Then
:   Goto end
:EndIf
:xn:=((1−bb)*(type−2)+1)*yo+(type−1)*aa*(1−bb*yo^(2))*yo+f|x=xo and a=aa and b=bb
:yn:=⁻xo+f|x=xn and a=aa and b=bb
:xlist:=augment(xlist,{xn}):ylist:=augment(ylist,{yn})
:xo:=xn:yo:=yn
:EndFor
:Lbl end
:Disp "Coordinates for scatter diagram in xlist and ylist"
:EndPrgm
```

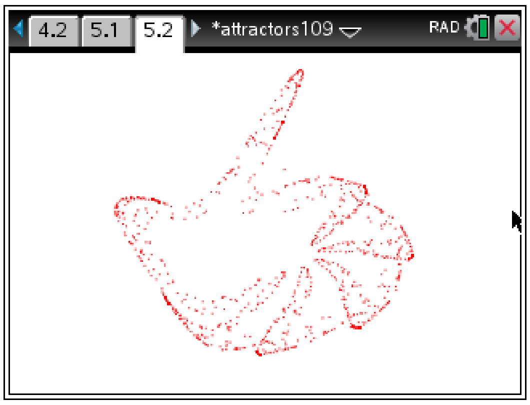
You can choose between G-M-attractors of type 1 and type 2. In case of type 2 you have to enter function *f*(*x*) as third argument. See first one input example for type 1:



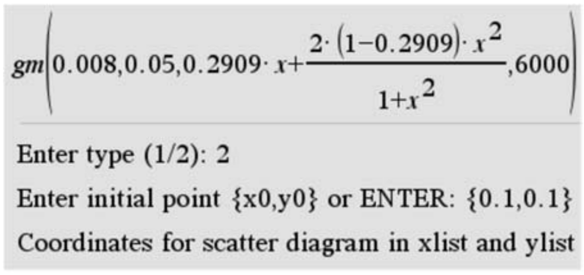
$$gm\left(0.245,1,a\cdot x+\frac{2\cdot(1-a)\cdot x^2}{1+x^2},5000\right)$$

```
Enter type (1/2): 1
Enter initial point {x0,y0} or ENTER: {1,1}
Coordinates for scatter diagram in xlist and ylist
```



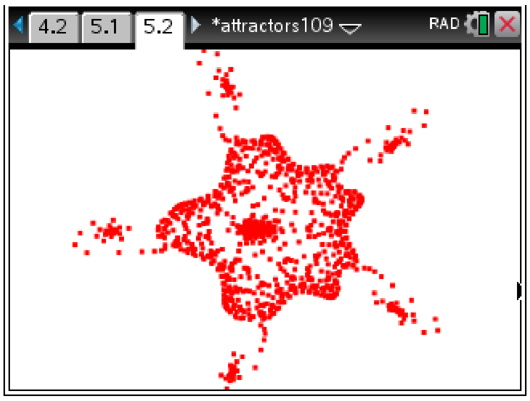


One example for GM-attractor of type 2

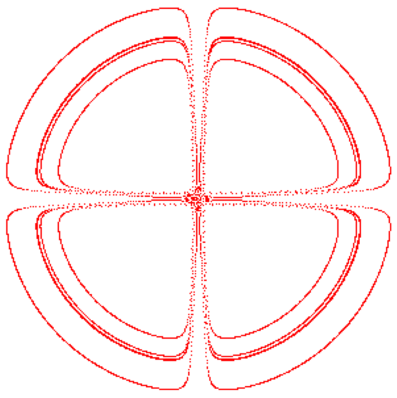


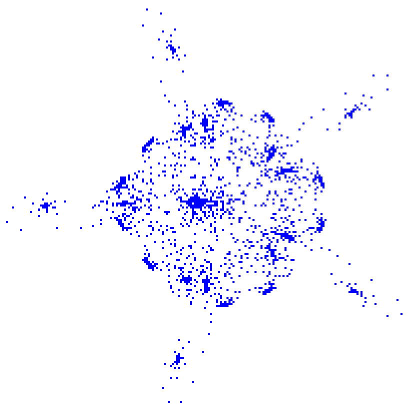
$$gm\left(0.008,0.05,0.2909\cdot x+\frac{2\cdot(1-0.2909)\cdot x^2}{1+x^2},6000\right)$$

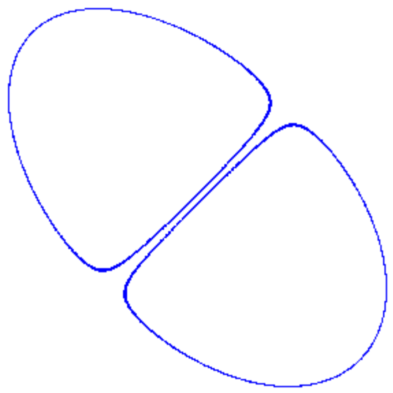
```
Enter type (1/2): 2
Enter initial point {x0,y0} or ENTER: {0.1,0.1}
Coordinates for scatter diagram in xlist and ylist
```

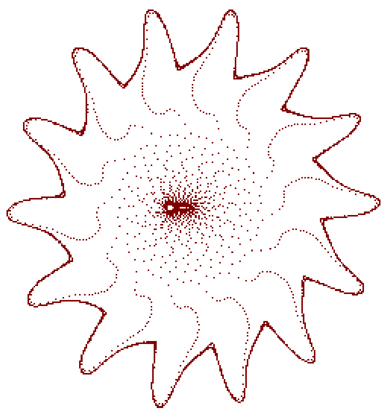


Another four attractors (gumowski2.dfw):









15 Peter De Jong-Attractors – and "My- Attractor"

And there are a couple of websites presenting Peter De Jong-attractors [5, 9, 10]. The dynamic system creating this beautiful family of strange attractors is given by

$$x_{n+1} = \sin(a \cdot y_n) - \cos(b \cdot x_n)$$
$$y_{n+1} = \sin(c \cdot x_n) - \cos(d \cdot y_n)$$

with x_0 and y_0 any real number not being both = 0.

My first DERIVE-function reads as follows:

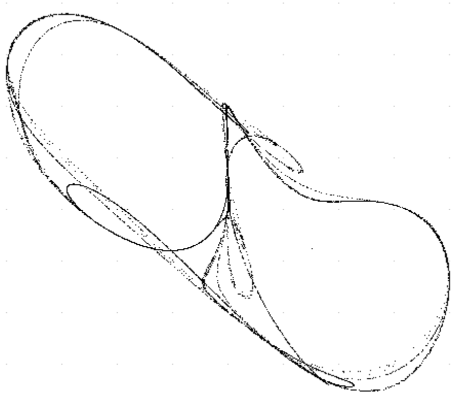
```
pdj(a, b, c, d, n, dummy, x, y, i, pts) :=
  Prog
    dummy := RANDOM(0)
    [x := RANDOM(1), y := RANDOM(1)]
    pts := [[x, y]]
    i := 1
    Loop
      If i > n exit
      x := SIN(a·y) - COS(b·x)
      y := SIN(c·x) - COS(d·y)
      pts := APPEND(pts, [[x, y]])
      i :+ 1
    pts
```

I was happy with my first strange attractor resulting from a random choice for *a*, *b*, *c*, *d*.

```
pdj(1.76, 1.67, -0.85, 2.1, 10000)
```

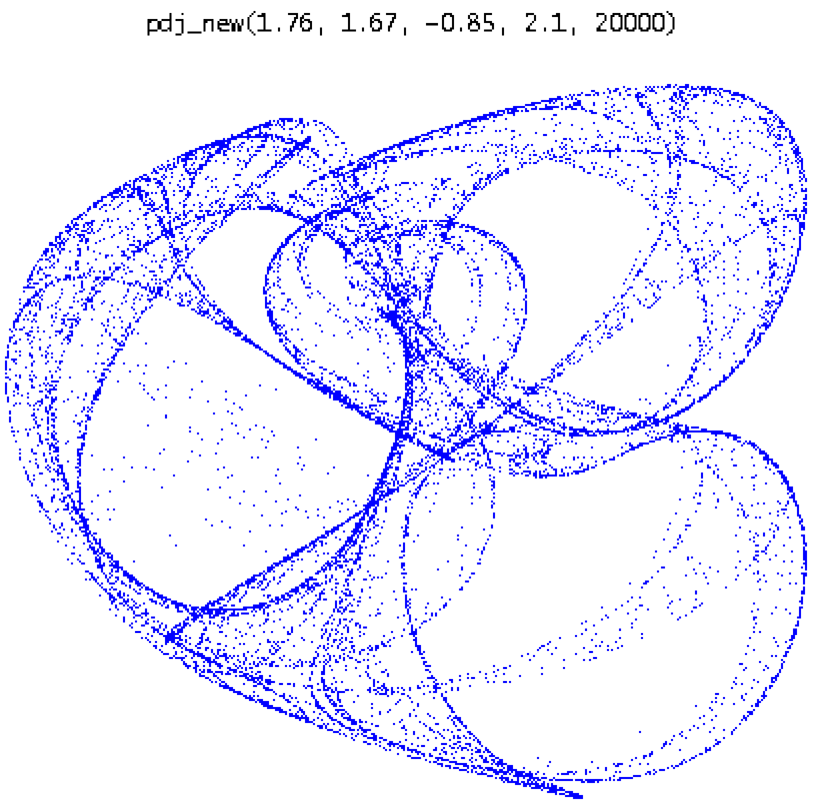
Inspecting the DERIVE-code carefully you may discover a big mistake in the algorithm. The two important lines describing the recursion are generating another dynamic system, namely

$$x_{n+1} = \sin(a \cdot y_n) - \cos(b \cdot x_n)$$
$$y_{n+1} = \sin(c \cdot x_{n+1}) - \cos(d \cdot y_n)$$


My first Peter De Jong-attractor which wasn't one

This is the true Peter De Jong-attractor function in DERIVE code followed by the respective graph.

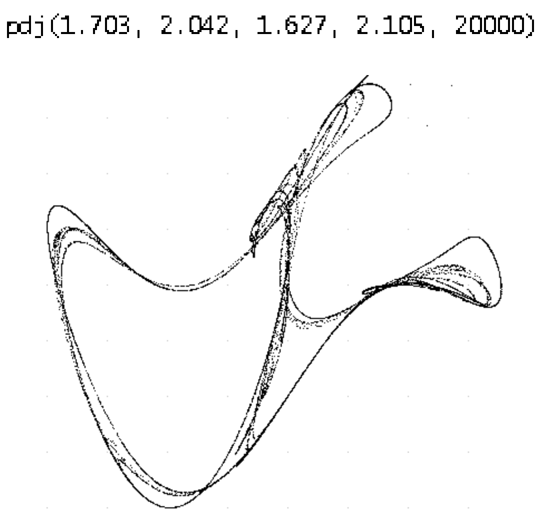
```
pdj_new(a, b, c, d, n, dummy, x0, y0, xn, yn, i, pts) :=
  Prog
    dummy := RANDOM(0)
    [x0 := RANDOM(1), y0 := RANDOM(1)]
    x0 := 0
    y0 := 0.5
    pts := [[x0, y0]]
    i := 1
    Loop
      If i > n exit
      xn := SIN(a·y0) - COS(b·x0)
      yn := SIN(c·x0) - COS(d·y0)
      pts := APPEND(pts, [[xn, yn]])
      [x0 := xn, y0 := yn]
      i :+ 1
    pts
```

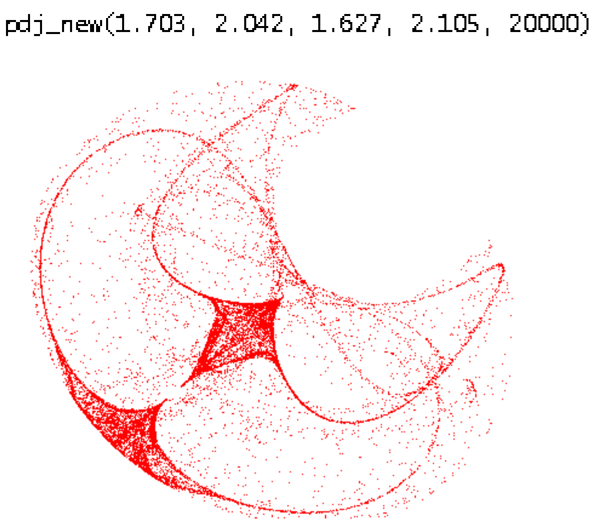


What a difference!

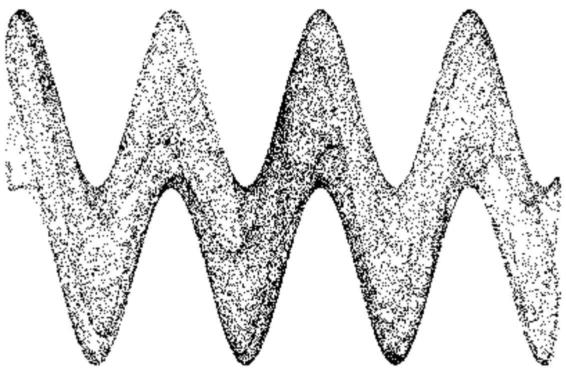
Can it really be that I found by mere chance a new attractor type – the “JB-attractor”?. This is the great and exciting side of moving in this field of mathematics. It makes you feel as a discoverer. Just experiment and play around with functions and parameter values and you will have – intentionally or not –a big chance to get a reward for your efforts.

Please compare the “JB-attractors” (left) and the respective PDJ-attractors (right) using the same parameter values.

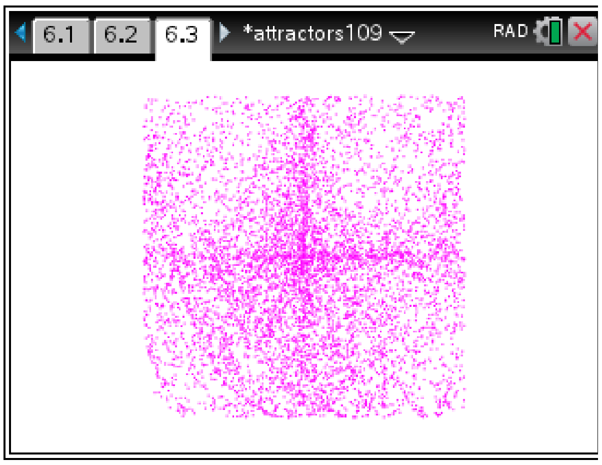




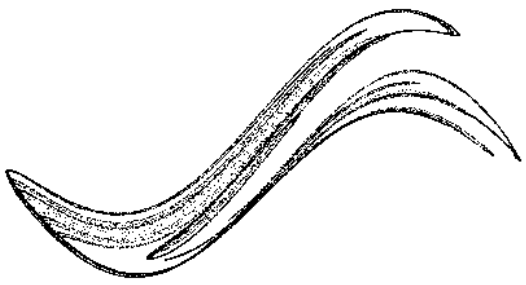
“My JB-attractor” (left) vs Peter De Jong-attractor (right) `pdj(2.814, 4.998, 5.817, 6.903, 20000)`



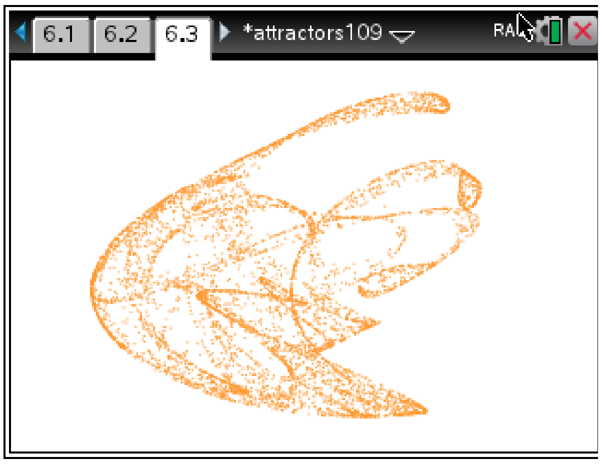
$pdj(1.4,{}^-2.3,2.4,2.1,7000)$



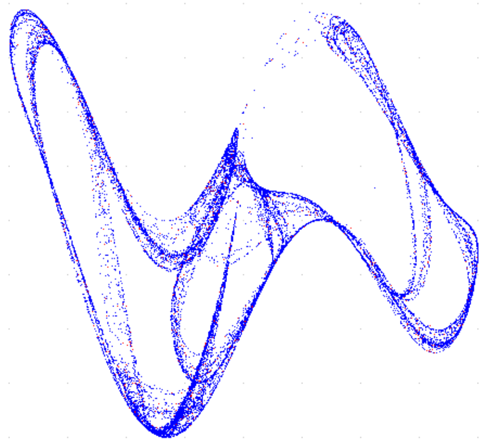
`pdj(1.916, -2.325, 1.565, 0.914, 20000)`



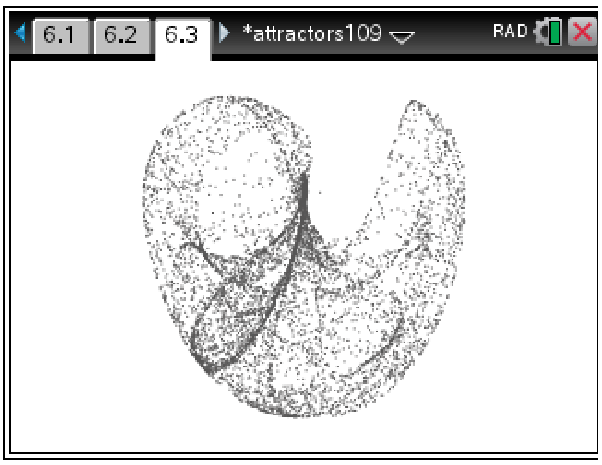
$pdj(2.814,4.998,5.817,6.903,7000)$



`pdj(1.4, -2.3, 2.4, 2.1, 20000)`



$pdj(1.916,{}^-2.325,1.565,0.914,7000)$



Some other comparisons (“true” PdJ-attractor is in the right column)

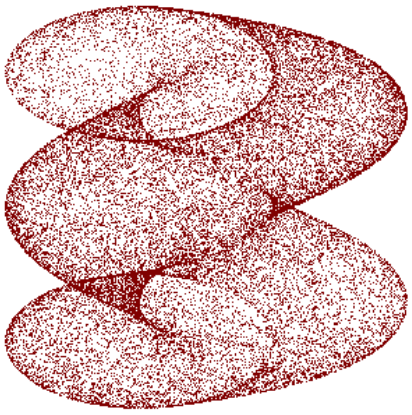
You can change the signs, the parameters, and add other functions in the PDJ-definition. Just give it a try. You may also combine the search for nice patterns with random generated sets of parameter values.

16 Clifford Pickover's Million Points

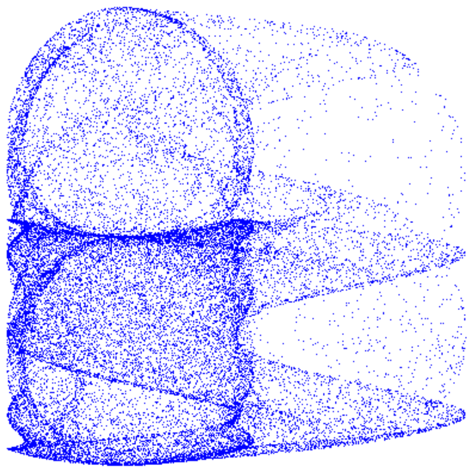
There is a chapter "One Million Points Sculptures" in Clifford Pickover's very recommendable book *Computers and the Imagination* [10] (in German: *Mit den Augen des Computers* [9]) where Pickover presents graphs consisting of one-million-point-attractors generated by dynamic systems based also on trigonometric functions.

This was challenge enough for me to try with our CAS. 10000 points are sufficiently enough for producing the "sculptures".

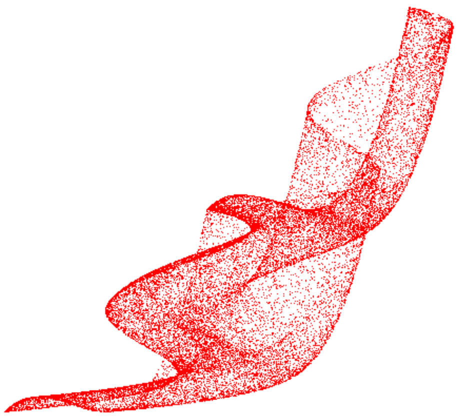
```
cliff0(a, b, n, x0, y0, xn, yn, i, pts) :=
  Prog
    [x0 := 10, y0 := 10]
    pts := [[x0, y0]]
    i := 1
    Loop
      If i > n exit
      xn := SIN(b·y0) + SIN(b·x0)^2
      yn := SIN(a·x0) + SIN(a·y0)^2
      pts := APPEND(pts, [[xn, yn]])
      [x0 := xn, y0 := yn]
      i :+ 1
    pts

cliff0(5, 10, 30000)
```



```
cliff1(a, b, n, x0, y0, xn, yn, i, pts) :=
  Prog
    [x0 := 10, y0 := 10]
    pts := [[x0, y0]]
    i := 1
    Loop
      If i > n exit
      xn := SIN(b·y0) + SIN(b·x0)^2 + SIN(b·x0)^3
      yn := SIN(a·x0) + SIN(a·y0)^2 + SIN(a·y0)^3
      pts := APPEND(pts, [[xn, yn]])
      [x0 := xn, y0 := yn]
      i :+ 1
    pts

cliff1(5, 10, 30000)
```



```
cliff2(a, b, n, x0, y0, xn, yn, i, pts) :=
  Prog
    [x0 := 10, y0 := 10]
    pts := [[x0, y0]]
    i := 1
    Loop
      If i > n exit
      xn := SIN(b·y0) + SIN(b·x0)^2 + SIN(a·x0)^3
      yn := SIN(a·x0) + SIN(b·y0)^2 + SIN(b·y0)^3
      pts := APPEND(pts, [[xn, yn]])
      [x0 := xn, y0 := yn]
      i :+ 1
    pts

cliff2(5, 10, 30000)
```



"Clifford –attractors"

`cliff0()` and `cliff1()` are two of Pickover's originals, `cliff2()` is a slight change.

Preparing this paper, I studied once more Pickover's general recipe (which in my opinion has typos - at least - in the German translation).

I believe that it shall read as follows:

$$\begin{matrix} x_{n+1}=\sum_{k=1}^{\infty}\sin^k(b\cdot f(x_n,y_n)) \\ y_{n+1}=\sum_{k=1}^{\infty}\sin^k(a\cdot g(x_n,y_n)) \end{matrix} \quad \text{with} \quad \begin{matrix} f(x_n,y_n)=p\cdot x_n+(1-p)\cdot y_n \\ g(x_n,y_n)=q\cdot x_n+(1-q)\cdot y_n \end{matrix}; \; p,q\in\{0,1\}.$$

So, we have a sum of powers of sine-functions with $b{\cdot}x_n$ or $b{\cdot}y_n$ for x_{n+1} and with $a{\cdot}x_n$ or $a{\cdot}y_n$ for y_{n+1}. According to Pickover we take only the first expressions of the sums. All initial points (x_0, y_0) lead to the same figure. I take $x_0 = 20$, $y_0 = 30$.

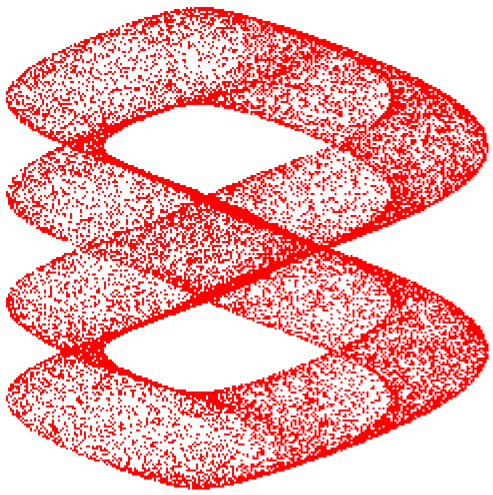
```
cliff(a, b, o, n, dummy, rx, ry, x0, y0, xn, yn, i, pts) :=
  Prog
    dummy := RANDOM(0)
    [x0 := 20, y0 := 30]
    pts := [[x0, y0]]
    rx := VECTOR(RANDOM(2), j, o)
    ry := VECTOR(RANDOM(2), j, o)
    DISPLAY(Σ(SIN((rx↓j·(xn - yn) + yn)·b)^j, j, 1, o))
    DISPLAY(Σ(SIN((ry↓j·(xn - yn) + yn)·a)^j, j, 1, o))
    i := 1
    Loop
      If i > n exit
      xn := Σ(SIN((rx↓j·(x0 - y0) + y0)·b)^j, j, 1, o)
      yn := Σ(SIN((ry↓j·(x0 - y0) + y0)·a)^j, j, 1, o)
      pts := APPEND(pts, [[xn, yn]])
      [x0 := xn, y0 := yn]
      i :+ 1
    pts
```

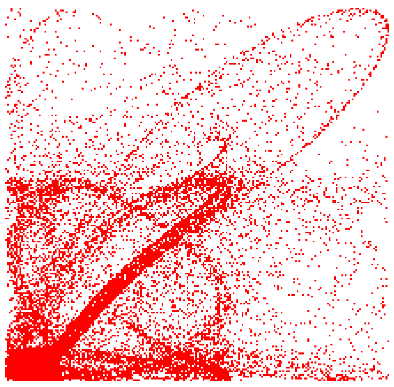
The `rx↓j` and `ry↓j` are the p and q of the formulae above and generate a random composition of the sine powers. Please observe the two `DISPLAY`– functions, which present the randomly created recursions – for later documentation and possible reproduction.

Now let's have a run:

```
#19:  cliff(5, 10, 3, 30000)

SIN(10·xn)^2 + SIN(10·yn)^3 + SIN(10·yn)

SIN(5·xn)^2 + SIN(5·yn)^3 + SIN(5·yn)
```

Plot is right, another example is below:





I asked myself: “Why not randomly choose the coefficients *a* and *b* in both recursion equations?”

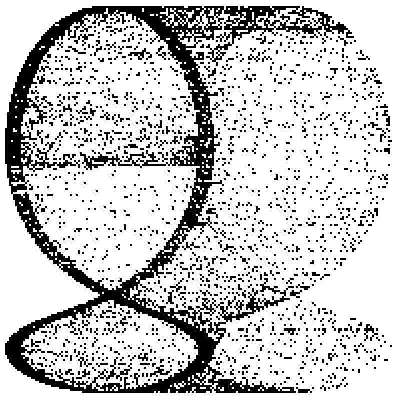
```
clifff(a, b, o, n, x0 := 20, y0 := 30, dummy, rx, ry, ra, rb, x0, y0, xn, yn, i, pts) :=
  Prog
    dummy := RANDOM(0)
    pts := [[x0, y0]]
    rx := VECTOR(RANDOM(2), j, o)
    ry := VECTOR(RANDOM(2), j, o)
    ra := VECTOR(RANDOM(2), j, o)
    rb := VECTOR(RANDOM(2), j, o)
    DISPLAY(∑(SIN((rx↓j·(xn - yn) + yn)·(ra↓j·(a - b) + b))^j, j, 1, o))
    DISPLAY(∑(SIN((ry↓j·(xn - yn) + yn)·(rb↓j·(a - b) + b))^j, j, 1, o))
    i := 1
    Loop
      If i > n exit
      xn := ∑(SIN((rx↓j·(x0 - y0) + y0)·(ra↓j·(a - b) + b))^j, j, 1, o)
      yn := ∑(SIN((ry↓j·(x0 - y0) + y0)·(rb↓j·(a - b) + b))^j, j, 1, o)
      pts := APPEND(pts, [[xn, yn]])
      [x0 := xn, y0 := yn]
      i :+ 1
    pts
```

The first try resulted in a very nice attractor.

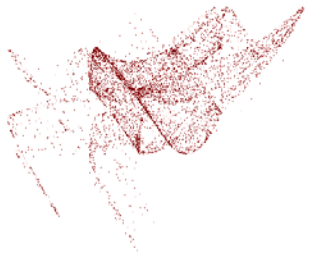
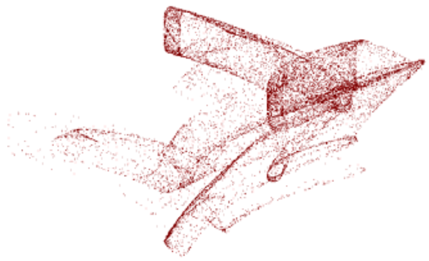
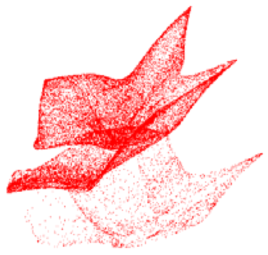
```
#22:  clifff(5, 10, 3, 30000)

SIN(10·xn)^3 + SIN(10·xn)^2 + SIN(10·yn)

SIN(5·yn)^3 + SIN(5·yn)^2 + SIN(5·yn)
```

The random number generator gave only *b*s for x_{n+1} and only *a*s for y_{n+1}.

The attractor is pretty “attractive”, isn’t it?



```
#24:  clifff(1, 3, 5, 30000)

SIN(xn)^5 + SIN(xn)^3 + SIN(3yn)^4 + SIN(3·yn)^2 + SIN(3·yn)

SIN(yn)^4 + SIN(yn) + SIN(3·xn)^2 + SIN(3·yn)^5 + SIN(3·yn)^3

#25:  clifff(1, 3, 5, 30000)

SIN(xn)^2 + SIN(xn) + SIN(3·xn)^5 + SIN(3·xn)^3 + SIN(3·yn)^4

SIN(xn) + SIN(yn)^5 + SIN(yn)^4 + SIN(3·xn)^2 + SIN(3·yn)^3

#26:  clifff(1, 3, 5, 30000)

SIN(yn)^2 + SIN(3·xn)^5 + SIN(3·xn) + SIN(3·yn)^4 + SIN(3·yn)^3

SIN(yn)^5 + SIN(yn) + SIN(3·xn)^3 + SIN(3·xn)^2 + SIN(3·yn)^4
```

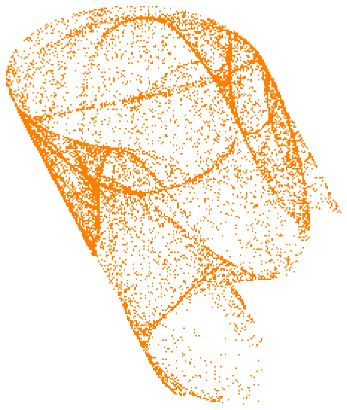
I tried to find some more details about this family of attractors in the web and so I came across the site

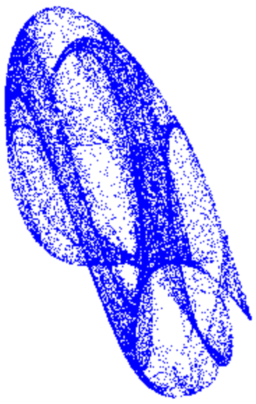
https://fronkonstin.com/2017/11/07/drawing-10-million-points-with-ggplot-clifford-attractors/

where a certain “*Fronkonstin*” used a different pair of equations generating a Clifford-attractor:

```
cliff4(a, b, c, d, n, x0, y0, xn, yn, i, pts) :=
  Prog
    [x0 := 10, y0 := 10]
    pts := [[x0, y0]]
    i := 1
    Loop
      If i > n exit
      xn := SIN(a·y0) + c·COS(a·x0)
      yn := SIN(b·x0) + d·SIN(b·y0)
      pts := APPEND(pts, [[xn, yn]])
      [x0 := xn, y0 := yn]
      i :+ 1
    pts
```

```
cliff4(-1.244580466, -1.251918341, -1.81590817, -1.908667352, 30000)
```



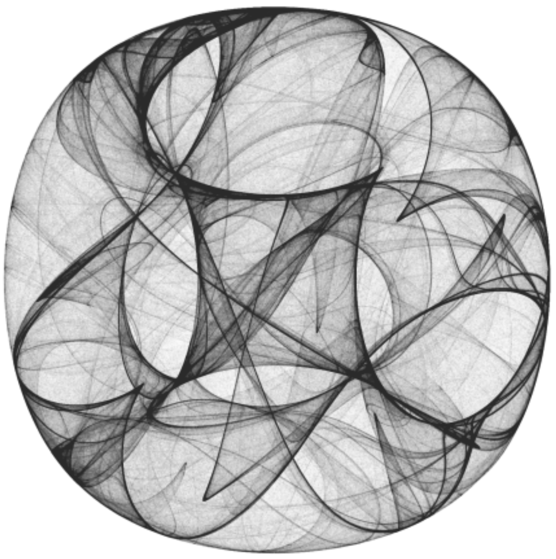


```
cliff4(-1.244580466, -1.81590817, -1.251918341, -1.908667352, 30000)
```

The first call of cliff4 shows the first 30 000 points of his graph, the second one with two parameters exchanged give the blue graph.

What you can see right is the result of 10 million points finally rendered with an extra software.

There are some other great 10 million-point plots on the site.



More nice graphs can be found at:

https://fronkonstin.com/2014/10/13/beautiful-curves-the-harmonograph/
http://www.walkingrandomly.com/?p=151

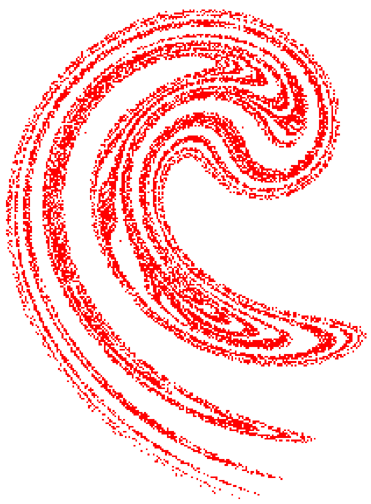
17 Finally, the last one – the Ikeda-Attractor

The Ikeda-system [9, 10] is given by:

$$x_{n+1} = a + b \cdot (x_n \cos t - y_n \sin t)$$
$$y_{n+1} = b \cdot (x_n \sin t + y_n \cos t)$$
$$t = c - \frac{d}{x_n^2 + y_n^2 + 1}$$

The system is insensitive with regard to the initial values. We can take any real numbers in order to receive the attractor.

```
ikeda(a, b, c, d, n, x0 := 0.1, y0 := 0.1, pts, i, xn, yn, t) :=
  Prog
    i := 1
    pts := [[x0, y0]]
    Loop
      If i > n
         RETURN pts
      t := c - d/(x0^2 + y0^2 + 1)
      xn := a + b·(x0·COS(t) - y0·SIN(t))
      yn := b·(x0·SIN(t) + y0·COS(t))
      pts := APPEND(pts, [[xn, yn]])
      If ABS(xn) > 10000
         xn := 0
      If ABS(yn) > 10000
         yn := 0
      [x0 := xn, y0 := yn]
      i :+ 1

ikeda(1, 0.9, 0.4, 6, 20000)
```

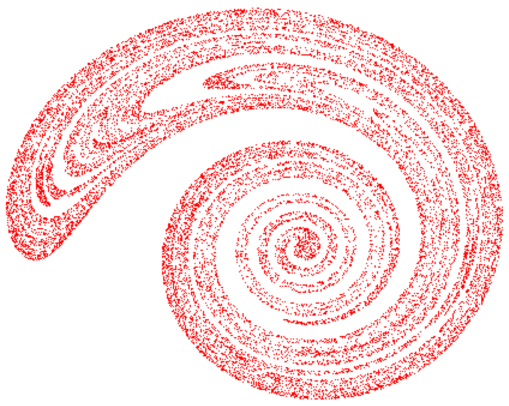


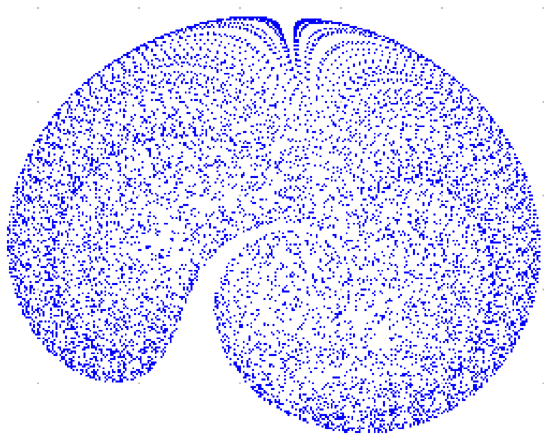
Ikeda attractor

It needs only exchanging two signs to receive other exciting forms:

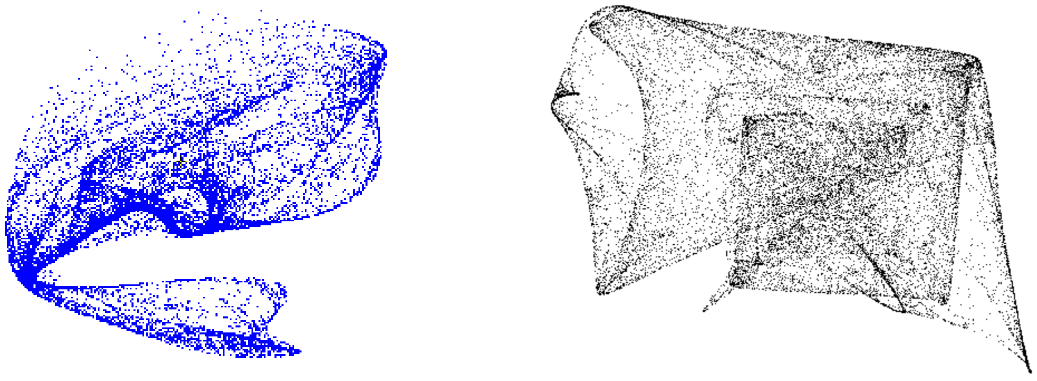
```
xn := a + b·(x0·COS(t) + y0·SIN(t)),
yn := b·(x0·SIN(t) - y0·COS(t)),
```

```
ikeda0(2.2, 0.9, 4, 30.5, 20000)          ikeda0(2.98, 0.99, 3.26, 47.74, 10000)
```





Ikeda-attractor with exchanged signs



Ikeda-related attractors

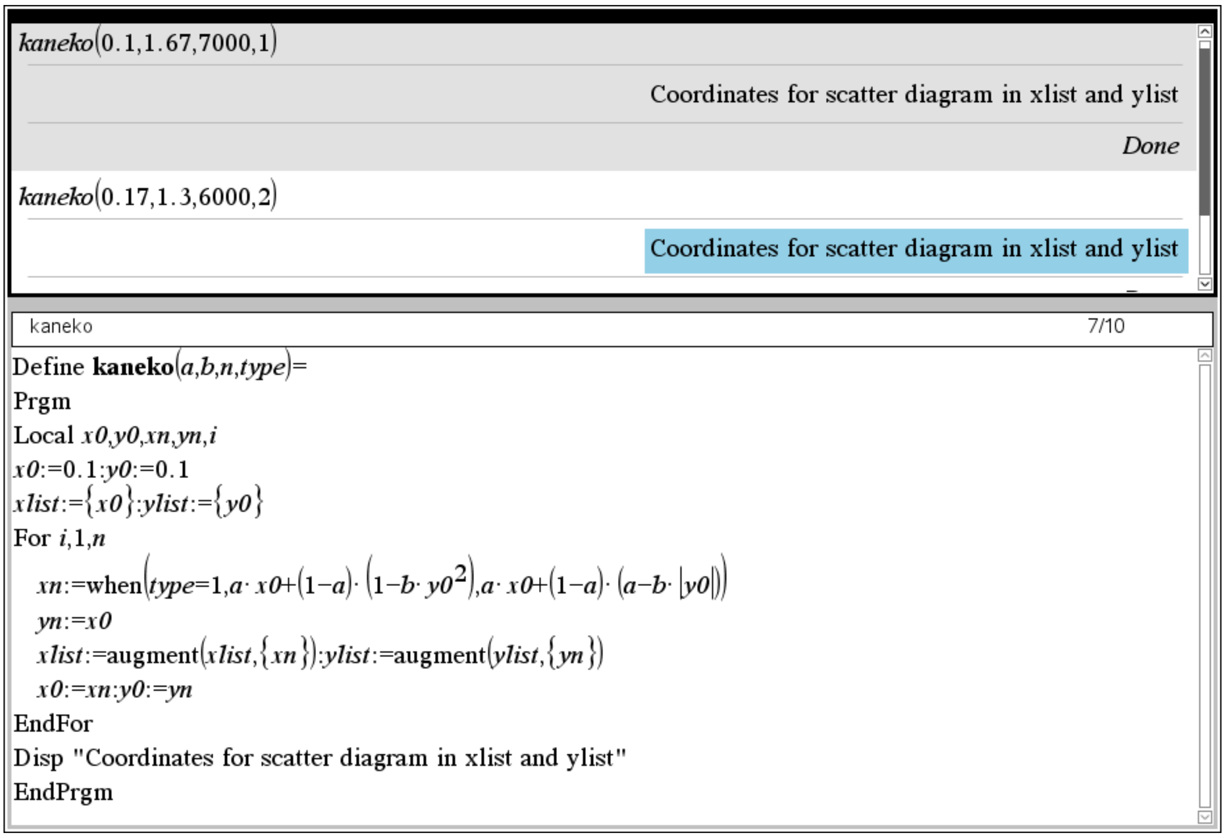
18 I cannot resist …

… to add two more attractors. There are so many resources on Chaos and Fractals in the web and by mere chance I came across "Fraktalwelt" (maintained by *Ulrich Schwebinghaus*, see under the references) where two more fractal types are presented: Kaneko attractor and Martin attractor. They were not included in my original Strange Attractors paper.

I decided to add them here in order to have two TI-Nspire treatments as final chapter in my series of contributions.

I'll start with the Kaneko-attractor (*Dr. Kunihiko Kaneko, University of Tokyo*).

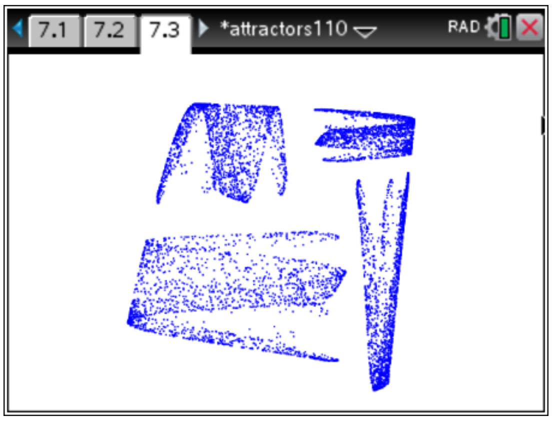
I believe that it is not necessary to give explicitly the recursion formulae. They are easy to read off from the program code. The attractors do not depend on initial values. So, we need only to enter the values for parameters *a* and *b*, the number of iterations and the attractor type (1/2).

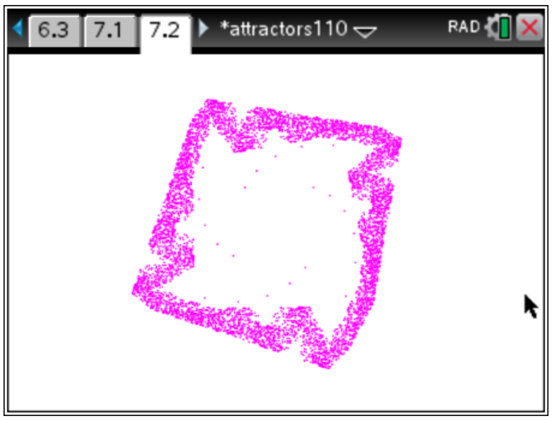


```
kaneko(0.1,1.67,7000,1)
                    Coordinates for scatter diagram in xlist and ylist
                                                                  Done
kaneko(0.17,1.3,6000,2)
                    Coordinates for scatter diagram in xlist and ylist
```

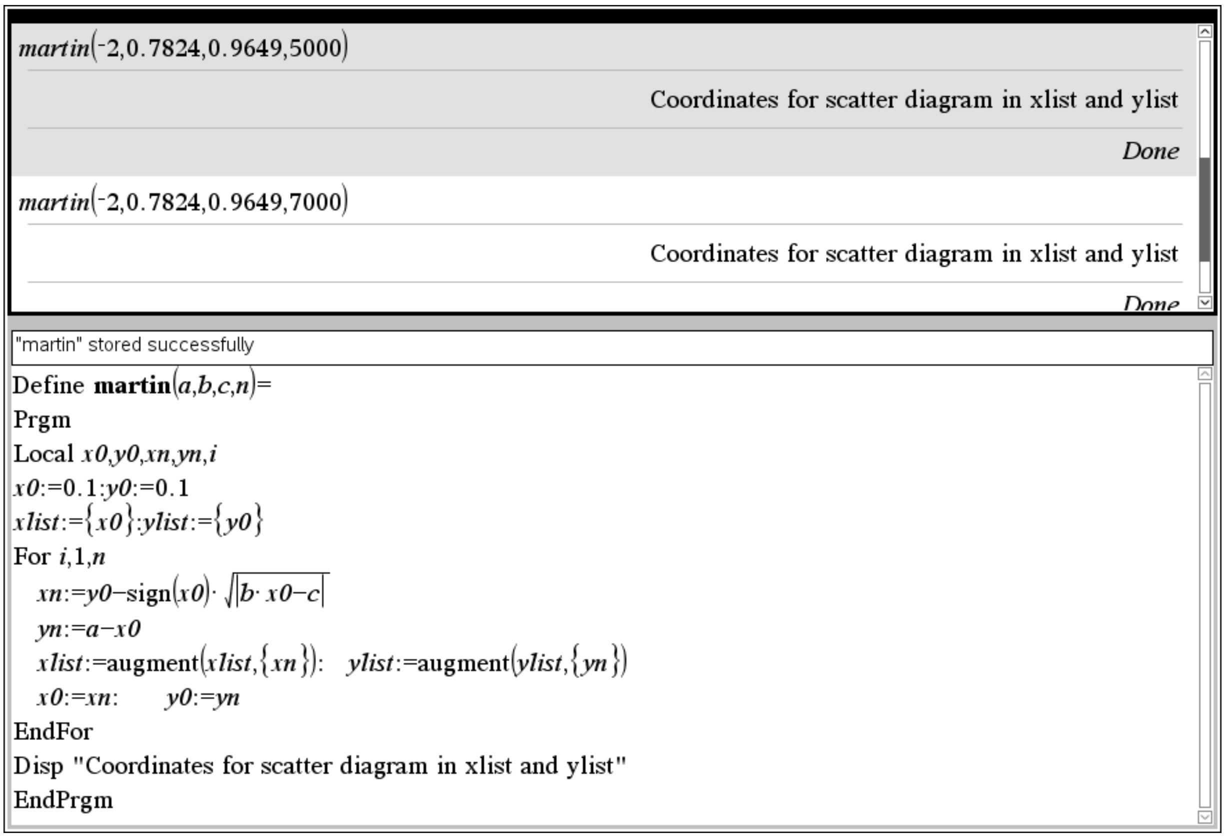
```
kaneko                                                            7/10
Define kaneko(a,b,n,type)=
Prgm
Local x0,y0,xn,yn,i
x0:=0.1:y0:=0.1
xlist:={x0}:ylist:={y0}
For i,1,n
  xn:=when(type=1,a·x0+(1−a)·(1−b·y0^2),a·x0+(1−a)·(a−b·|y0|))
  yn:=x0
  xlist:=augment(xlist,{xn}):ylist:=augment(ylist,{yn})
  x0:=xn:y0:=yn
EndFor
Disp "Coordinates for scatter diagram in xlist and ylist"
EndPrgm
```

Some parameters which will give nice graphs are provided on the website.





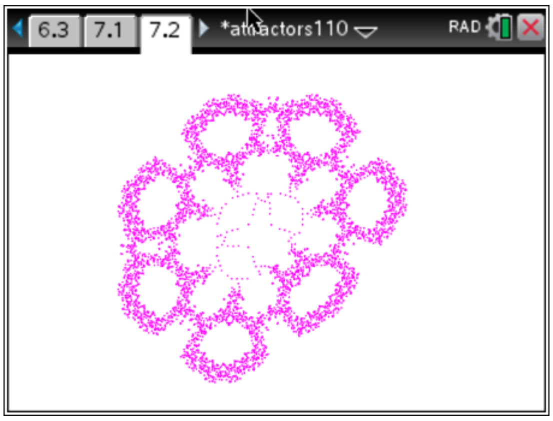
Then we have the Martin-attractor (the formula given on the website is not correct!):

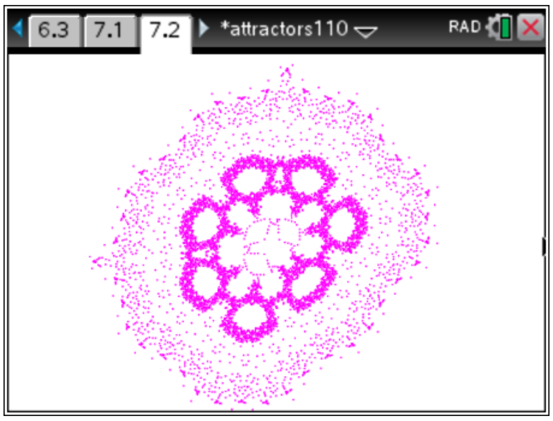


```
martin(⁻2,0.7824,0.9649,5000)
                    Coordinates for scatter diagram in xlist and ylist
                                                                  Done
martin(⁻2,0.7824,0.9649,7000)
                    Coordinates for scatter diagram in xlist and ylist
                                                                  Done
```

```
"martin" stored successfully
Define martin(a,b,c,n)=
Prgm
Local x0,y0,xn,yn,i
x0:=0.1:y0:=0.1
xlist:={x0}:ylist:={y0}
For i,1,n
  xn:=y0−sign(x0)·√(|b·x0−c|)
  yn:=a−x0
  xlist:=augment(xlist,{xn}):   ylist:=augment(ylist,{yn})
  x0:=xn:      y0:=yn
EndFor
Disp "Coordinates for scatter diagram in xlist and ylist"
EndPrgm
```

Unfortunately, there is a typo on the website: the expression for x_n is given without the absolute value under the root – this does not work.



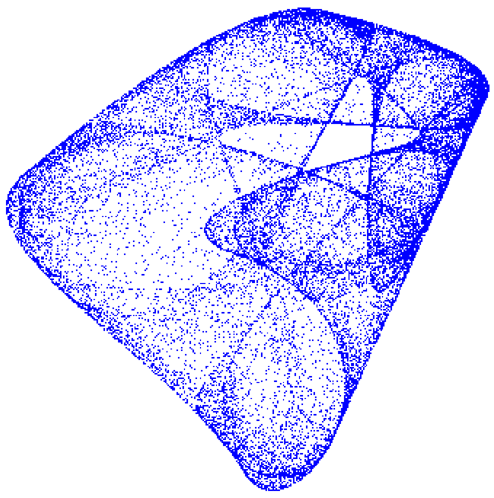


Both plots above show the same attractor with different number of iterations.

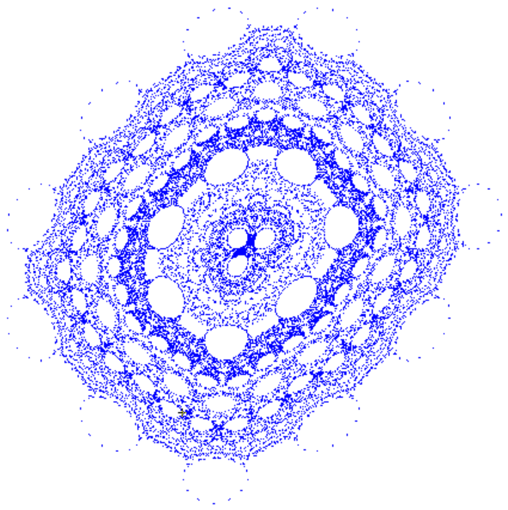
It is a good idea to test the programs with the parameters provided by *Schwebinghaus* but the real adventure will start when you try to explore the world of attractors by trying own parameters being aware that you might be the first person admiring your product. Try also to vary the recursion equations – maybe that you find another type of attractors – good luck.

Now I will really close with two DERIVE produced attractors using parameters which are not given on the website.

```
#12:  kaneko(0.45, 3.5, 1, 30000)
```



```
#16:  martin(-3, 5, 10, 30000)
```

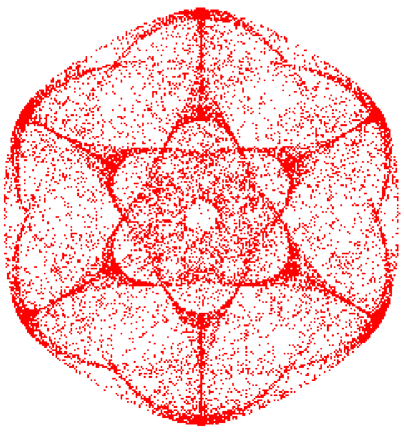


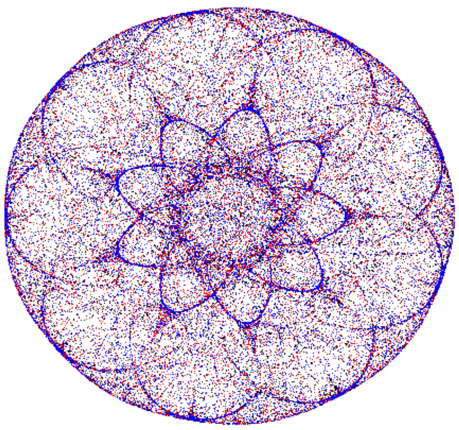
19 First Summary

Producing, admiring and discussing strange attractors offers another approach to mathematics. It is not so easy to convince ordinary secondary school students that a theorem and/or its proof is “beautiful”. These attractors are beautiful and the students can feel as discoverers and creators. This may change their attitude towards mathematics and can be a motivation for the “hard facts”, too.

Students can be asked for internet research. Let them find other attractors – there are a lot more. You can use several tools – not only DERIVE – for receiving satisfying results.

Below are two examples of my further search in the web:

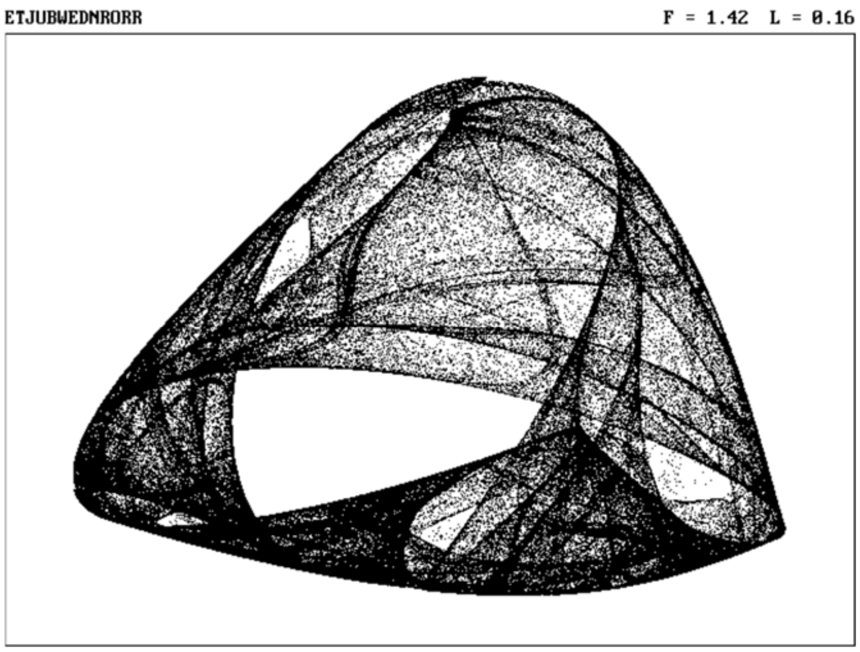




For further investigations: symmetric *Golubitsky*-attractors

20 The Duffing-Oscillator

If you run Sprott's SA-program or read his book then you can find two numbers together with the generated attractors as depicted below: L is the now well-known Lyapunov-exponent and F is the *Fractal Dimension* which we have not addressed until now. This shall change.



For this "Strange Attractor" its Lyapunov-exponent is L = 0.16 and its Fractal Dimension is F = 1.42.

But let me tell you how it began: I subscribed the free *Electronical Journal for Mathematics & Technology* (https://php.radford.edu/~ejmt/) and investigated the *ContentIndex*. So, I came across a title which attracted – attractors again! – me: *Chaotic dynamics with Maxima* written by two Mexican mathematicians (*Antonio Morante* and *José A. Vallejo*). (https://arxiv.org/abs/1301.3240)

As I have some – not too much – experience with Maxima I downloaded the pdf-file and was very enthusiastic because I found an approach for calculating the fractal dimension together with something which was absolutely new for me, the Poincaré-sections.

Fortunately, not only the explications and descriptions are given in this paper, but also the Maxima-code. It was a challenge to "translate" this code into the DERIVE- and TI-Nspire-language and – by the way – to learn something new.

I skip the first eight paragraphs of the paper which treat the logistic equation, Feigenbaum-diagram, Lorenz attractor, Lyapunov exponent and the differential equation $\ddot{x}(t) = -\frac{1}{4}x^3(t) + x(t) - \frac{1}{10}\dot{x}(t)$. Then the authors add an oscillating force in form of a sine-function and describe:

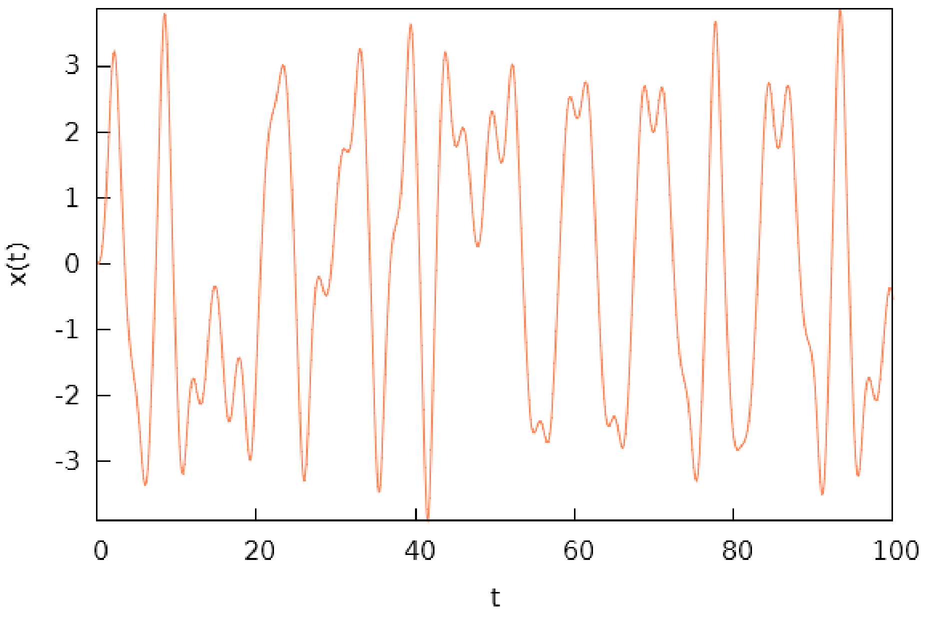
We start solving the differential equation numerically

$$\ddot{x}(t) = -\frac{1}{4}x^3(t) + x(t) - \frac{1}{10}\dot{x}(t) + 2.5\cdot\sin(2t).$$

And proceed plotting its solution (%t20) and its phase diagram (%t21):

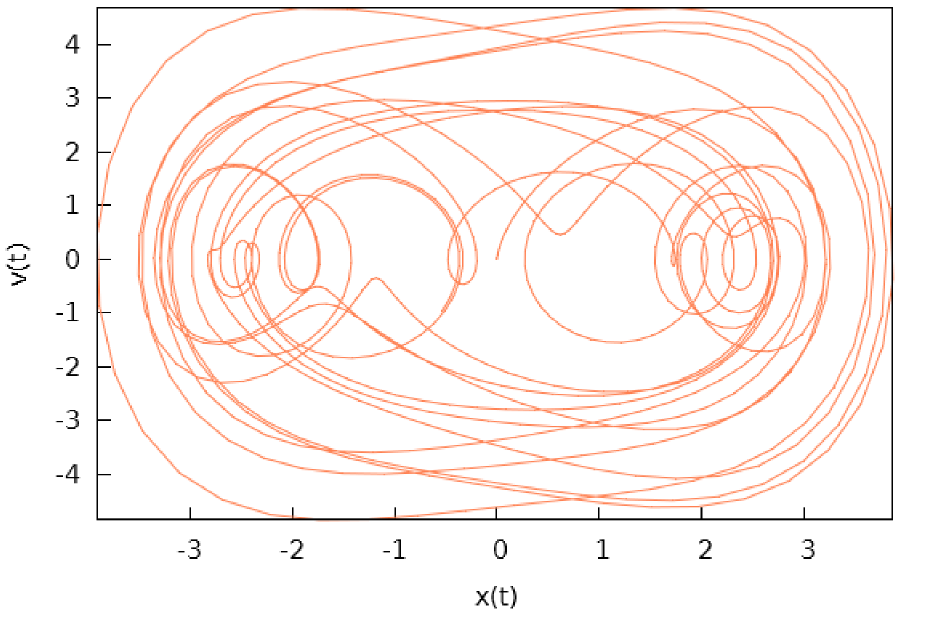
```
(%i15)  duff1:[v,-v/10+x-x^3/4+2.5*sin(2*t)]$
(%i16)  icduff1:[0,0]$;
(%i17)  sduff1:rk(duff1,[x,v],icduff1,[t,0,100,0.1])$;
(%i18)  cduff1:map(lambda([x],rest(x,-1)),sduff1)$;
(%i19)  pduff1:map(lambda([x],rest(x)),sduff1)$;
(%i20)  wxdraw2d(point_type=none,points_joined=true,color = coral,
            xlabel="t",ylabel="x(t)",points(cduff1));
```

(%t20)



```
(%i21)  wxdraw2d(point_type=none,points_joined=true,color = coral,
            xlabel="x(t)",ylabel="v(t)",points(pduff1));
```

(%t21)

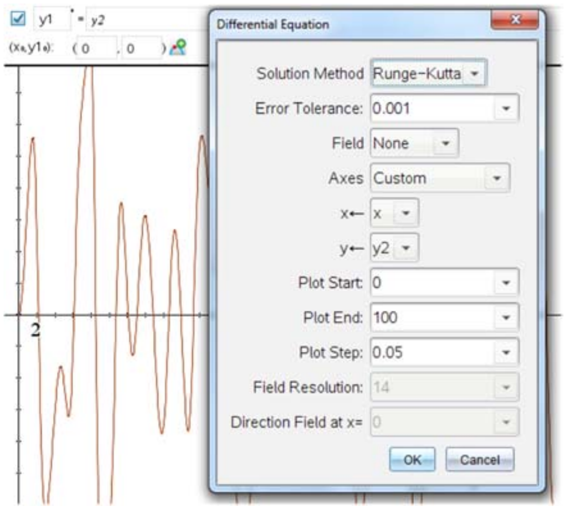


For editing the 2nd order DE it is necessary to rewrite it as a system of two differential equations.

This is the advice how to do given in the DERIVE Online-Help:

> A second or higher order ordinary differential equation can always be transformed into an equivalent system of first order differential equations as follows:
>
> First, introduce a unique new variable for each derivative except the highest.
>
> Next, replace the highest order derivative with the first derivative of the variable that represents the next-highest order derivative.
>
> Finally, for each of the other variables, add to the system an equation that equates its first derivative to the variable that represents the next higher order derivative.
>
> For example, this procedure transforms a second order equation of the form
> f (x, y", y', y) = 0 into the system of first order equations f (x, v', v, y) = 0 and y' = v.

I followed this hint considering the appropriate notification of the variables (compare with Maximas's `duff1:[v,-v/10+x-x^3/4+2.5*sin(2*t)])` and got satisfying results in form of impressive plots on the TI-NspireCAS-screen.

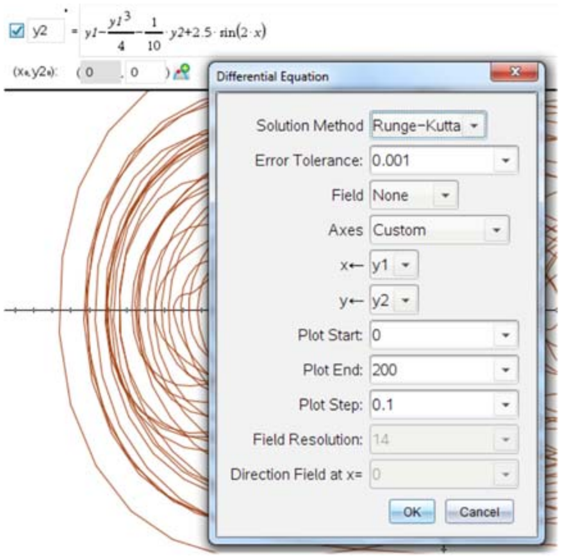


$y1' = y2$

$(x_0, y1_0)$: (0 , 0)

Differential Equation

Solution Method: Runge-Kutta
Error Tolerance: 0.001
Field: None
Axes: Custom
x←: x
y←: y2
Plot Start: 0
Plot End: 100
Plot Step: 0.05
Field Resolution: 14
Direction Field at x=: 0



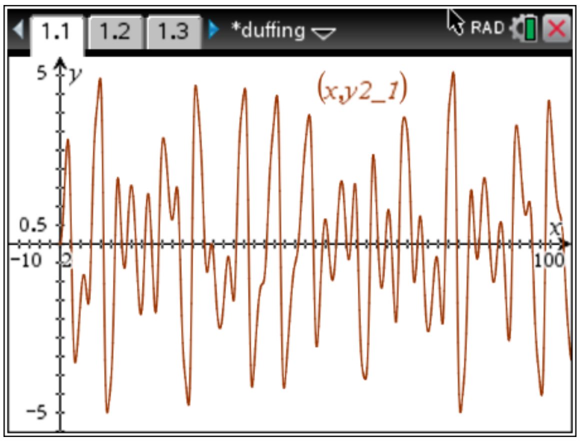
$y2' = y1-\frac{y1^3}{4}-\frac{1}{10}\cdot y2+2.5\cdot\sin(2\cdot x)$

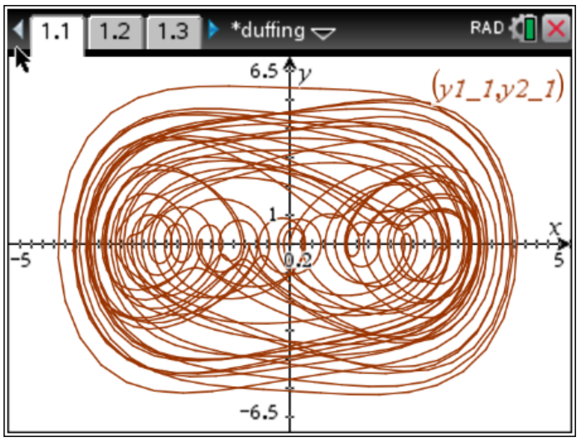
$(x_0, y2_0)$: (0 , 0)

Differential Equation

Solution Method: Runge-Kutta
Error Tolerance: 0.001
Field: None
Axes: Custom
x←: y1
y←: y2
Plot Start: 0
Plot End: 200
Plot Step: 0.1
Field Resolution: 14
Direction Field at x=: 0

In order to save space I print the handheld screens:

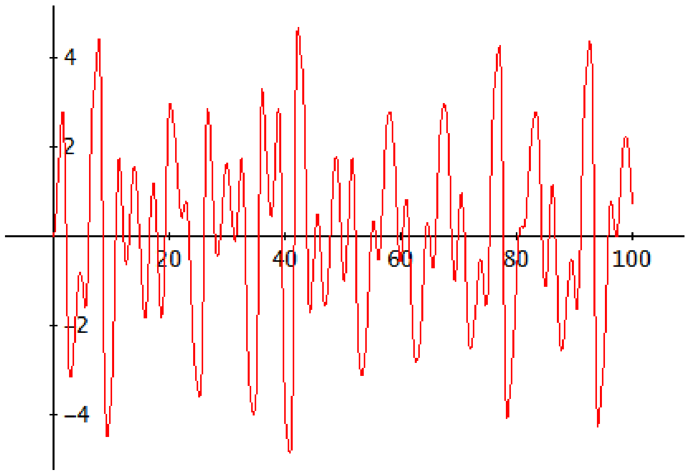




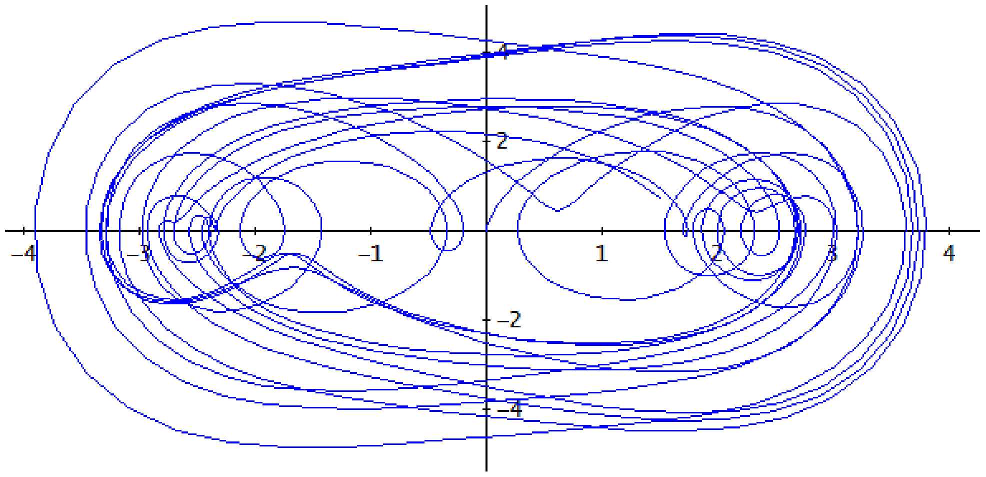
Following once more the instructions it is no problem to present the oscillator on the DERIVE-screen:

$$\#7: \quad \text{duffing1} := \text{RK}\left(\left[v,\ -\frac{x^3}{4} + x - \frac{v}{10} + 2.5\cdot\text{SIN}(2\cdot t)\right],\ [t,\ x,\ v],\ [0,\ 0,\ 0],\ 0.1,\ 1000\right)$$

```
#8:   duffing1 COL [1, 3]
```



```
#9:   duffing1 COL [2, 3]
```



21 The Poincaré-Section

The second part is a little bit harder to perform. Poincaré proposed a technique to illustrate the special dynamics of this solution.

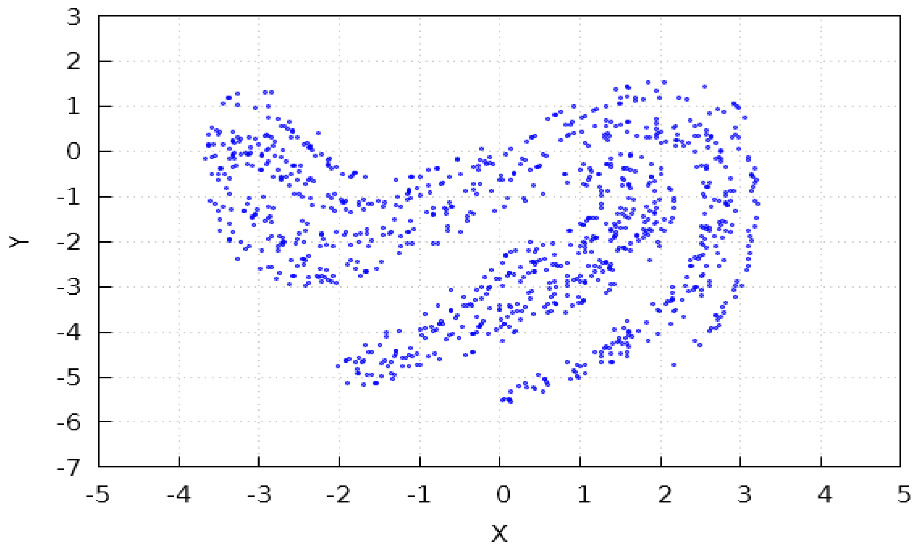
He introduced stops equally spaced in time and fixed the respective points. The frequency of our oscillator is 2, hence the period is $T = \frac{2\pi}{2} = \pi$. What we do now is the following: we take every moment of the phase diagram when a full period is done and plot the respective point.

We produce the phase diagram for 1000 periods applying a step width of π/30 and plot every thirtieth point – and surprisingly we see an attractor evolving - as a consequence of the chaotic behavior of the solution as shown above.

This is how the two Mexican mathematicians did using Maxima:

```
(%i30)  τ:bfloat(π);
(τ)     3.141592653589793b0
(%i31)  maxiter2:1000$;
(%i32)  sduff3:rk(duff1,[x,v],icduff1,[t,0,maxiter2*τ,τ/30])$;
(%i33)  pduff3:create_list(sduff3[i],i,makelist(i*30,i,1,maxiter2))$;
(%i34)  ptsduff3:map(lambda([x],rest(x)),makelist(pduff3[i],i,1,
        maxiter2))$;
(%i35)  wxdraw2d(point_size=0.3,point_type=circle,
        color = blue,xticks=1,yticks=1,
        xrange=[-5,5] ,yrange=[-7,3],points(pduff3),grid=true)$;
```

(%t35)

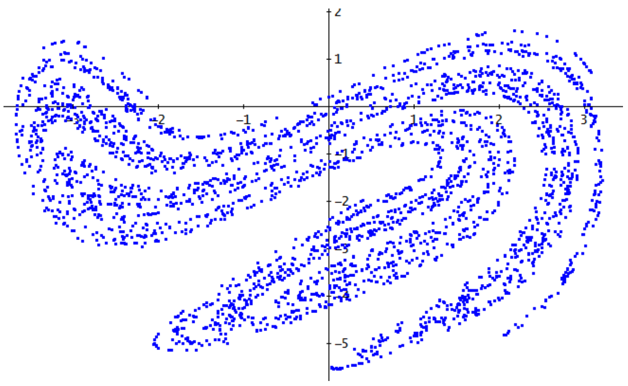


The DERIVE code is very short. I perform 90 000 RK-iteration steps which gives 3000 points of the Poincaré-section.

```
#1:   [Precision := Approximate, Notation := Scientific]
```

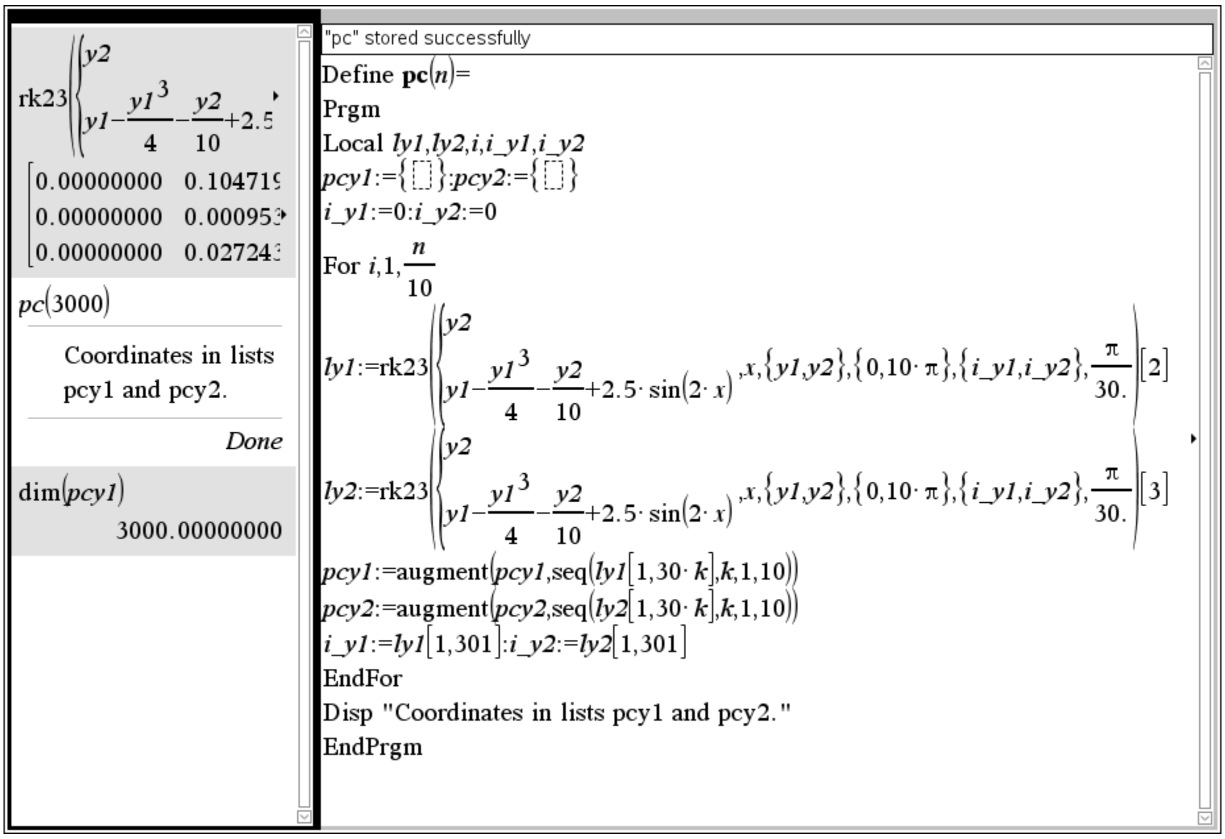
#2: $\text{duff} := \left[v,\ -\dfrac{x^3}{4} + x - \dfrac{v}{10} + 2.5\cdot \text{SIN}(2\cdot t)\right]$

```
      poincare(maxiter, sd) :=
        Prog
#3:       sd := RK(duff, [t, x, v], [0, 0, 0], π/30, maxiter·30)
          (VECTOR(sd↓(30·i), i, maxiter)) COL [2, 3]

#4:   poincare(3000)
```



With the TI-NspireCAS it is not so easy – at least for me – because of the restricted memory resources. I was not able to generate 30 000 iterations in the Calculator in order to find 1000 points of the Poincaré-section. One can create the RK-table using the rk23()-function even in the Lists & Spreadsheet App but – just in a restricted amount.

So, I found a workaround writing a program. It calculates the points of the Poincaré section in 300 steps with 10 periods each. I take the last iteration values of one period as initial conditions of the next one – and I hope that it works and that the resulting figure will match with the above ones. Give it a try and much luck:



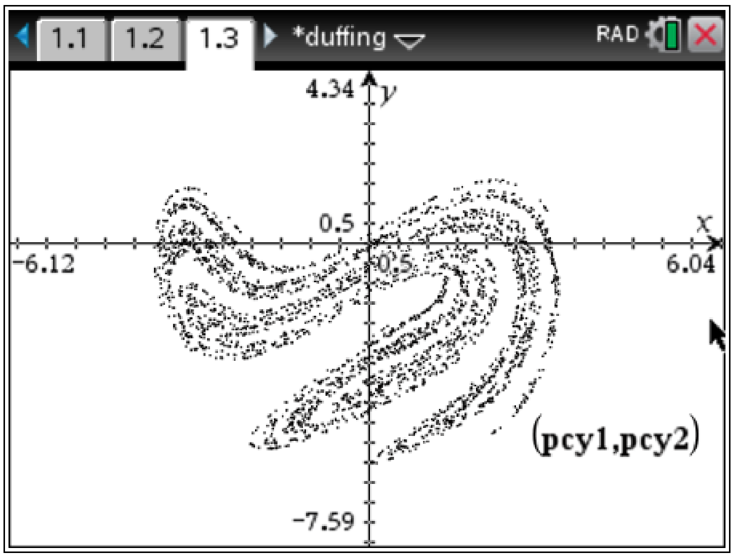
```
rk23( {y2 ; y1 − y1^3/4 − y2/10 + 2.5 ...
[0.00000000  0.10471...
 0.00000000  0.00095...
 0.00000000  0.02724...
pc(3000)
      Coordinates in lists
      pcy1 and pcy2.
                          Done
dim(pcy1)
             3000.00000000
```

```
"pc" stored successfully
Define pc(n)=
Prgm
Local ly1,ly2,i,i_y1,i_y2
pcy1:={□}:pcy2:={□}
i_y1:=0:i_y2:=0
For i,1,n/10
ly1:=rk23({y2 ; y1−y1^3/4−y2/10+2.5·sin(2·x)},x,{y1,y2},{0,10·π},{i_y1,i_y2},π/30.)[2]
ly2:=rk23({y2 ; y1−y1^3/4−y2/10+2.5·sin(2·x)},x,{y1,y2},{0,10·π},{i_y1,i_y2},π/30.)[3]
pcy1:=augment(pcy1,seq(ly1[1,30·k],k,1,10))
pcy2:=augment(pcy2,seq(ly2[1,30·k],k,1,10))
i_y1:=ly1[1,301]:i_y2:=ly2[1,301]
EndFor
Disp "Coordinates in lists pcy1 and pcy2."
EndPrgm
```

Wow, it works and it gives a satisfying plot of the section (3000 points). It is not quite the same because of different Runge-Kutta methods (RK32 and RK4).

The question could arise if all duffing oscillators give (strange) attractors as Poincaré sections.

It is now no problem for us to find an answer.



I change the oscillating force to 2.5 sin(t) ($\rightarrow$ frequency = 1 and period = 2π) and adapt `poincare()` to `poincare2()`:

$$\text{duff2} := \left[v,\ -\frac{x^3}{4} + x - \frac{v}{10} + 2.5\cdot\text{SIN}(t) \right]$$

```
poincare2(maxiter, sd) :=
  Prog
    sd := RK(duff2, [t, x, v], [0, 0, 0], π/30, maxiter·60)
    (VECTOR(sd↓(60·i), i, maxiter)) COL [2, 3]
```

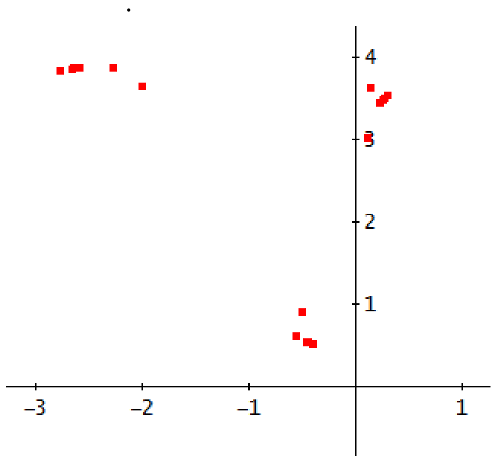
```
poincare2(500)
```

There is no strange attractor, but there are three-point attractors. That says to us that the phase diagram prefers three positions after each period (cycle).

I plot the solution together with the phase diagram and the Poincare sections (red) in one plot.

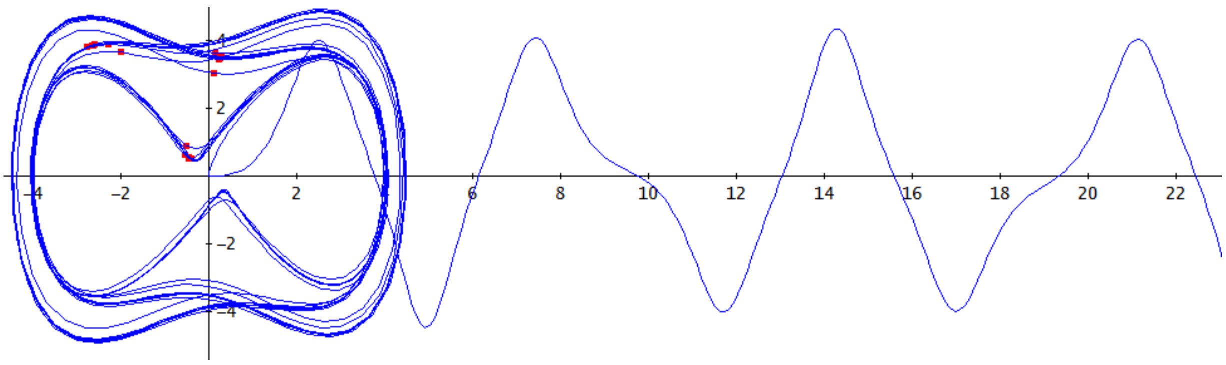
We see three cycles corresponding with the three clusters of P.S.-points.

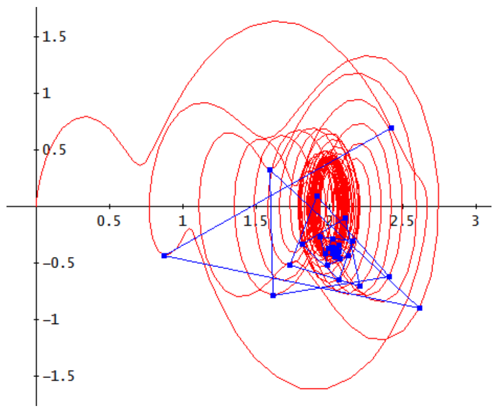
Conclusion: We cannot expect a strange attractor automatically.



$$\left(\text{RK}\left(\text{duff2},\ [t,\ x,\ v],\ [0,\ 0,\ 0],\ \frac{\pi}{30},\ 500\cdot 60\right)\right)\ \text{COL}\ [1,\ 2]$$

$$\left(\text{RK}\left(\text{duff2},\ [t,\ x,\ v],\ [0,\ 0,\ 0],\ \frac{\pi}{30},\ 500\cdot 60\right)\right)\ \text{COL}\ [2,\ 3]$$





Here is another variation of the oscillator tending to one limit cycle and the section points tending to one-point attractor. You can follow the connecting lines.

$$\text{duff3} := \left[v,\ -\frac{x^3}{4} + x - \frac{v}{10} + 1.5\cdot\text{SIN}(4\cdot t) \right]$$

```
poincare3(500)
```

$$\left(\text{RK}\left(\text{duff3},\ [t,\ x,\ v],\ [0,\ 0,\ 0],\ \frac{\pi}{30},\ 500\cdot 30\right)\right)\ \text{COL}\ [2,\ 3]$$

22 The Fractal Dimension

A fractal dimension is an index for characterizing fractal patterns or sets by quantifying their complexity as a ratio of the change in detail to the change in scale.

There are several ways to define a fractal dimension. Morante and Vallejo apply the *Box-Counting dimension*. For this purpose, the attractor will be enclosed in a box which is in our case a grid of 10 x 10 boxes ([-5,5] × [-7,3]). The authors proceed by dividing this box further in 20 x 20, 30 x 30, …, 200 x 200 boxes and count how many of the boxes contain at least one point of the attractor.

So, we have a sequence of k scales ε_k (10,20,30, …, 200) and a corresponding sequence $N(k)$ of point containing grid boxes. The box-counting dimension D is then defined as

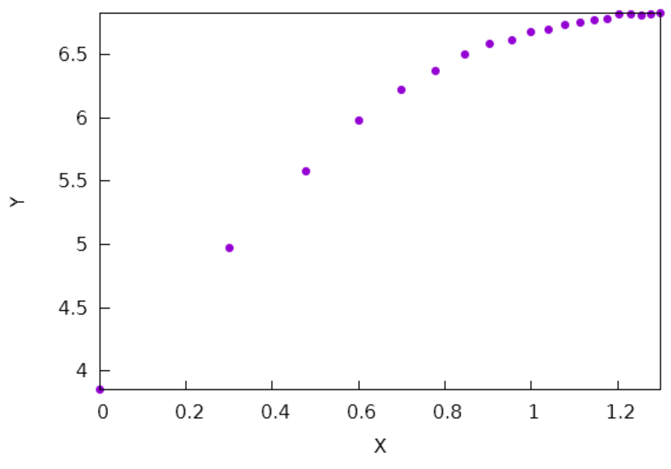
$$D = \lim_{k\to\infty} \frac{\log N(k)}{\log \varepsilon_k}.$$

Morante and Valeja plot log $N(k)$ versus ε_k and estimate D as the slope of the respective linear regression line.

I believe that it will be the best to show now the Maxima-procedure first:

```
(%i36)  resolution:20$
(%i37)  for n:1 thru resolution do
          (Z[n]:substpart("[",zeromatrix(10*n,10*n),0),
          boxcount[n]:0,
          for k:1 thru maxiter2 do
           (ix:floor(n*(ptsduff3[k][1]+5)),
            iy:floor(n*(ptsduff3[k][2]+7)),
            if is (Z[n][ix][iy]=0) then
                     (Z[n][ix][iy]:1,boxcount[n]:boxcount[n]+1)))$
(%i38)  makelist(boxcount[n],n,1,resolution);
(%o38)  [47,145,266,395,502,585,669,726,748,799,811,845,855,878,883,
         916,916,908,913,924]
(%i39)  fitdim:makelist([log(n)/log(10),log(boxcount[n])],n,1,
        resolution),numer$
(%i40)  wxdraw2d(point_size=1,point_type=filled_circle,
        color=dark_violet,points(fitdim));
```

(%t40)



The authors propose – according to some resources – to eliminate the first seven and the last two points in order to give a suitable estimation for the fractal dimension:

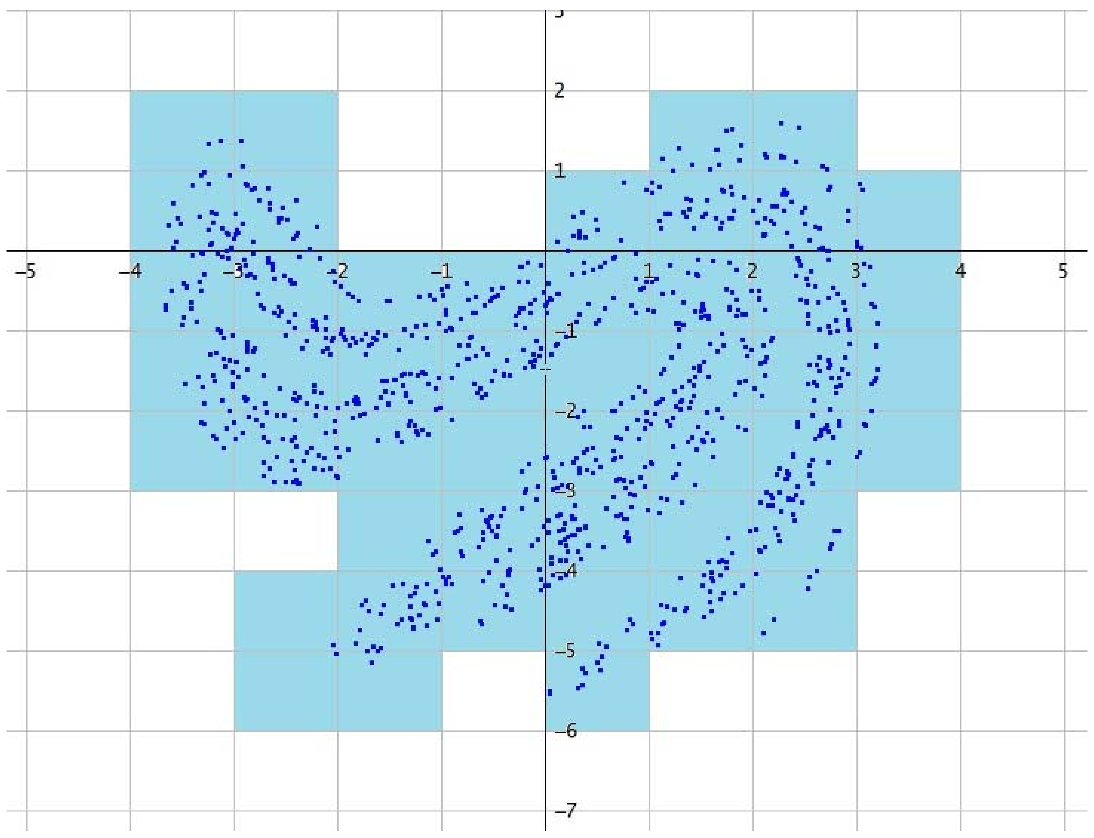
```
(%i41)   fitdimension:rest(rest(fitdim,7),-2)$

(%i42)   load(stats)$

(%i43)   model:linear_regression(fitdimension)$

(%i44)   fiteqn:take_inference('b_estimation,model);
(fiteqn) [5.987530299487617,0.6796510687848922]

(%i45)   fract_dim:second(%);
(fract_dim)0.6796510687848922
```

In the authors' opinion the fractal dimension of the Poincaré-section of the Duffing-oscillator is approximately 0.68 (slope of the regression line). I must admit that I have my concerns because I cannot explain their use log(n)/log(10) as argument for the log of the counted boxes instead of log(10n)?

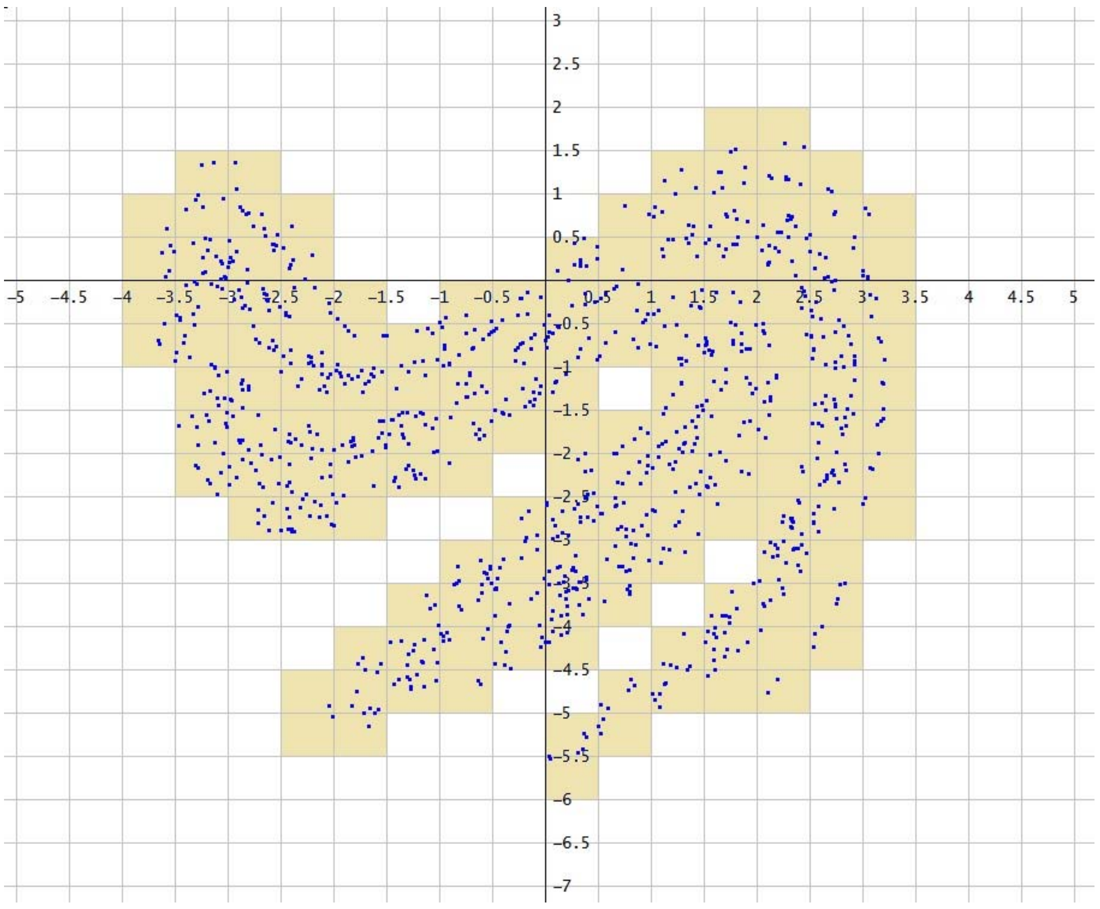
However, let's try transferring this to DERIVE:

```
       fractdim(set, res, x_range, y_range, n, ix, iy, k, k_, bcounts, z, bc) :=
         Prog
           n := 1
           bcounts := []
           k_ := DIM(set)
           Loop
             If n > res exit
             z := VECTOR(VECTOR(0, i, n·10), i, n·10)
             bc := 0
             k := 1
#16:         Loop
               If k > k_ exit
               ix := FLOOR(10·n·(set↓k↓1 - x_range↓1)/(x_range↓2 - x_range↓1)) + 1
               iy := FLOOR(10·n·(set↓k↓2 - y_range↓1)/(y_range↓2 - y_range↓1)) + 1
               If z↓(10·n + 1 - iy)↓ix = 0
                  z↓(10·n + 1 - iy)↓ix := 1
               k :+ 1
             bcounts := APPEND(bcounts, [[10·n, Σ(Σ(z))]])
             "IF(n = 1,          RETURN z)"
             n :+ 1
           bcounts'

#17:   duffpc := poincare(1000)

#18:   boxc := fractdim(duffpc, 20, [-5, 5], [-7, 3])

#19:   boxc := [ 10   20   30   40   50   60   70   80   90   100  110  120
                 48  146  262  387  492  580  644  677  757   789  803  824

          130  140  150  160  170  180  190  200 ]
          863  869  876  895  906  918  913  923 ]
```

Before proceeding I wanted to check, if my DERIVE program for counting the boxes which contain at least one attractor point works properly. I printed the set of 1000 points together with a 10 × 10 and then with a 20 × 20 grid and counted the boxes:



0	5	7	6	5	7	7	7	4	0	**Σ = 48**



Please check it! **Σ = 146**

This looks pretty good!!

Moreover, I built in a special trick for visualizing the filled grid boxes. If you remove the quotes at the end of the program you will receive the 10 × 10 grid as a matrix which represents the boxes after performing the first step.

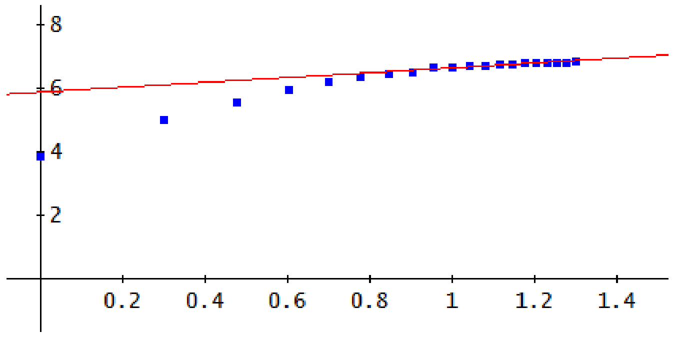
$$\#20:\quad \texttt{fractdim(duffpc, 20, [-5, 5], [-7, 3])} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

You are invited to compare this matrix with the first plot on page 74.

The number of counted boxes differ from the Maxima numbers (%o38). So, I don't expect the same result for the regression line:

$$\#21:\quad \texttt{boxc\_max} := \texttt{VECTOR}\left(\left[\frac{\texttt{LOG(i)}}{\texttt{LOG(10)}},\ \texttt{LOG}(\texttt{boxc}_{2,i})\right],\ \texttt{i},\ 20\right)$$

$$\#22:\quad \texttt{FIT}([\texttt{x},\ \texttt{a}\cdot\texttt{x} + \texttt{b}],\ \texttt{boxc\_max}_{[8,\ \ldots,\ 18]}) = 0.7644\cdot x + 5.8815$$



The fractal dimension here is ~ 0.76.

$$\#23:\quad \texttt{my\_boxc} := \texttt{VECTOR}([\texttt{LOG(10}\cdot\texttt{i)},\ \texttt{LOG}(\texttt{boxc}_{2,i})],\ \texttt{i},\ 20)$$

$$\#24:\quad \texttt{FIT}([\texttt{x},\ \texttt{a}\cdot\texttt{x} + \texttt{b}],\ \texttt{my\_boxc}_{[8,\ \ldots,\ 18]}) = 0.33186\cdot x + 5.1176$$

Using log(10*n*) as argument gives quite another result: ~0.33.

On page 72 I explained (according to all resources):

The box-counting dimension D is then defined as $D = \lim\limits_{k\to\infty} \dfrac{\log N(k)}{\log \varepsilon_k}$.

```
                 ⎛ LOG(boxc    )             ⎞
                 ⎜          2,i              ⎟
#25:   VECTOR⎜ ──────────────, i, 1, 20⎟
                 ⎝   LOG(10·i)               ⎠

#26:   [1.6812, 1.6635, 1.6371, 1.6152, 1.5844, 1.5541, 1.5223, 1.4873,

         1.4732, 1.4485, 1.4229, 1.4024, 1.3888, 1.3694, 1.3521, 1.3392,

         1.3257, 1.3137, 1.2991, 1.2886]
```

I will come back to this sequence later.

23 Changing Scale or/and Range

I changed the range for -4 ≤ x ≤ +4 and -6 ≤ y ≤ 2:

```
#27:   boxc_sm := fractdim(duffpc, 20, [-4, 4], [-6, 2])

                  ⎡ 10   20   30   40   50   60   70   80   90   100  110  120
#28:   boxc_sm := ⎢
                  ⎣ 70  197  355  492  612  660  731  789  811  852  857  876

          130  140  150  160  170  180  190  200 ⎤
                                                 ⎥
          893  903  918  923  940  940  933  949 ⎦

       my_boxc_sm := VECTOR([LOG(10·i), LOG(boxc_sm    )], i, 20)
#29:                                               2,i

       FIT([x, a·x + b], my_boxc_sm            ) = 0.21622·x + 5.7349
#30:                                [8, ..., 18]

                 ⎛ LOG(boxc_sm    )             ⎞
                 ⎜             2,i              ⎟
#31:   VECTOR⎜ ─────────────────, i, 1, 20⎟
                 ⎝    LOG(10·i)                 ⎠
```

I notice that the FD changes but the sequence of the fractal dimensions tends again to approximately 1.29, 1.30

```
#32:   [1.845, 1.7635, 1.7264, 1.6803, 1.6402, 1.5856, 1.5521, 1.5223,

         1.4885, 1.4652, 1.4367, 1.4152, 1.3958, 1.3772, 1.3615, 1.3452,

         1.3329, 1.3183, 1.3032, 1.2938]


                                 1.29
#33:   VECTOR(FLOOR((10·n)     ), n, 20)

#34:   [19, 47, 80, 116, 155, 196, 239, 285, 331, 380, 429, 480, 533, 586,

         641, 697, 753, 811, 870, 929]
```

And indeed $scale^{1.29}$ gives approximately the numbers of the boxes …

Then I changed the scaling from 10, 20, 30, … to 1, 2, 4, 8, 16, …:

```
         fractdim2(set, res, x_range, y_range, n, ix, iy, k, k_, bcounts, z, bc) :=
           Prog
             n := 0
             bcounts := []
             k_ := DIM(set)
             Loop
               If n > res exit
               z := VECTOR(VECTOR(0, i, 2^n), i, 2^n)
               bc := 0
               k := 1
#36:           Loop
                 If k > k_ exit
                 ix := FLOOR(2^n·(set↓k↓1 - x_range↓1)/(x_range↓2 - x_range↓1)) + 1
                 iy := FLOOR(2^n·(set↓k↓2 - y_range↓1)/(y_range↓2 - y_range↓1)) + 1
                 If z↓(2^n + 1 - iy)↓ix = 0
                    z↓(2^n + 1 - iy)↓ix := 1
                 k :+ 1
               bcounts := APPEND(bcounts, [[2^n, Σ(Σ(z))]])
               "IF(n = 4,          RETURN z)"
               n :+ 1
             bcounts`
```

I performed all calculations from above using this new scaling for both ranges:

```
#37:  boxc2 := fractdim2(duffpc, 13, [-5, 5], [-7, 3])
```

#38: $\text{boxc2} := \begin{bmatrix} 1 & 2 & 4 & 8 & 16 & 32 & 64 & 128 & 256 & 512 & 1024 & 2048 & 4096 & 8192 \\ 1 & 4 & 14 & 32 & 98 & 286 & 617 & 856 & 952 & 985 & 988 & 991 & 993 & 996 \end{bmatrix}$

#39: $\text{FIT}([x,\ a\cdot x + b],\ \text{VECTOR}([\text{LOG}(\text{boxc2}_{1,i}),\ \text{LOG}(\text{boxc2}_{2,i})],\ i,\ 6,\ 12)) = 0.24784\cdot x + 5.2512$

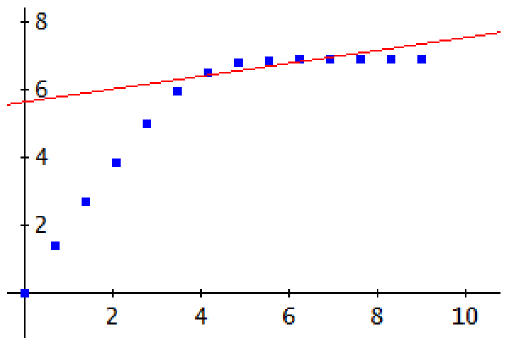
#40: $\text{VECTOR}\left(\dfrac{\text{LOG}(\text{boxc2}_{2,i})}{\text{LOG}(\text{boxc2}_{1,i})},\ i,\ 4,\ 12\right)$

```
#41:  [1.6666, 1.6536, 1.6319, 1.5448, 1.3916, 1.2368, 1.1048, 0.99483, 0.90479]

#42:  boxc2_ := fractdim2(duffpc, 13, [-4, 4], [-6, 2])
```

#43: $\text{boxc2\_} := \begin{bmatrix} 1 & 2 & 4 & 8 & 16 & 32 & 64 & 128 & 256 & 512 & 1024 & 2048 & 4096 & 8192 \\ 1 & 4 & 15 & 48 & 146 & 387 & 677 & 895 & 966 & 987 & 992 & 993 & 993 & 994 \end{bmatrix}$

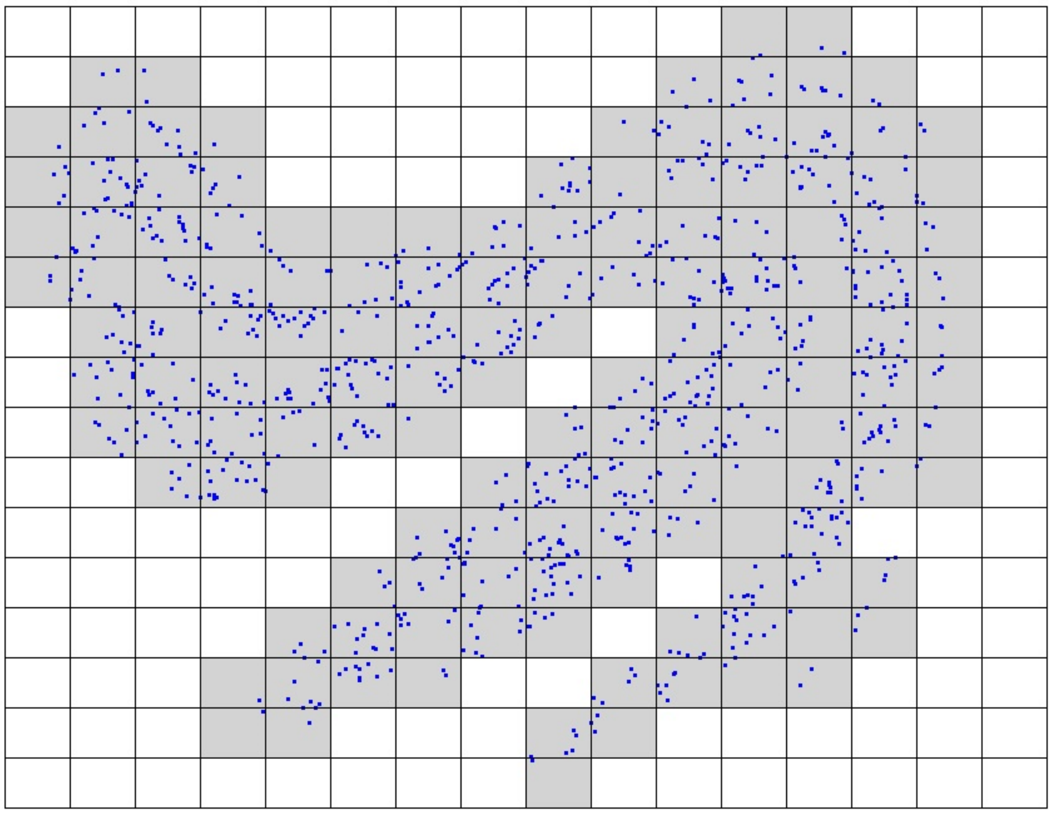
#44: $\text{FIT}([x,\ a\cdot x + b],\ \text{VECTOR}([\text{LOG}(\text{boxc2\_}_{1,i}),\ \text{LOG}(\text{boxc2\_}_{2,i})],\ i,\ 6,\ 12)) = 0.19006\cdot x + 5.6376$

#45: $\text{VECTOR}\left(\dfrac{\text{LOG}(\text{boxc2\_}_{2,i})}{\text{LOG}(\text{boxc2\_}_{1,i})},\ i,\ 3,\ 9\right) = [1.9534,\ 1.8616,\ 1.7974,\ 1.7192,\ 1.5671,\ 1.4008,\ 1.2394]$

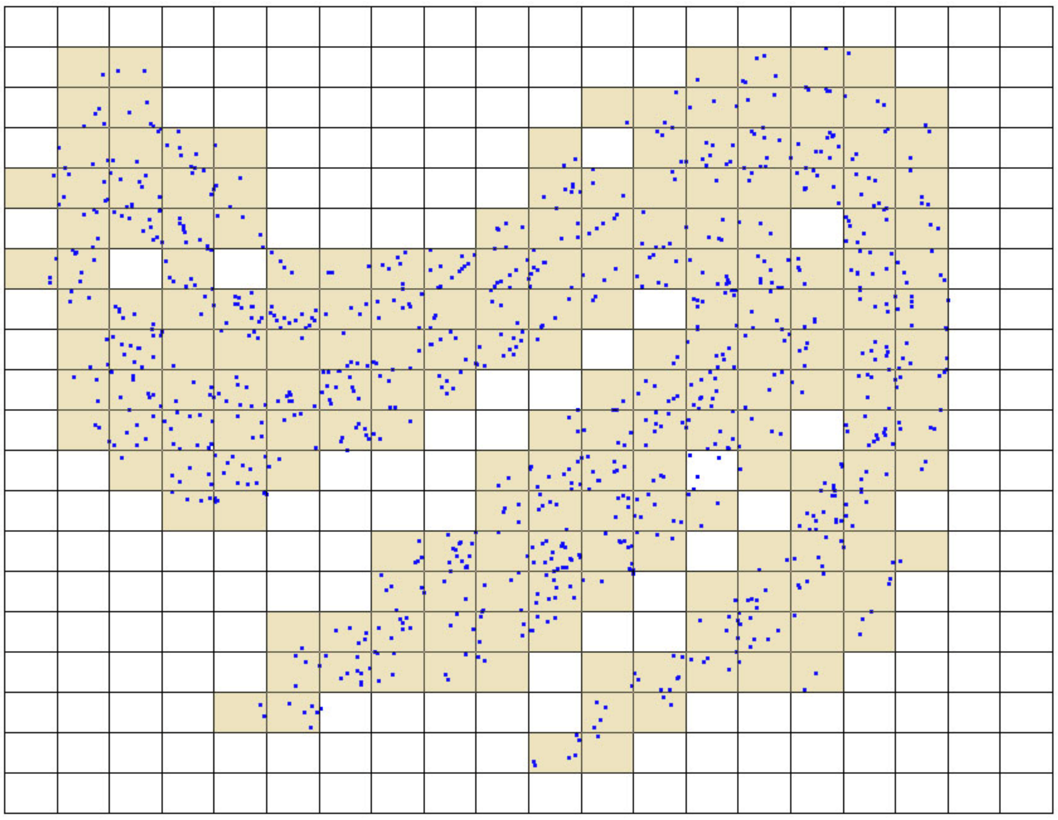


The quotient $\dfrac{\log N(k)}{\log \varepsilon_k}$ tends again very similar to approximately 1.30.

I could not resist to count again the boxes for 16 × 16 and the 20 × 20 grid of boxes for the smaller range in order to test my `fractdim`-programs.



146 boxes?



197 boxes?

Seems to be ok!

25 Another Kind of Fractal Dimension

Sprott uses another kind of fractal dimension. I cite a part of his explanation [1]:

> *One method is to draw a small circle somewhere on the plane that surrounds at least one of the points. We then draw a second circle with the same center but with twice the radius. Now we count the number of points inside each circle. Let's say the smaller circle encloses n_1 points and the larger circle encloses n_2 points. Obviously n_2 is greater than or equal to n_1 because all the points inside the inner circle are also inside the outer circle.*
>
> *If the points are widely separated, then n_2 equals n_1. If the points are part of a straight line, the larger circle on average encloses twice as many points as the smaller circle, but if the points are part of a plane, the larger circle on average encloses four times as many points as the smaller circle, because the area of a circle is proportional to the square of its radius. Thus, for these simple cases the dimension is given by*
>
> $$F = \log_2\left(\frac{n_2}{n_1}\right).$$

Sprott suggests to use a value of ten instead of doubling the radius and chooses the smaller circle about 0.6% the size of the attractor and the larger circle about 6% of this size. As a consequence, we will use logarithms of base 10 instead of base 2. This dimension is called *correlation dimension* which is never greater than the fractal dimension, but it tends not to be much smaller either.

> *Rather than count the number of data points within a circle, which would require that the calculation run to conclusion with the coordinates of all the points saved, we use the equivalent method of determining the probability that two randomly chosen points are within a certain distance of one another. To do this, the distance of each new iterate from one of its randomly chosen predecessors is calculated.*
>
> *Now you see why we bothered to save the last 500 iterates! We exclude the most recent 20 points, because the iterates are likely to be abnormally highly correlated with their recent predecessors. Thus, with each iteration, we have only one additional calculation to do in which we compare the distance of the iterate to one of its randomly chosen predecessors and increment n_1 and n_2, as appropriate.*

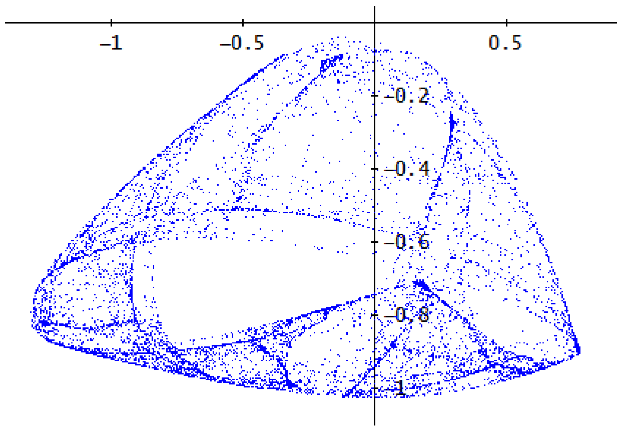
You will find more details in Sprott's book [1].

I "translated" Sprott's Quick BASIC subprogram for calculating the dimension into DERIVE language and then compared my results with Sprott's ones. Sprott discards the first 1000 iterates before calculating the dimension. His BASIC program updates continuously the dimension value using much more points than my DERIVE program which must first store all attractor points.

```
fd(set, dummy, xmax, xmin, ymax, ymin, pt, dmax, d2, n1, n2, i, j) :=
  Prog
    [dummy := RANDOM(0), n1 := 0, n2 := 0, i := 1000]
    [xmax := MAX(set↓↓1), xmin := MIN(set↓↓1)]
    [ymax := MAX(set↓↓2), ymin := MIN(set↓↓2)]
    dmax := (xmax - xmin)^2 + (ymax - ymin)^2
    Loop
      WRITE(i)
      If i > DIM(set) exit
      j := FLOOR(501 + 480·RANDOM(1))
      d2 := ABS(set↓i - set↓j)^2
      If d2 < 0.004·dmax
         n2 :+ 1
      If d2 < 0.00004·dmax
         n1 :+ 1
      i :+ 1
    LOG(n2/n1, 10)
```

I used my map-program from page 24 to create one of Sprott's attractors and calculated the correlation dimension 20 times:

```
qu_map := map(ETJUBWEDNRORR, 10000)

APPROX(VECTOR(fd(qu_map), k, 5), 3)

[1.30, 1.48, 1.45, 1.44, 1.37]

[1.57, 1.64, 1.48, 1.44, 1.31]

[1.39, 1.33, 1.51, 1.43, 1.34]

[1.38, 1.61, 1.40, 1.59, 1.49]
```

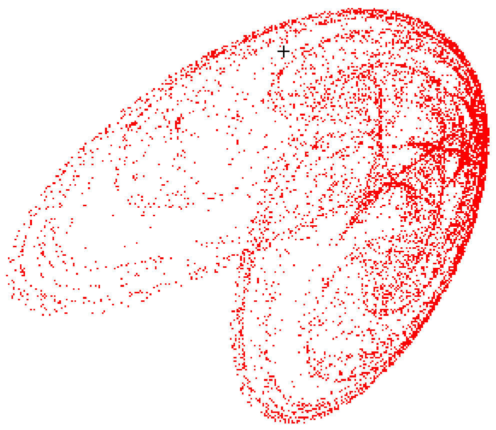


A screen shot of the BASIC program containing much more iterations gives a dimension of 1.42. Thus, my program seems to work correctly (L is the respective Lyapunov-exponent). See page 65.

Let's have a second example:

```
qu_map2 := map(EZPMSGCNFRENG, 10000)

APPROX(VECTOR(fd(qu_map2), k, 5), 3)

[2.01, 1.66, 1.61, 1.76, 1.70]

[1.70, 1.57, 1.68, 1.63, 1.85]
```

Sprott's value: F = 1.67



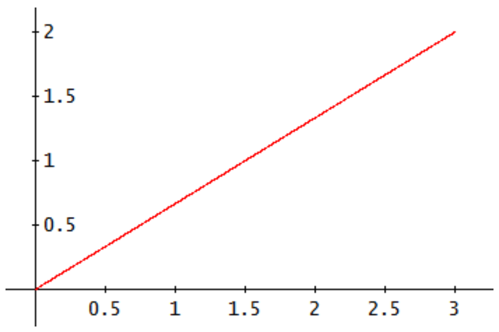
You are invited to calculate the box-counting fractal dimension of these – or other – attractors.

25 Dimension 1, Dimension 2

According to my understanding the set of points filling a line should have dimension 1 and points filling a plane should have dimension 2. So, let's check!

The "attractor" is a segment consisting of 5000 random points. Then I calculate the correlation dimension 10 times:

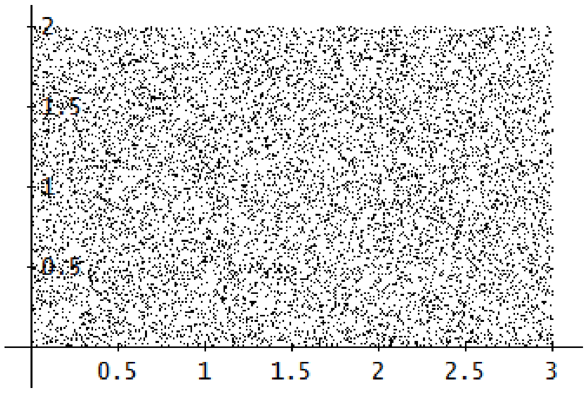
```
line := VECTOR(RANDOM(1)·[3, 2], k, 5000)

APPROX(VECTOR(fd(line), k, 5), 3)

[1.04, 1.10, 0.959, 0.938, 0.961]

[0.936, 0.867, 1.05, 0.921, 0.947]
```



Close enough to 1, isn't it?

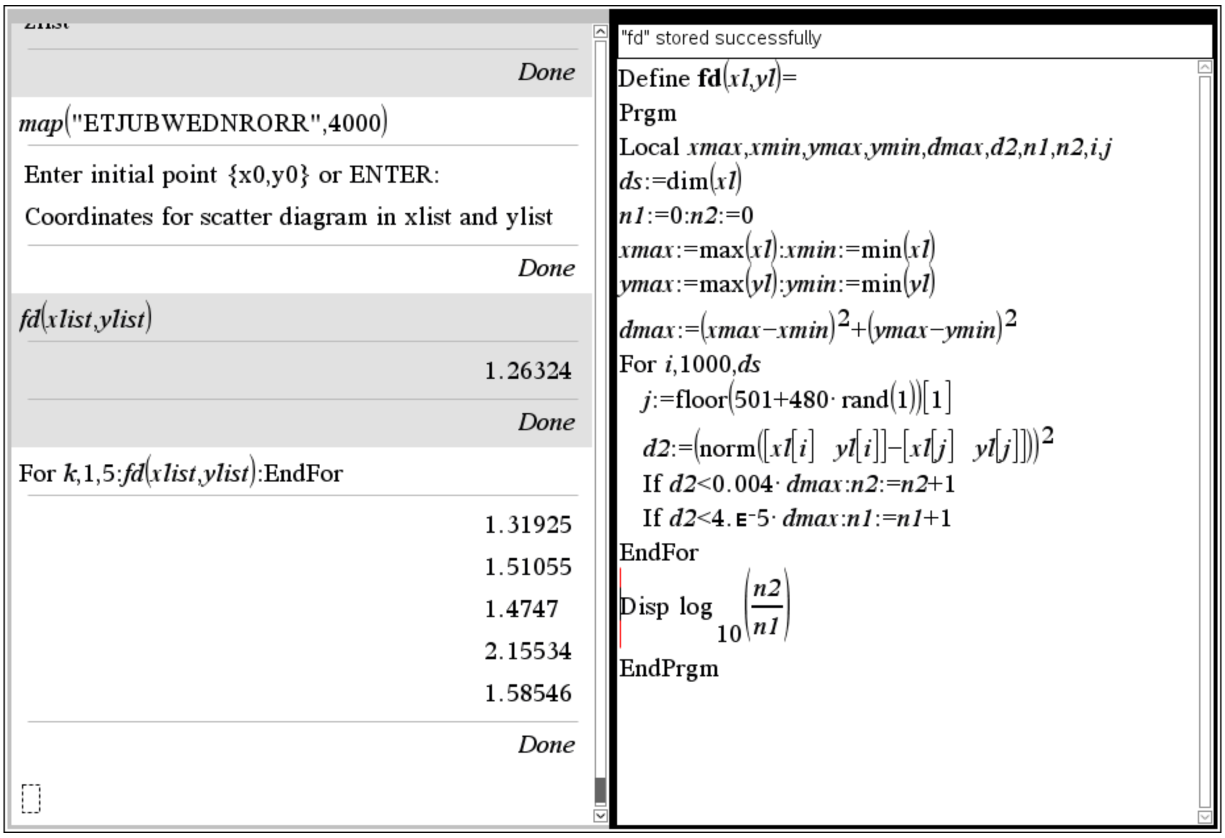
The next attractor is a rectangle "filled" with 10 000 points:

```
rect := VECTOR([3·RANDOM(1), 2·RANDOM(1)], k, 10000)

APPROX(VECTOR(fd(rect), k, 10), 3)

[2.03, 1.83, 1.58, 1.82, 1.98, 1.86, 1.84, 2.35, 2.02, 1.99]

[2, 2.04, 2.02, 1.74, 1.98, 1.67, 2.04, 1.98, 2.05, 1.86]
```



26 Fractal Dimension on TI-NspireCAS

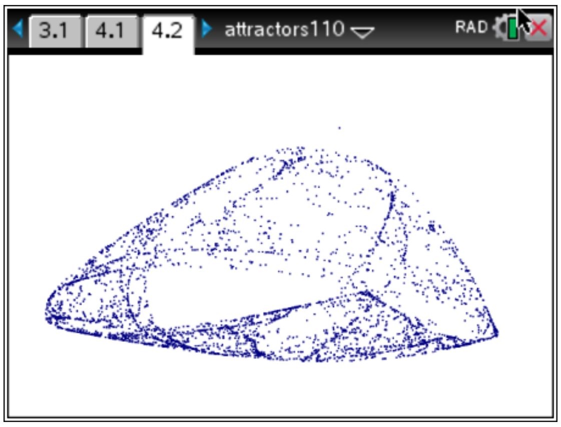
I did not intend to treat the FD with TI-Nspire, too. But finally I thought that it would be fine to add at least one method for Nspire.



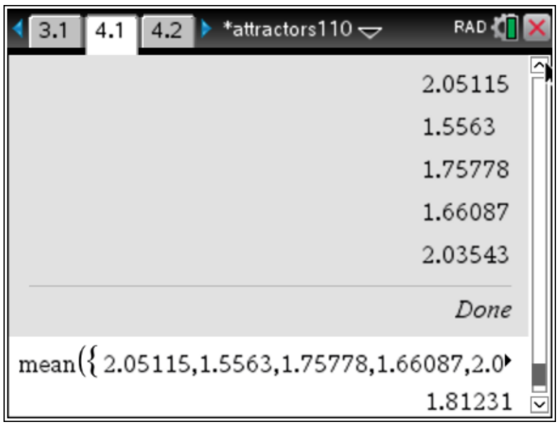
```
zlist
                                    Done
map("ETJUBWEDNRORR",4000)
Enter initial point {x0,y0} or ENTER:
Coordinates for scatter diagram in xlist and ylist
                                    Done
fd(xlist,ylist)
                                 1.26324
                                    Done
For k,1,5:fd(xlist,ylist):EndFor
                                 1.31925
                                 1.51055
                                 1.4747
                                 2.15534
                                 1.58546
                                    Done
```

```
"fd" stored successfully
Define fd(x1,y1)=
Prgm
Local xmax,xmin,ymax,ymin,dmax,d2,n1,n2,i,j
ds:=dim(x1)
n1:=0:n2:=0
xmax:=max(x1):xmin:=min(x1)
ymax:=max(y1):ymin:=min(y1)
dmax:=(xmax−xmin)^2+(ymax−ymin)^2
For i,1000,ds
  j:=floor(501+480· rand(1))[1]
  d2:=(norm([x1[i]  y1[i]]−[x1[j]  y1[j]]))^2
  If d2<0.004· dmax:n2:=n2+1
  If d2<4.E-5· dmax:n1:=n1+1
EndFor
Disp log_10(n2/n1)
EndPrgm
```

On page 80 you can find the respective DERIVE results together with Sprott's FD-values. Comparing them one must have in mind that we cannot consider so many points for calculating the FD with TI-Nspire. 5000 points will cause "Resource exhaustion".

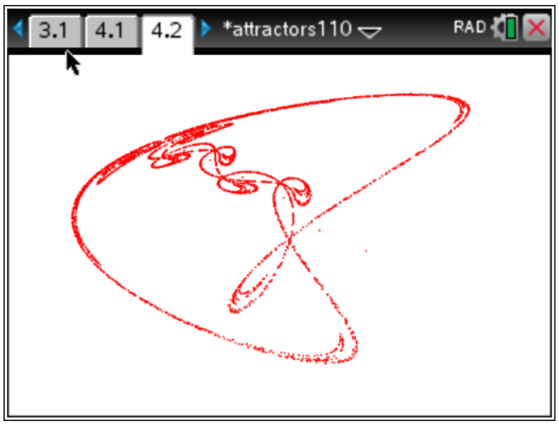


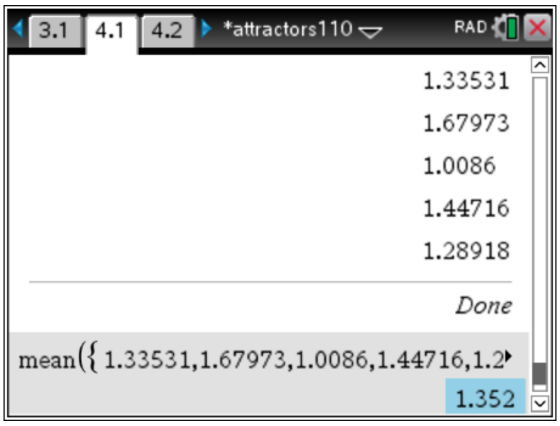
Plot to map("ETJUBWEDNRORR",4000)



```
                                   2.05115
                                    1.5563
                                   1.75778
                                   1.66087
                                   2.03543
                                      Done
mean({2.05115,1.5563,1.75778,1.66087,2.0▸
                                   1.81231
```

FD for map("EZPMSGCNFRENG",5000)





```
                                   1.33531
                                   1.67973
                                    1.0086
                                   1.44716
                                   1.28918
                                      Done
mean({1.33531,1.67973,1.0086,1.44716,1.2▸
                                     1.352
```

Plot and FDs for map("EJTTSMBOGLLQF",4000)

Now I will stop this never-ending story of strange attractors. It kept me busy many, many hours but I am not quite sure, if I am right in all my considerations. So, I could not verify dimension 2 for the rectangle with the box-counting dimension. It would be great to receive any comment and/or improvement.

There are lots of respective websites and books, but I could find only one of them performing the box-counting method step by step (MAPLE) – there were just descriptions – many of them pretty the same!

27 References and Resources

[1] Julien C. Sprott, *Strange Attractors: Creating Patterns in Chaos* http://sprott.physics.wisc.edu/fractals/booktext/

[2], [3] DERIVE Newsletter 10 from 1993 and 13 from 1994, www.austromath.at/dug/

[4] VENSIM PLE, Simulation software for educational purposes, download free of charge http://www.vensim.com/download.html

[5] Frank Piefke, *Simulationen mit dem Personalcomputer*, Hüttig 1991

[6] Herbert Voß, *Chaos und Fraktale selbst programmieren*, Franzis' 1994

[7] K.-H. Becker & M. Dörfler, *Dynamische Systeme und Fraktale*, Vieweg 1989

[8] E. D. Schmitter, *Fraktale Geometrie*, Hofacker 1989

[9] Clifford Pickover, *Mit den Augen des Computers*, Markt und Technik 1992

[10] Clifford Pickover, *Computers and the Imagination*, St. Martin's Press Inc. 1991

[11] Clifford Pickover, *Pattern, Chaos and Beauty*, Dover Publications 1989

[12] Josef Böhm, *Dynamic Systems/Dynamische Systeme*, http://rfdz.ph-noe.ac.at/acdca/materialien.html

[13] Hans Lauwerier, *Fraktale verstehen und selbst programmieren,* Wittig Fachbuch 1989

[14] T. Wegener & M. Peterson, *Fraktale Welten*, te-wi 1992

[15] D. Peak/M. Frame, *Komplexität – Das gezähmte Chaos*, Birkhäuser 1995

Recommended websites

www.math.kit.edu/iana1/~melcher/media/lyapunov-final.pdf

en.wikipedia.org/Lyapunov_exponent

mathworld.wolfram.com/LyapunovCharacteristicExponent.html

math.cmaisonneuve.qc.ca/alevesque/chaos_fract/Attracteurs/Attracteurs.html

blog.nihilogic.dk/2009/10/strange-attractors-beautiful-chaos-and.html

www.robert-doerner.de/Henon-System/henon-system.html

www.complexification.net/gallery/machines/peterdejong/

demonstrations.wolfram.com/PeterDeJongAttractors/

www.cc.gatech.edu/~phlosoft/attractors/

wonderfl.net/tag/Chaos

paulbourke.net/fractals/peterdejong/, paulbourke.net/fractals/ikeda/, paulbourke.net/fractals/fracdim/

paulbourke.net/papers/shier2013/paper.pdf

people.mbi.ohio-state.edu/mgolubitsky/reprintweb-0.5/output/papers/symmetry_increasing.pdf

en.wikipedia.org/wiki/List_of_chaotic_maps

www.xplora.org/downloads/Knoppix/Fraktalwelt/myhome/simpiter.htm#martin

www.fraktalwelt.de/myhome/simpiter2.htm,

www.fraktalwelt.de/myhome/fractype.htm (English and German)

imagej.nih.gov/ij/plugins/fraclac/FLHelp/Fractals.htm

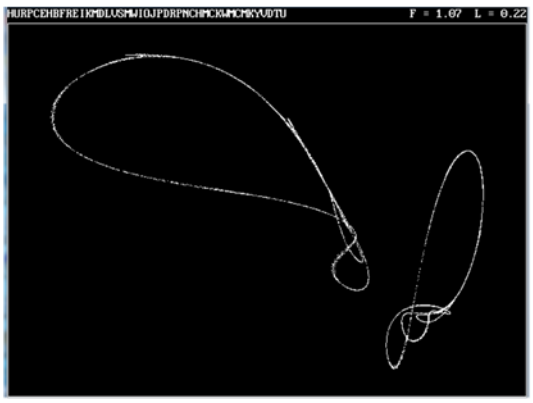
arxiv.org/

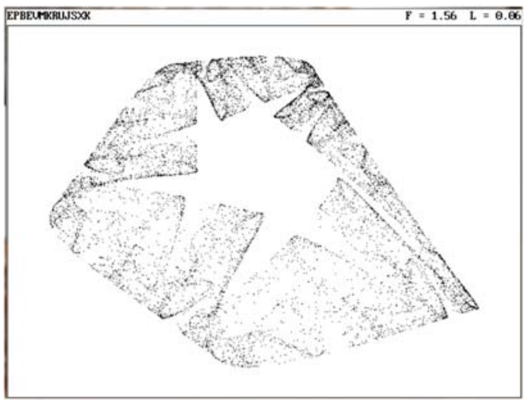
en.wikipedia.org/wiki/Fractal_dimension

www.fast.u-psud.fr/~moisy/ml/boxcount/html/demo.html

The screenshots below are two results produced by Sprott's SA.EXE which is a compiled BASIC program. It does not run under WIN 7/10. But it does work using a "DOS Box"-program.

Download Dos Box: https://sourceforge.net/projects/dosbox/


HURPCEHBFREIKMDLUSMWIOJPDRPNCHMCKWMCMKYVDTU F = 1.07 L = 0.22


EPBEUMKRUJSXK F = 1.56 L = 0.06

I recommend downloading Sprott's book text. It contains much more wonderful inspirations which I cannot present in the frame of this paper.