THE DERIVE - NEWSLETTER #110

ISSN 1990-7079

THE BULLETIN OF THE

DERIVE

USER GROUP

+ CAS-TI

Contents:

July 2018

Dear Josef,

I would like, please, to know if with Derive it is possible to create a table of values of two functions for which the ranges of variability are assigned.

For example, one thing like

Table[(a*b)^b, b^a, a = {2,3,....10}, b = {2,3,...10}]

so that a type table can be obtained

a = 2, b=2, (a*b)^b = 16, b^a = 4
a = 2, b=3, (a*b)^b = 216, b^a = 9
…
a = 3, b = 2, (a*b)^b =36, b^a = 8
a = 3, b = 3, (a*b)^b = 729, b^a =27

ecc. ecc.

If my request is not applicable, I apologize.
Thank you very much,
Giuseppe Ornaghi

Hi Guiseppe,

Try this:

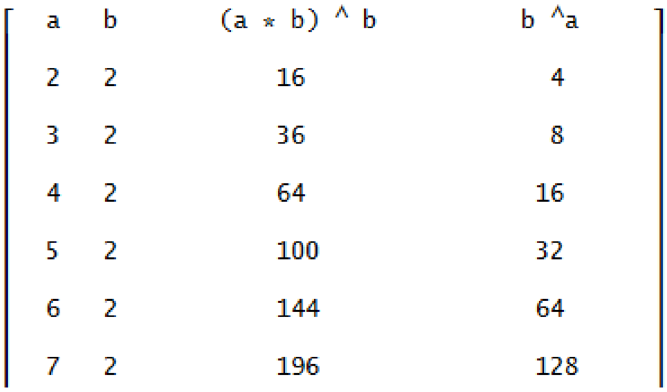
VECTOR(VECTOR([a, b, (a·b)^b, b^a], a, [2, ..., 10]), b, [2, ..., 10]).

Or even better:

APPEND(VECTOR(VECTOR([a, b, (a·b)^b, b^a], a, [2, ..., 10]), b, [2, ..., 10])).

Maybe that you would like to add a headline:

APPEND([["a", "b", "(a * b) ^ b", "b ^a"]], APPEND(VECTOR(VECTOR([a, b, (a·b)^b, b^a], a, [2, ..., 10]), b, [2, ..., 10])))



```
a   b      (a * b) ^ b       b ^a
2   2          16              4
3   2          36              8
4   2          64             16
5   2         100             32
6   2         144             64
7   2         196            128
```

You must know that the VECTOR-function is one of my favorite DERIVE constructs.

Best regards

Josef

Dear DUG Members,

I am writing my letter with a very bad conscience because DNL#110 is so long overdue.

The main problem was that parts of this DNL needed a lot of work.

I extended my original Attractors paper not only by adding some other attractors but also by treating them with the TI-Nspire. It is great and amazing as well that we can obtain satisfying results even on the handheld screen.

Much more work caused the approach to the *Fractal Dimension*. I was inspired to this by an article on the Duffing oscillator and its Poincaré sections. The authors added an investigation on the fractal dimension of the attractor similar plot. It is a Maxima-file and this was a challenge to translate it into DERIVE- and TI-Nspire tongue.

However, I am not quite sure if my considerations are right. I did not find so many resources how to calculate the FD. There are lots of verbal explanations – all of them more or less pretty similar – but very few examples. So, I must help myself. It would be great to receive any comments and/or improvements.

Nevertheless, the FD-activity was very interesting and exciting.

I was very glad to find a paper from our friend Carl Leinbach. He provides one more example of a meaningful application of integration. As teacher I was always frustrated finding so many integration examples in the textbooks which only calculated areas between function graphs – without giving the area any sense. Carl is living in a seniors' residence and I am quite sure that he would enjoy any correspondence. His email-address is given at the end of his paper.

A mail from Argentina gave reason for an extended answer and a contribution for this DNL. Have you ever heard about Laguerre's method for polynomial root finding? And if so, did you know that this numerical method is implemented in DERIVE and the TI-NspireCAS, as well? Many thanks to Albert Rich and David Stoutemyer for their immediate responses.

Last but not least I'd like to thank three DUG members who helped David Dyer to get the requested DERIVE manuals. They sent the books free of charge to David. It's wonderful how our community is working.

Don Phillips sent some Nspire-Goodies for you, which will be presented in the next DNL.

Finally, my wife and I would like to wish you a nice summer, fine holidays and we will meet again in fall.

Best regards and wishes

Josef

The *DERIVE-NEWSLETTER* is the Bulletin of the *DERIVE* & CAS-*TI User Group*. It is published at least four times a year with a content of 40 pages minimum. The goals of the *DNL* are to enable the exchange of experiences made with *DERIVE*, *TI*-CAS and other CAS as well to create a group to discuss the possibilities of new methodical and didactical manners in teaching mathematics.

Editor: Mag. Josef Böhm
D´Lust 1, A-3042 Würmla, Austria
Phone: ++43-(0)660 31 36 365
e-mail: nojo.boehm@pgv.at

Contributions:
Please send all contributions to the Editor. Non-English speakers are encouraged to write their contributions in English to reinforce the international touch of the *DNL*. It must be said, though, that non-English articles will be warmly welcomed nonetheless. Your contributions will be edited but not assessed. By submitting articles, the author gives his consent for reprinting it in the *DNL*. The more contributions you will send, the more lively and richer in contents the *DERIVE* & CAS-*TI Newsletter* will be.

Next issue: September 2018

Preview: Contributions waiting to be published

Some simulations of Random Experiments, J. Böhm, AUT, Lorenz Kopp, GER
Wonderful World of Pedal Curves, J. Böhm, AUT
Tools for 3D-Problems, P. Lüke-Rosendahl, GER
Simulating a Graphing Calculator in *DERIVE*, J. Böhm, AUT
Graphics World, Currency Change, P. Charland, CAN
Cubics, Quartics – Interesting features, T. Koller & J. Böhm, AUT
Logos of Companies as an Inspiration for Math Teaching
Exciting Surfaces in the FAZ / Pierre Charland´s Graphics Gallery
BooleanPlots.mth, P. Schofield, UK
Old traditional examples for a CAS – What´s new? J. Böhm, AUT
Mandelbrot and Newton with *DERIVE*, Roman Hašek, CZK
Tutorials for the NSpireCAS, G. Herweyers, BEL
Some Projects with Students, R. Schröder, GER
Dirac Algebra, Clifford Algebra, D. R. Lunsford, USA
A New Approach to Taylor Series, D. Oertel, GER
Rational Hooks, J. Lechner, AUT
Statistics of Shuffling Cards, Charge in a Magnetic Field, H. Ludwig, GER
Factoring Trinomials, D. McDougall, CAN
Selected Lectures from TIME 2016
More Applications of TI-Innovator^{TM} Hub and TI-Innovator^{TM} Rover
Surfaces and their Duals, Cayley Symmetroid
Affine Mappings –Treated Systematically, H. Nieder, GER
Three planes in space – how they can intersect
Goodies for TI-NspireCAS, Don Phillips, USA
Why ships can swim, W. Alvermann, GER
and others

Impressum:
Medieninhaber: *DERIVE* User Group, A-3042 Würmla, D´Lust 1, AUSTRIA
Richtung: Fachzeitschrift
Herausgeber: Mag. Josef Böhm

Attracted by (STRANGE) Attractors (4)

An illustrated guided tour from well-known to unknown attractors

Your Tour Guide: Josef Böhm

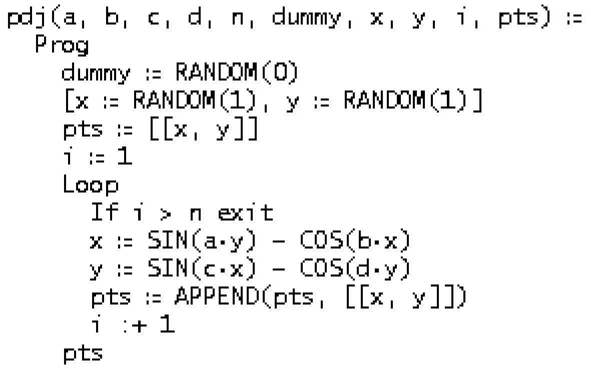
Peter De Jong-Attractors – and "My" Attractor

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

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

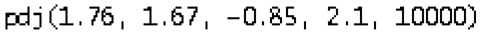
with x_0 and y_0 any real number not being both = 0.

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
```

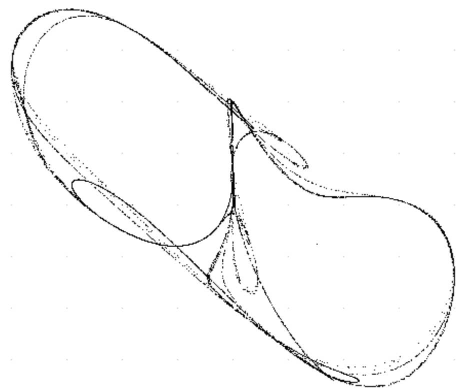
I was happy with my first strange attractor resulting from a random choice for *a*, *b*, *c*, *d*.



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

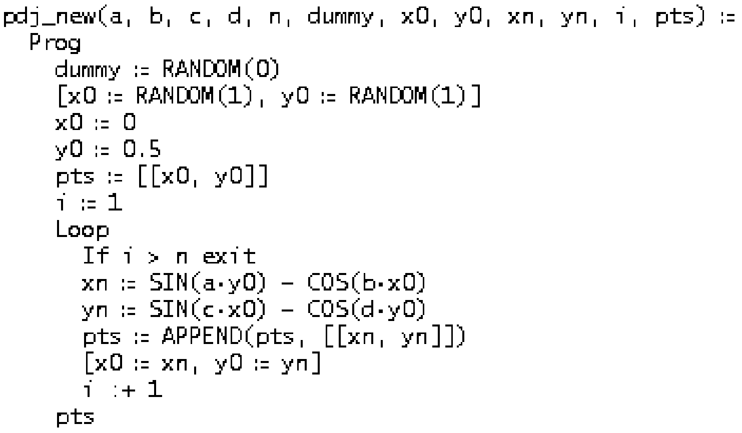
Inspecting the DERIVE-code carefully you may discover a big mistake in the algorithm. The two important lines describing the recursion are generating another dynamic system, namely

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



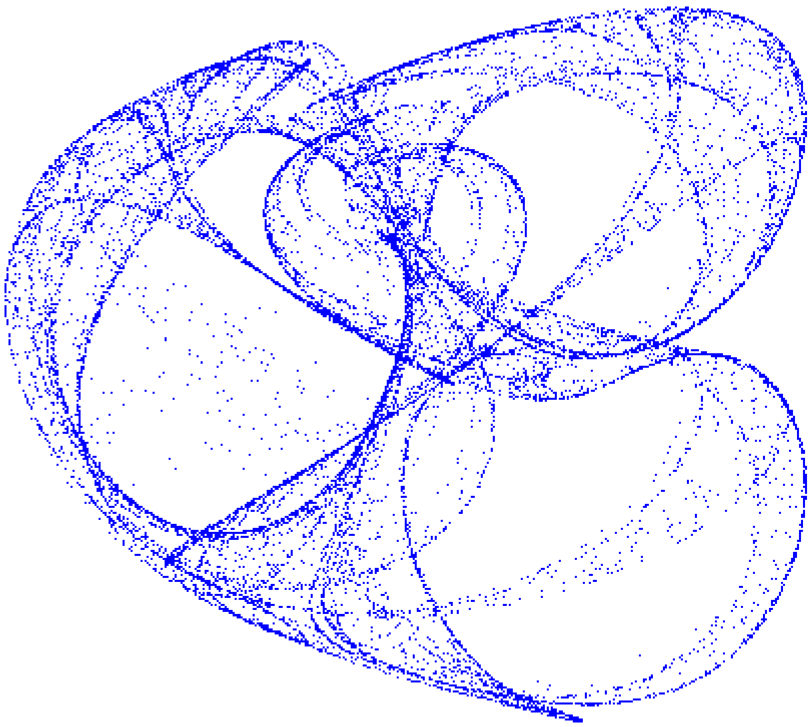
My first Peter De Jong-attractor which wasn't one

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
```

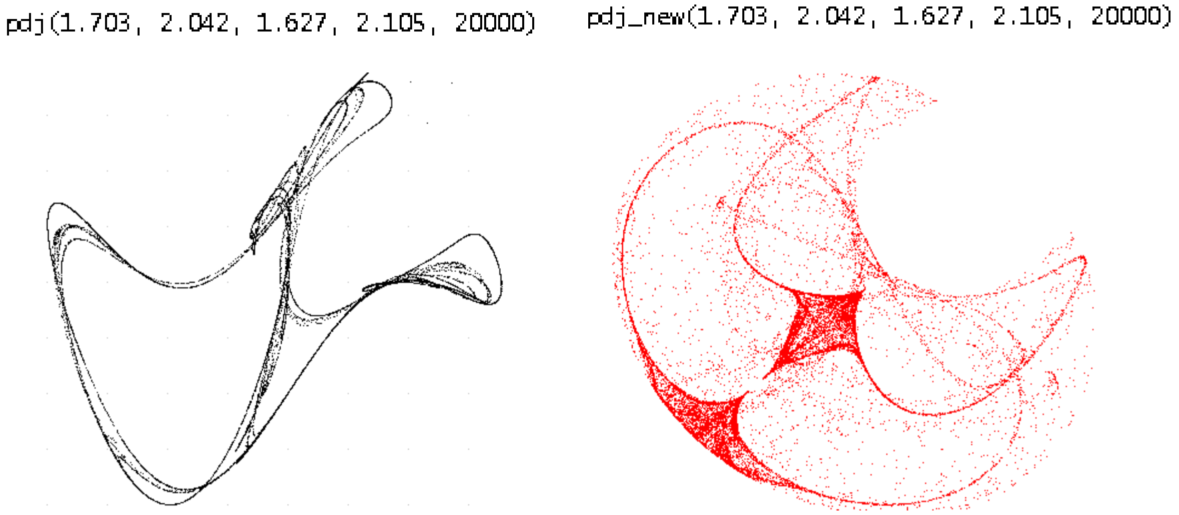
pdj_new(1.76, 1.67, -0.85, 2.1, 20000)



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.

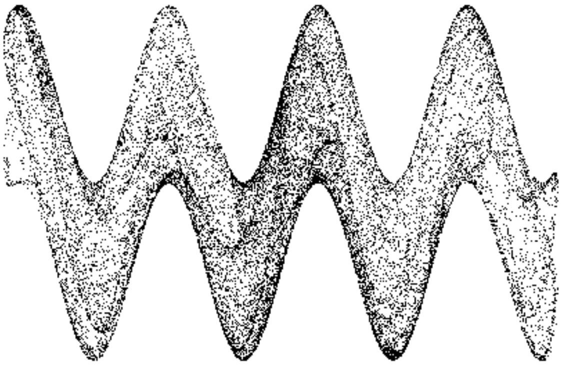


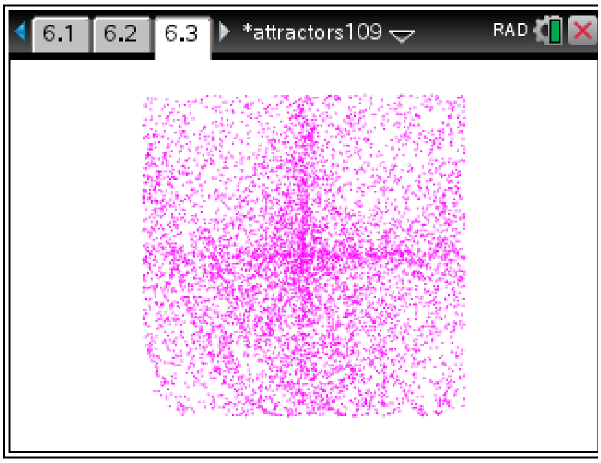
pdj(1.703, 2.042, 1.627, 2.105, 20000)

pdj_new(1.703, 2.042, 1.627, 2.105, 20000)

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

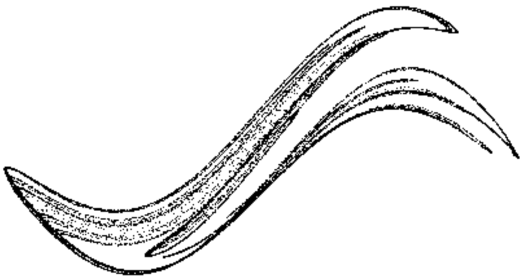
pdj(1.4,-2.3,2.4,2.1,7000)

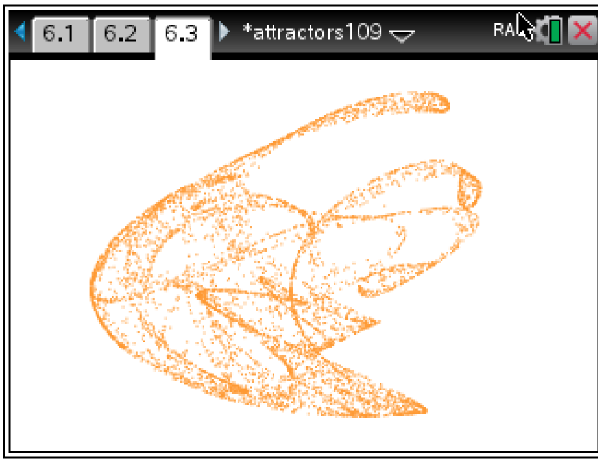




pdj(1.916, -2.325, 1.565, 0.914, 20000)

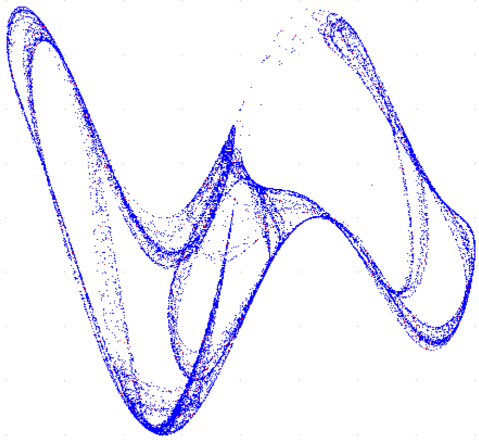
pdj(2.814,4.998,5.817,6.903,7000)

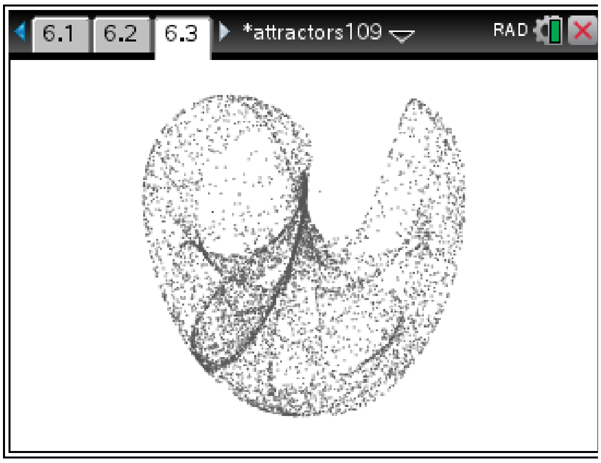




pdj(1.4, -2.3, 2.4, 2.1, 20000)

pdj(1.916,-2.325,1.565,0.914,7000)





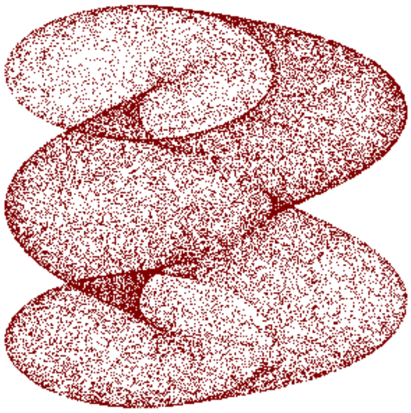
Some other comparisons ("true" PdJ-attractor is in the right column)

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.

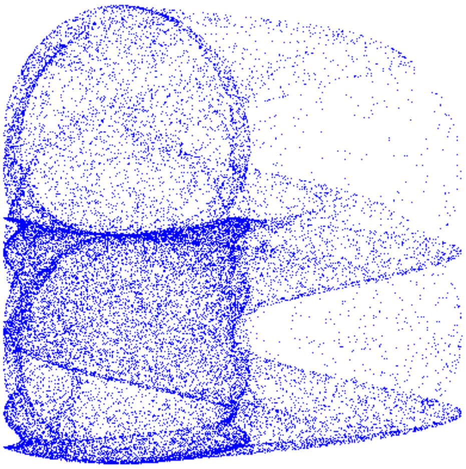
There is a chapter “One Million Points Sculptures” in Clifford Pickover’s very recommendable book *Computers and the Imagination* [10] (in German: *Mit den Augen des Computers* [9]) where Pickover presents graphs consisting of one-million-point-attractors generated by dynamic systems based also on trigonometric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

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

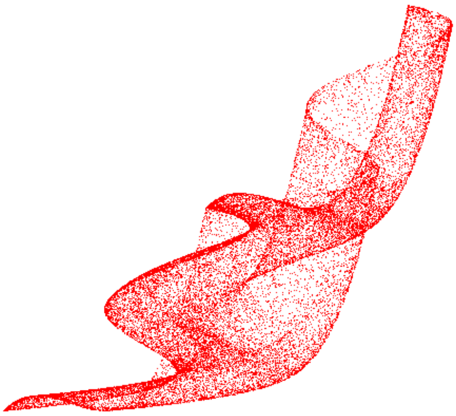
```
cliff0(a, b, n, x0, y0, xn, yn, i, pts) :=
  Prog
    [x0 := 10, y0 := 10]
    pts := [[x0, y0]]
    i := 1
    Loop
      If i > n exit
      xn := SIN(b·y0) + SIN(b·x0)^2
      yn := SIN(a·x0) + SIN(a·y0)^2
      pts := APPEND(pts, [[xn, yn]])
      [x0 := xn, y0 := yn]
      i :+ 1
    pts

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)
```



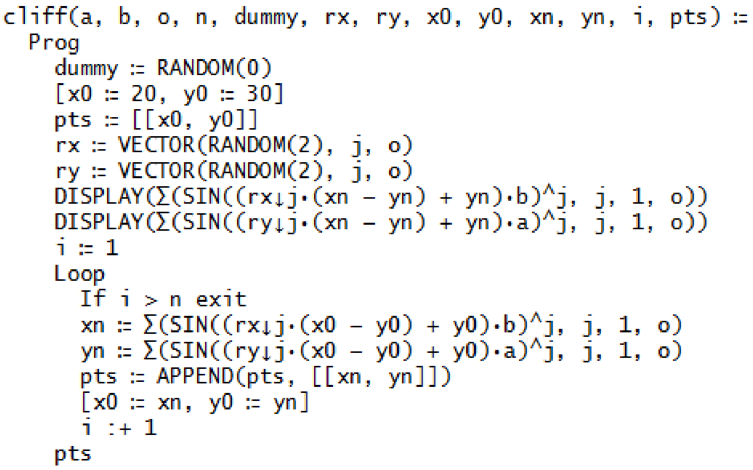
“Clifford –attractors”

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

Preparing this DNL 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:

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

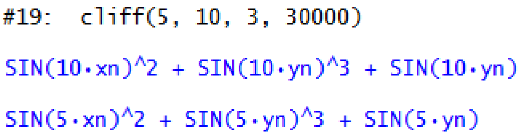
So, we have a sum of powers of sine-functions with $b{\cdot}x_n$ or $b{\cdot}y_n$ for x_{n+1}·and with $a{\cdot}x_n$ or $a{\cdot}y_n$ for y_{n+1}. According to Pickover we take only the first expressions of the sums. All initial points (x_0, y_0) lead to the same figure. I take $x_0 = 20$, $y_0 = 30$.



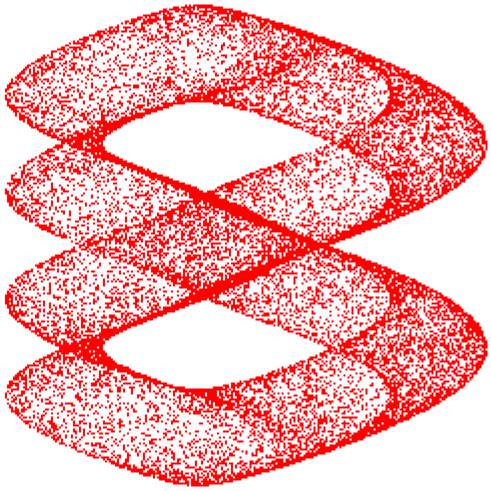
```
cliff(a, b, o, n, dummy, rx, ry, x0, y0, xn, yn, i, pts) :=
  Prog
    dummy := RANDOM(0)
    [x0 := 20, y0 := 30]
    pts := [[x0, y0]]
    rx := VECTOR(RANDOM(2), j, o)
    ry := VECTOR(RANDOM(2), j, o)
    DISPLAY(∑(SIN((rx↓j·(xn - yn) + yn)·b)^j, j, 1, o))
    DISPLAY(∑(SIN((ry↓j·(xn - yn) +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

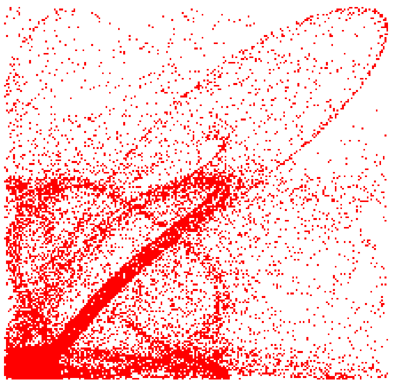
Now let's have a run:



```
#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?”

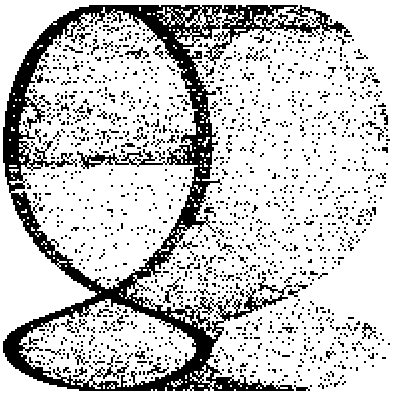
```
cliff f(a, b, o, n, x0 := 20, y0 := 30, dummy, rx, ry, ra, rb, x0, y0, xn, yn, i, pts) :=
  Prog
    dummy := RANDOM(0)
    pts := [[x0, y0]]
    rx := VECTOR(RANDOM(2), j, o)
    ry := VECTOR(RANDOM(2), j, o)
    ra := VECTOR(RANDOM(2), j, o)
    rb := VECTOR(RANDOM(2), j, o)
    DISPLAY(∑(SIN((rx↓j·(xn - 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.

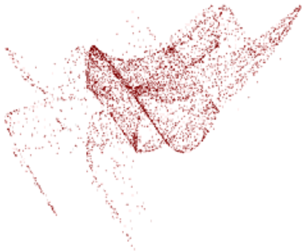
```
#22:  clifff(5, 10, 3, 30000)

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

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

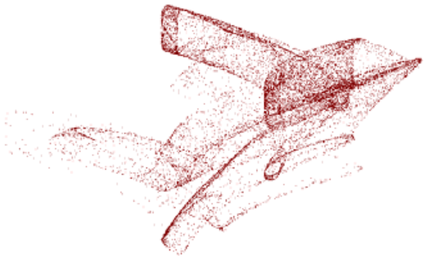
The random number generator gave only *b*s for x_{n+1} and only *a*s for y_{n+1}.

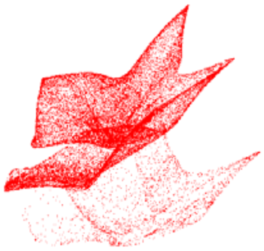
The attractor is pretty “attractive”, isn’t it?



```
#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
```



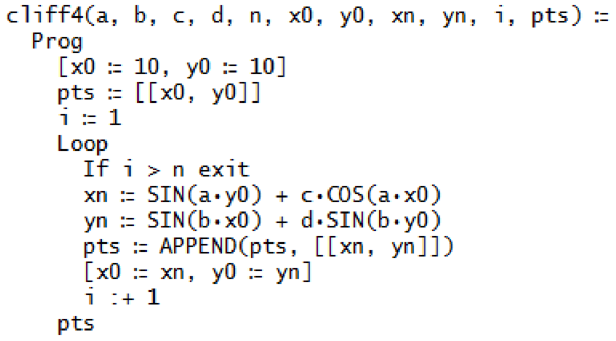




I tried to find some more details about this family of attractors in the web and so I came across the site

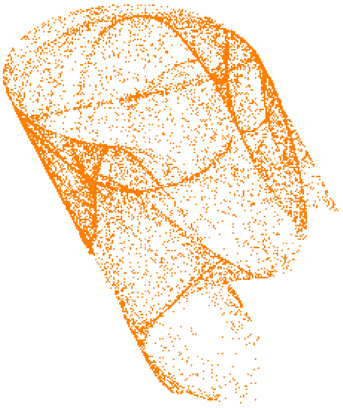
https://fronkonstin.com/2017/11/07/drawing-10-million-points-with-ggplot-clifford-attractors/

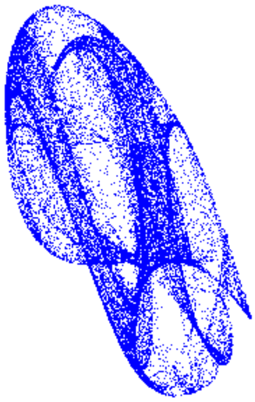
where a certain "*Fronkonstin*" used a different pair of equations generating a Clifford-attractor:



```
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)
```



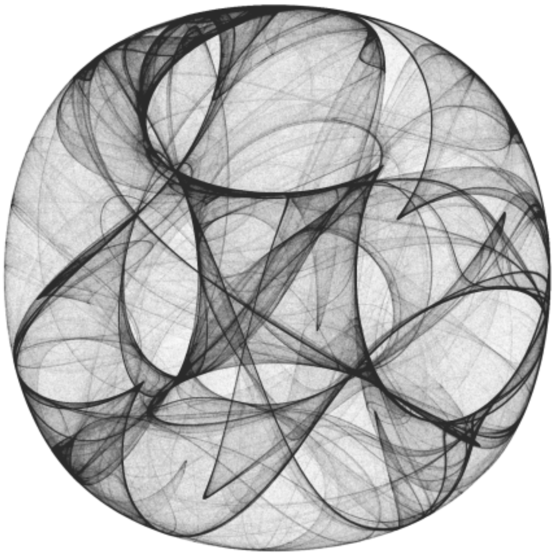


```
cliff4(-1.244580466, -1.81590817, -1.251918341, -1.908667352, 30000)
```

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:

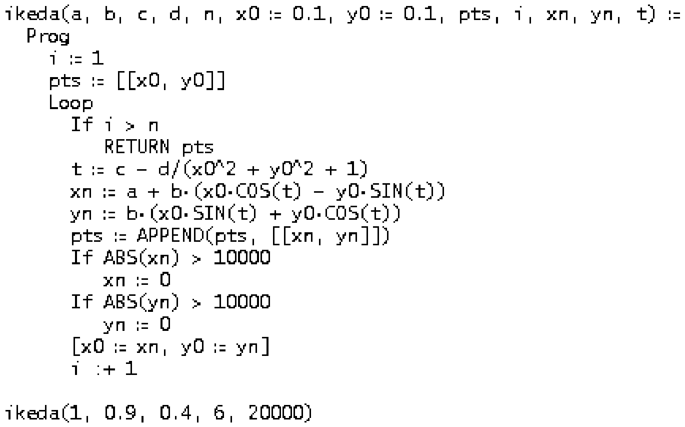
https://fronkonstin.com/2014/10/13/beautiful-curves-the-harmonograph/
http://www.walkingrandomly.com/?p=151

Finally, my last one – the Ikeda Attractor

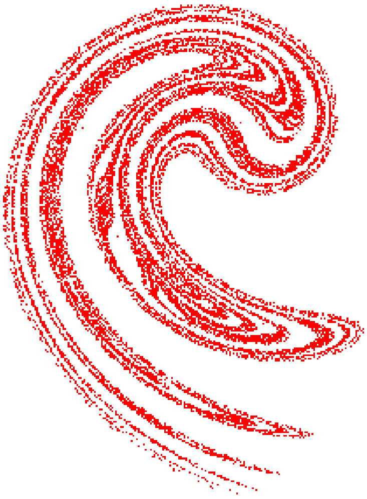
The Ikeda system [9, 10] is given by:

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

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



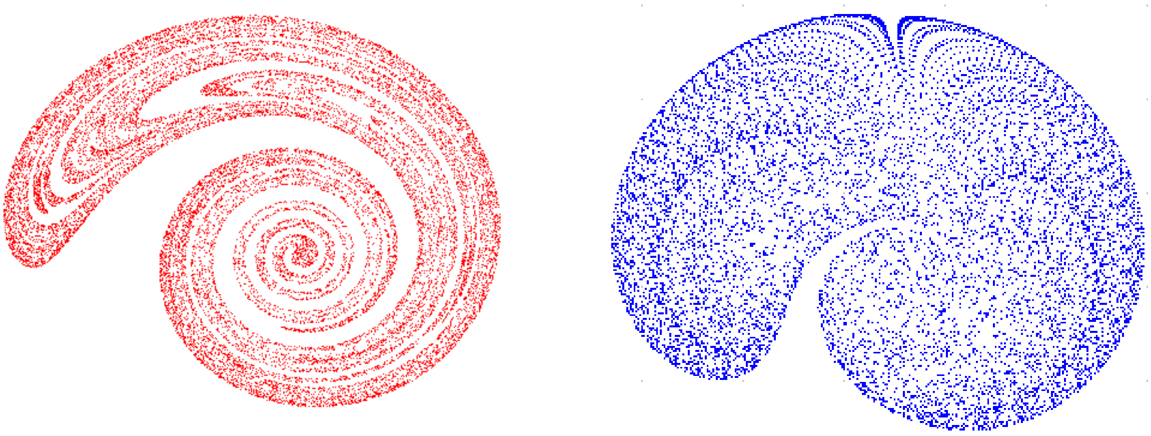
```
ikeda(a, b, c, d, n, x0 := 0.1, y0 := 0.1, pts, i, xn, yn, t) :=
  Prog
    i := 1
    pts := [[x0, y0]]
    Loop
      If i > n
         RETURN pts
      t := c - d/(x0^2 + y0^2 + 1)
      xn := a + b·(x0·COS(t) - y0·SIN(t))
      yn := b·(x0·SIN(t) + y0·COS(t))
      pts := APPEND(pts, [[xn, yn]])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

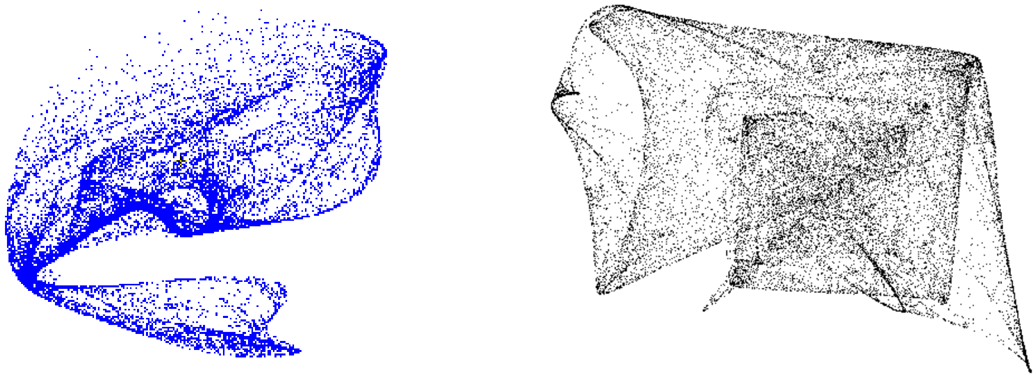
```
xn := a + b·(x0·COS(t) + y0·SIN(t)),
yn := b·(x0·SIN(t) - y0·COS(t)),
```



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

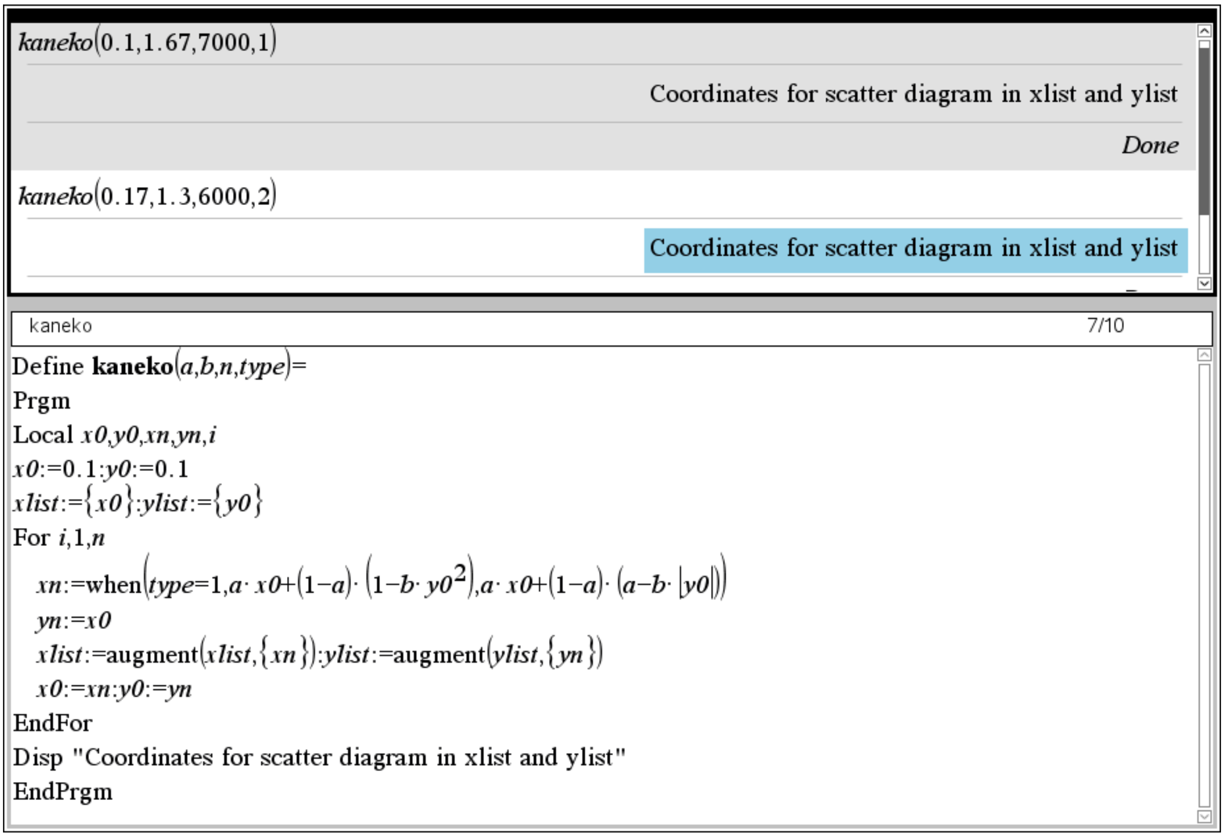
But not last attractor in this Newsletter

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).

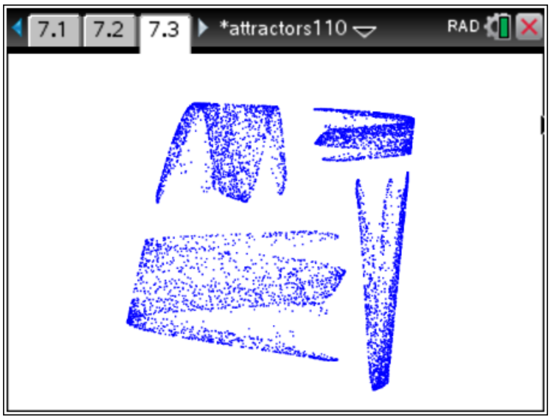


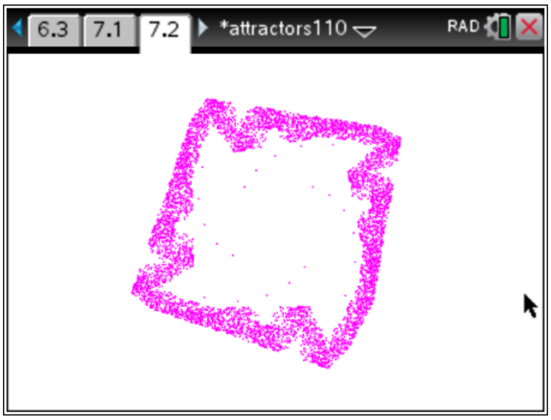
```
kaneko(0.1,1.67,7000,1)
                     Coordinates for scatter diagram in xlist and ylist
                                                                   Done
kaneko(0.17,1.3,6000,2)
                     Coordinates for scatter diagram in xlist and ylist
```

kaneko 7/10

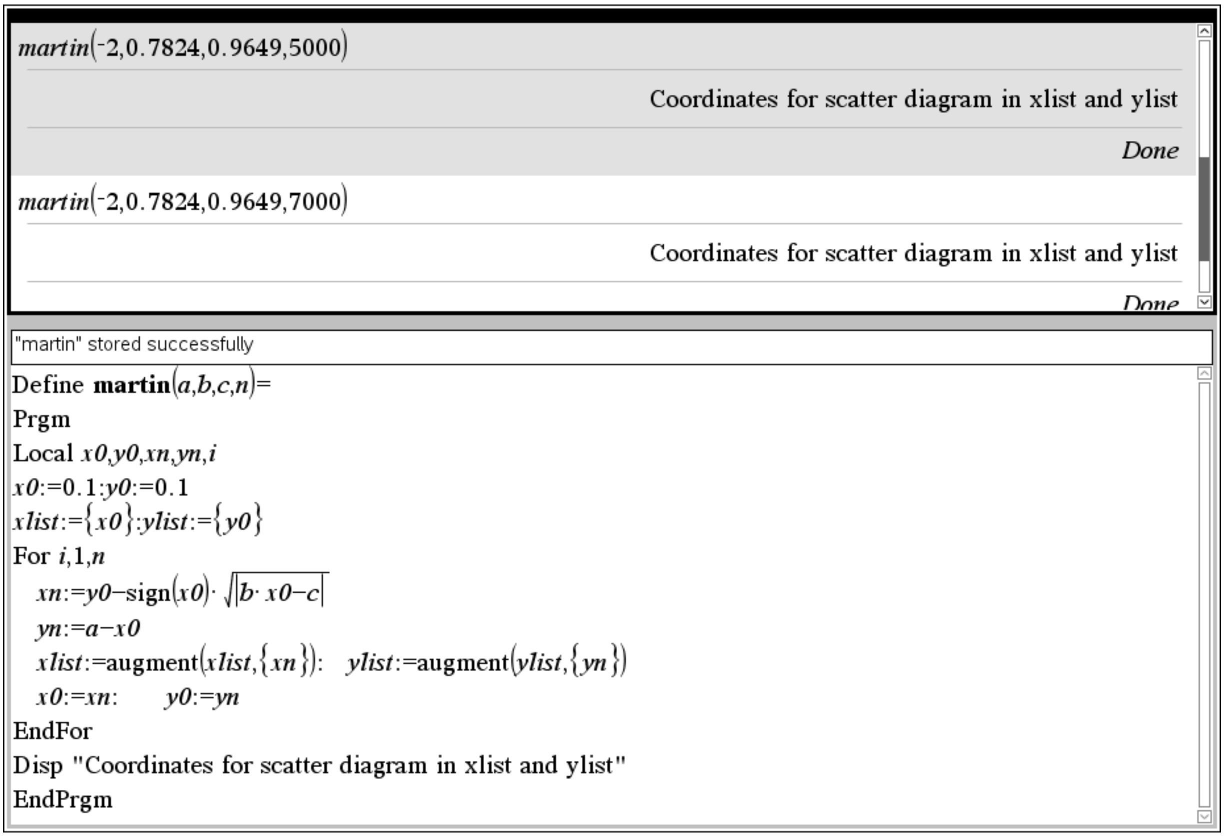
```
Define kaneko(a,b,n,type)=
Prgm
Local x0,y0,xn,yn,i
x0:=0.1:y0:=0.1
xlist:={x0}:ylist:={y0}
For i,1,n
  xn:=when(type=1,a·x0+(1−a)·(1−b·y0^2),a·x0+(1−a)·(a−b·|y0|))
  yn:=x0
  xlist:=augment(xlist,{xn}):ylist:=augment(ylist,{yn})
  x0:=xn:y0:=yn
EndFor
Disp "Coordinates for scatter diagram in xlist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

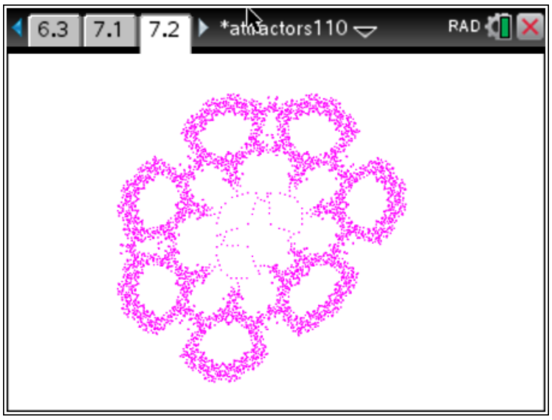


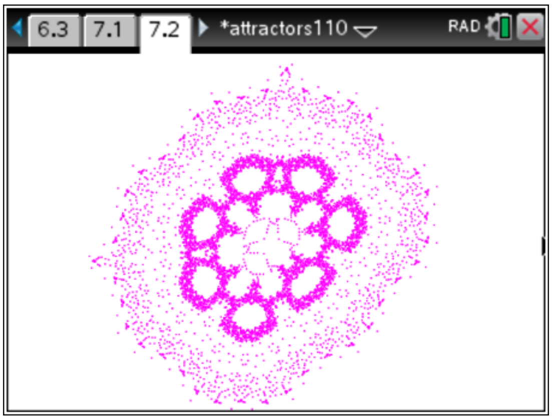
```
martin(⁻2,0.7824,0.9649,5000)
                    Coordinates for scatter diagram in xlist and ylist
                                                              Done
martin(⁻2,0.7824,0.9649,7000)
                    Coordinates for scatter diagram in xlist and ylist
                           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



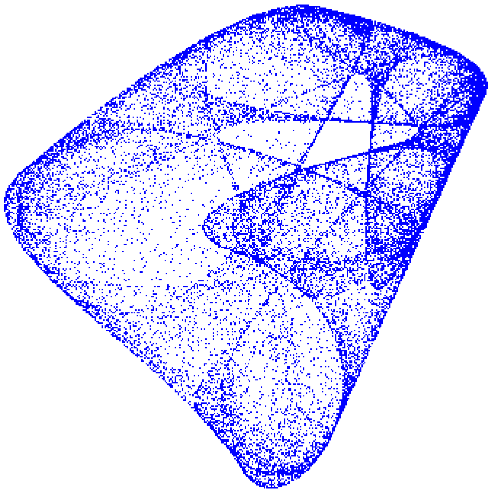


Both plots above show the same attractor with different number of iterations.

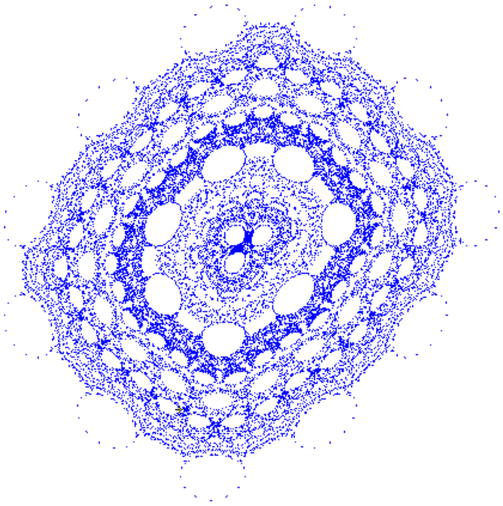
It is a good idea to test the programs with the parameters provided by *Schwebinghaus* but the real adventure will start when you try to explore the world of attractors by trying own parameters being aware that you might be the first person admiring your product. Try also to vary the recursion equations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

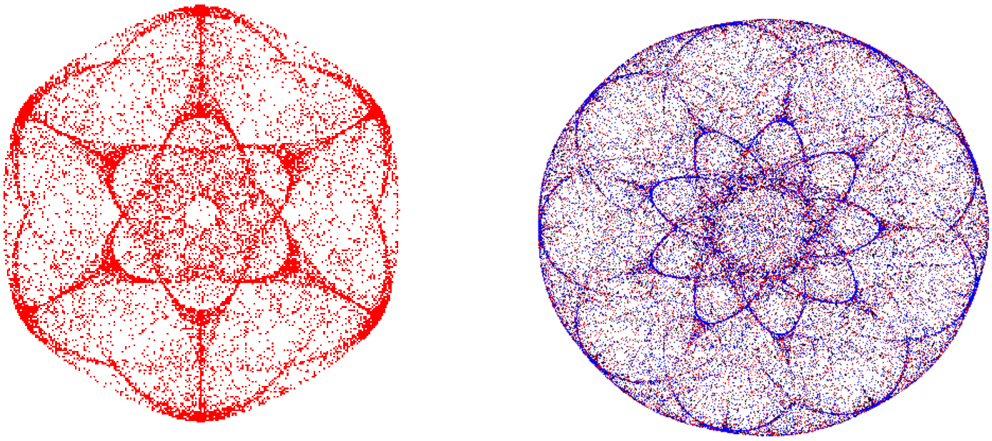


Summary

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 is one example of my further search in the web:



For further investigations: symmetric *Golubitsky*-attractors

[1] DERIVE Newsletter 13 from 1994, http://www.austromath.at/dug/

[2] DERIVE Newsletter 10 from 1993, http://www.austromath.at/dug/

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

[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

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/

paulbourke.net/fractals/peterdejong/

demonstrations.wolfram.com/PeterDeJongAttractors/

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

wonderfl.net/tag/Chaos

paulbourke.net/fractals/ikeda/

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

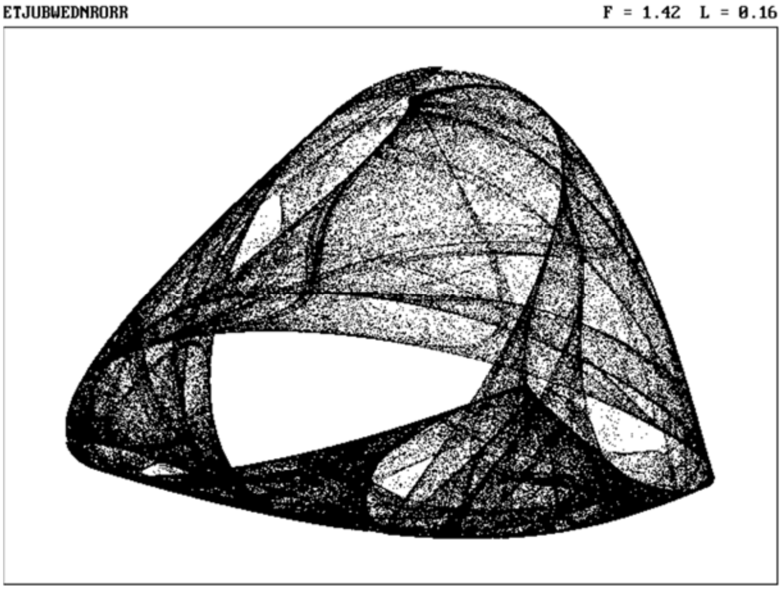
www.fraktalwelt.de/myhome/simpiter2.htm

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

Duffing Oscillator, Poincaré Sections and Fractal Dimension

Josef Böhm, Würmla. Austria

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:

1 The Duffing Oscillator

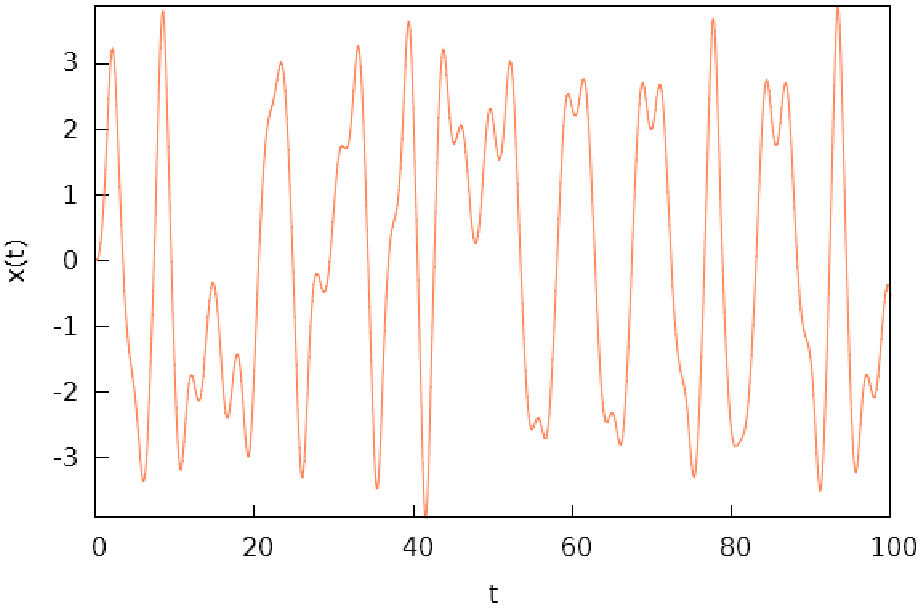
We start solving the differential equation numerically

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

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

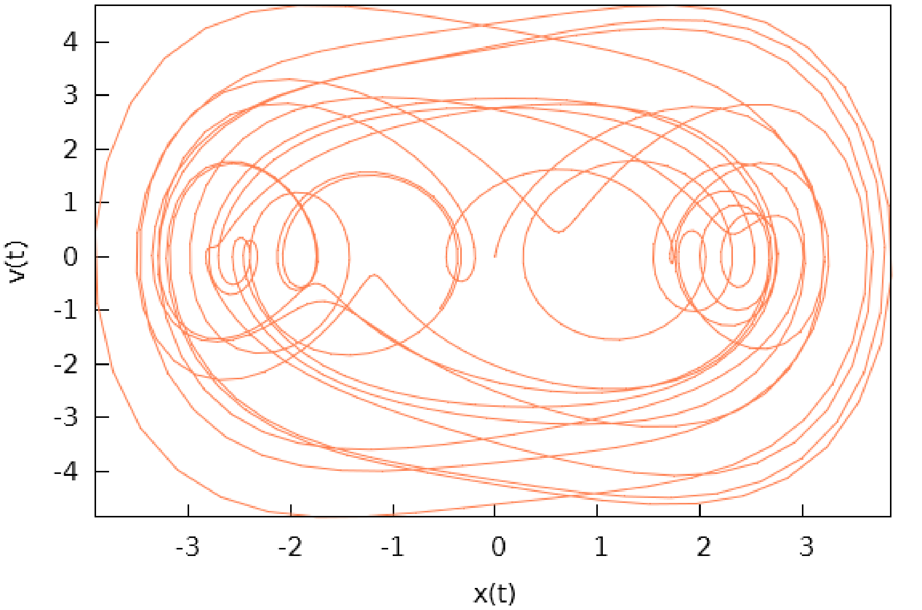
```
(%i15)  duff1:[v,-v/10+x-x^3/4+2.5*sin(2*t)]$
(%i16)  icduff1:[0,0]$;
(%i17)  sduff1:rk(duff1,[x,v],icduff1,[t,0,100,0.1])$;
(%i18)  cduff1:map(lambda([x],rest(x,-1)),sduff1)$;
(%i19)  pduff1:map(lambda([x],rest(x)),sduff1)$;
(%i20)  wxdraw2d(point_type=none,points_joined=true,color = coral,
   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

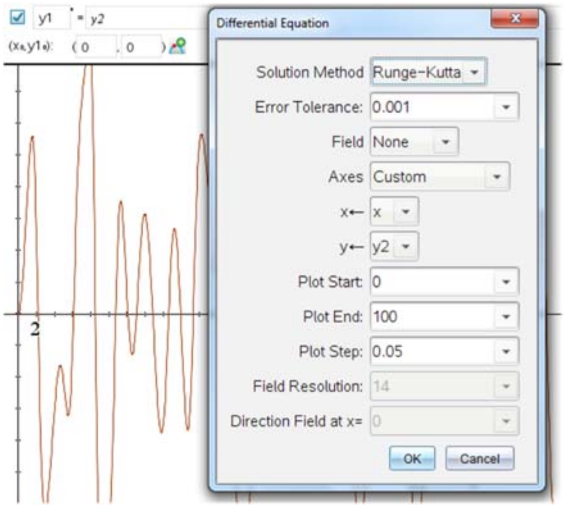


For editing the 2nd order DE it is necessary to rewrite it as a system of two differential equations.

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.



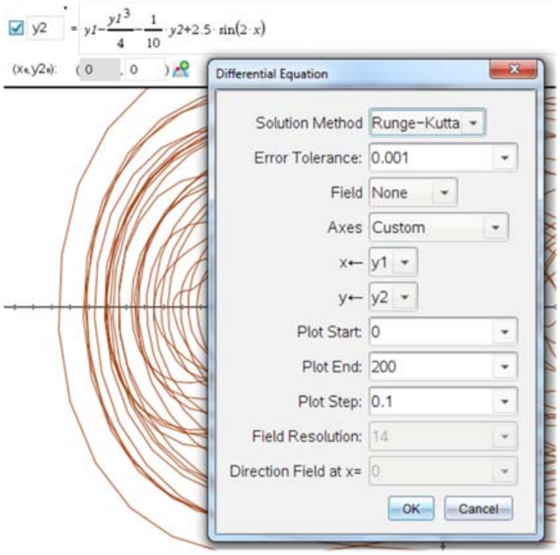
☑ y1' = y2

(x0,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

OK Cancel



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

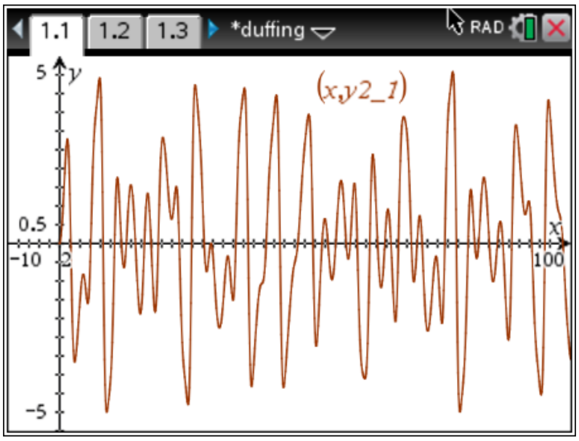
(x0,y2_0): (0 , 0)

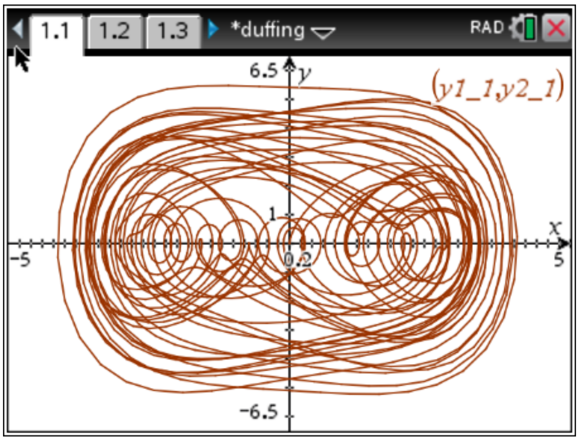
Differential Equation

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

OK Cancel

In order to save space I print the handheld screens:

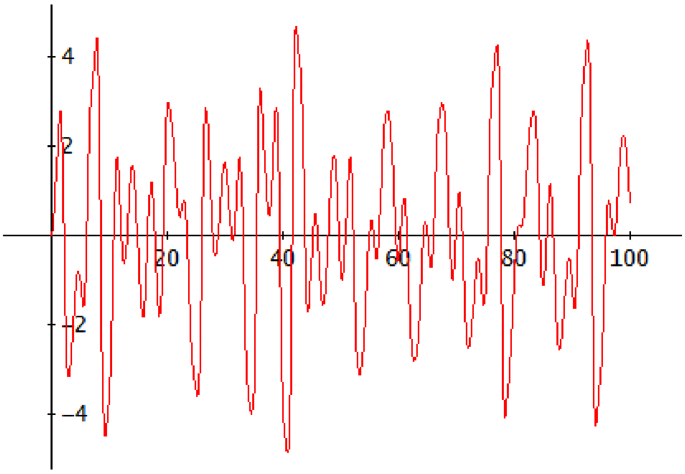




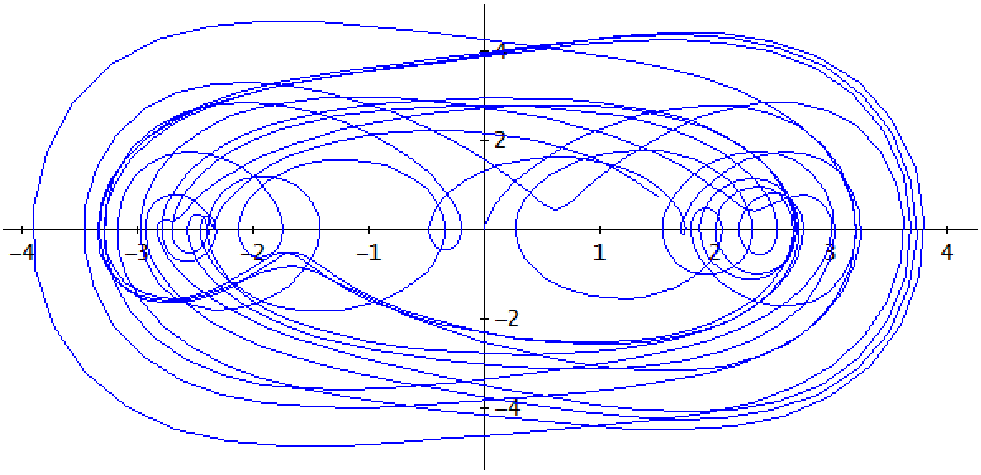
Following once more the instructions it is no problem to present the oscillator on the DERIVE-screen:

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

#8: duffing1 COL [1, 3]



#9: duffing1 COL [2, 3]



2 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.

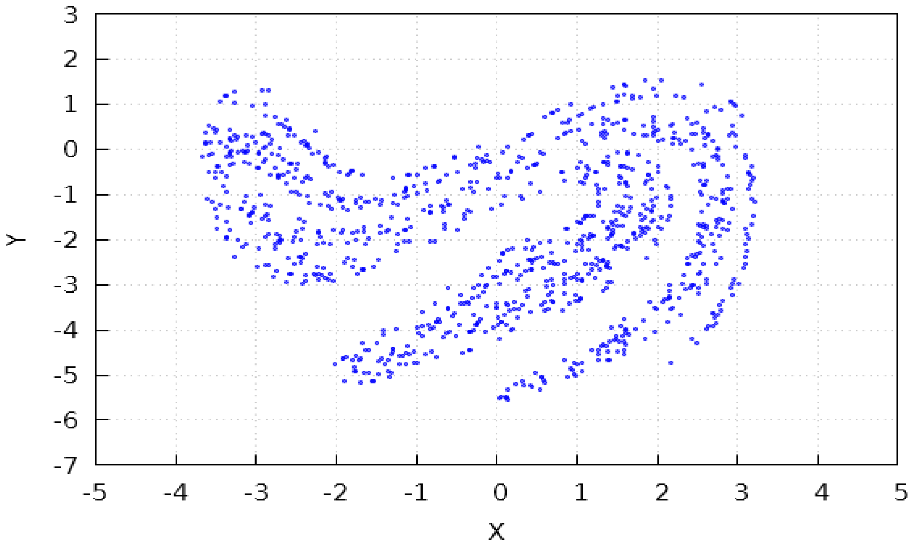
He introduced stops equally spaced in time and fixed the respective points. The frequency of our oscillator is 2, hence the period is $T = \dfrac{2\pi}{2} = \pi$. What we do now is the following: we take every moment of the phase diagram when a full period is done and plot the respective point.

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 with 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)

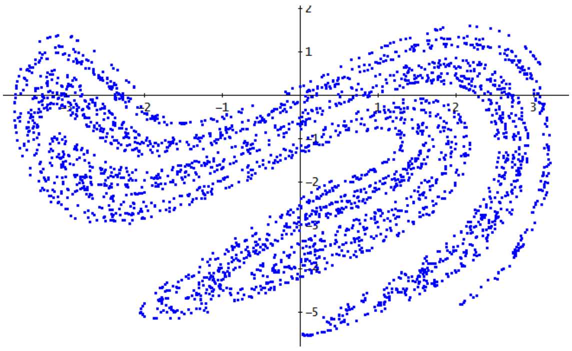


The DERIVE code is very short. I perform 90 000 RK-iteration steps which gives 3000 points of the Poincaré section.

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

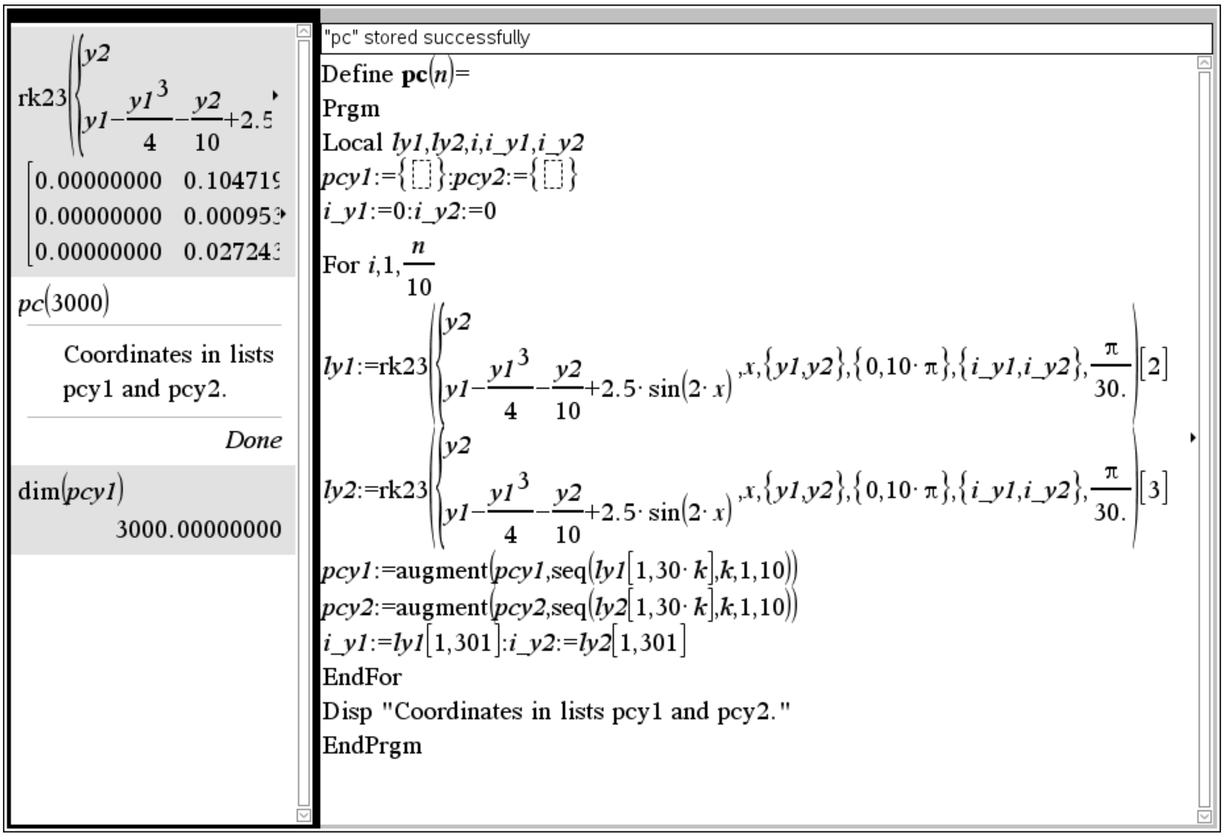
$$\#2:\quad \text{duff} := \left[v,\ -\frac{x^3}{4} + x - \frac{v}{10} + 2.5\cdot \text{SIN}(2\cdot t) \right]$$

```
      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:



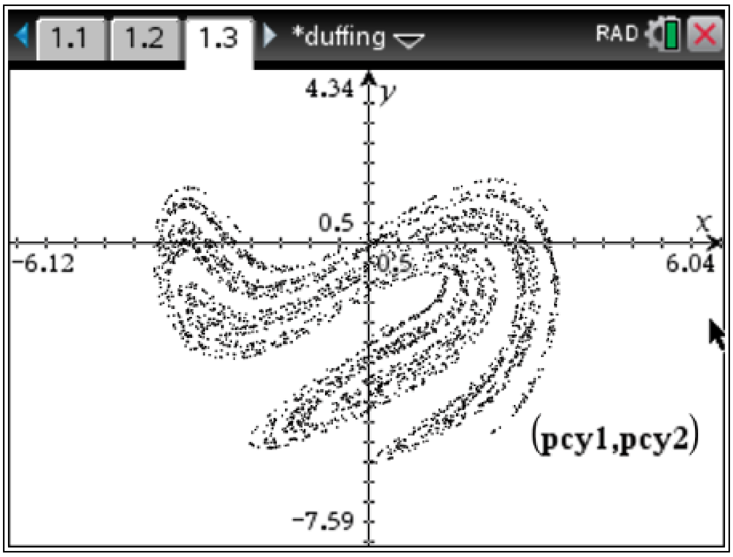
```
rk23({y2, y1-y1^3/4-y2/10+2.5 ▸
[0.00000000  0.10471
 0.00000000  0.00095▸
 0.00000000  0.02724]
pc(3000)
        Coordinates in lists
        pcy1 and pcy2.
                        Done
dim(pcy1)
                3000.00000000
```

```
"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*) (→ 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]
```

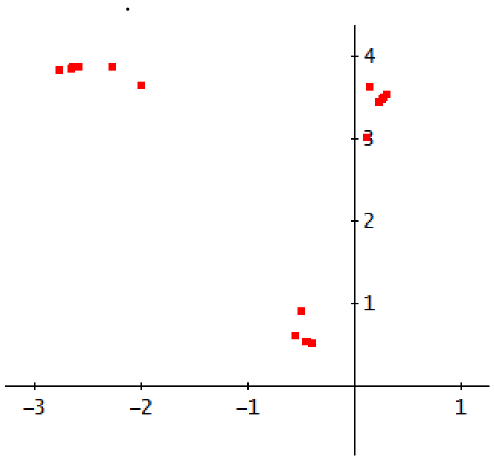
```
poincare2(500)
```

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.

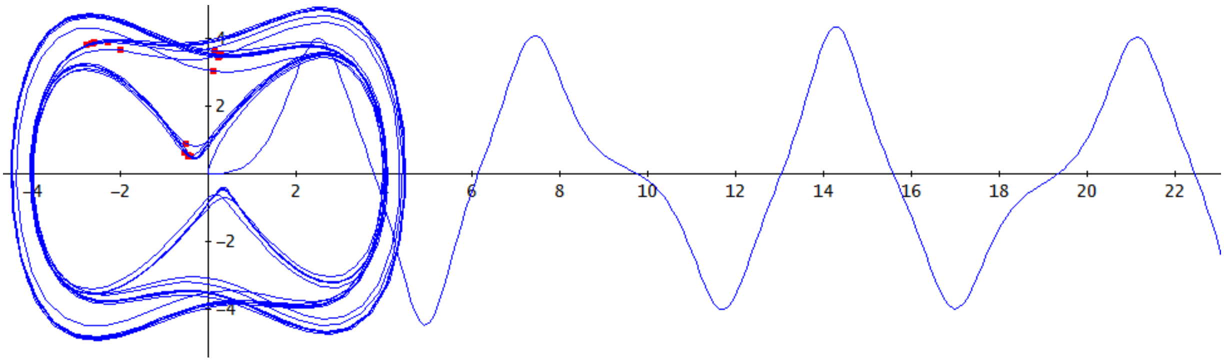
We see three cycles corresponding with the three clusters of P.S.-points.

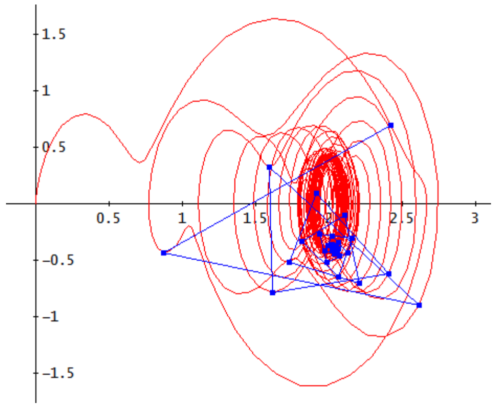
Conclusion: We cannot expect a strange attractor automatically.



$$\left(\text{RK}\left(\text{duff2},\ [t,\ x,\ v],\ [0,\ 0,\ 0],\ \frac{\pi}{30},\ 500\cdot60\right)\right)\ \text{COL}\ [1,\ 2]$$

$$\left(\text{RK}\left(\text{duff2},\ [t,\ x,\ v],\ [0,\ 0,\ 0],\ \frac{\pi}{30},\ 500\cdot60\right)\right)\ \text{COL}\ [2,\ 3]$$





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)
```

$$\left(\text{RK}\left(\text{duff3},\ [t,\ x,\ v],\ [0,\ 0,\ 0],\ \frac{\pi}{30},\ 500\cdot30\right)\right)\ \text{COL}\ [2,\ 3]$$

3 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.

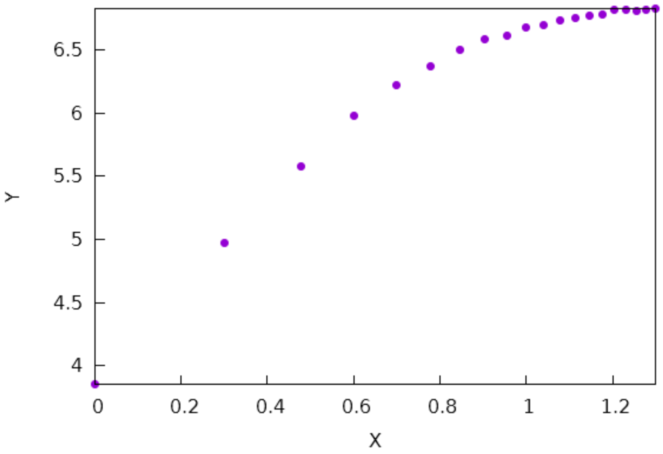
So, we have a sequence of *k* scales ε_k (10,20,30, …, 200) and a corresponding sequence *N*(*k*) of point containing grid boxes. The box-counting dimension *D* is then defined as

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

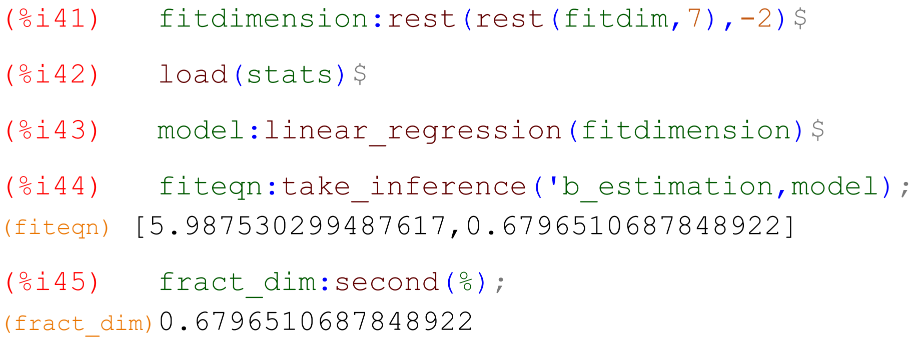
Morante and Valeja plot log *N*(*k*) versus ε_k and estimate *D* as the slope of the respective linear regression line.

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)
```



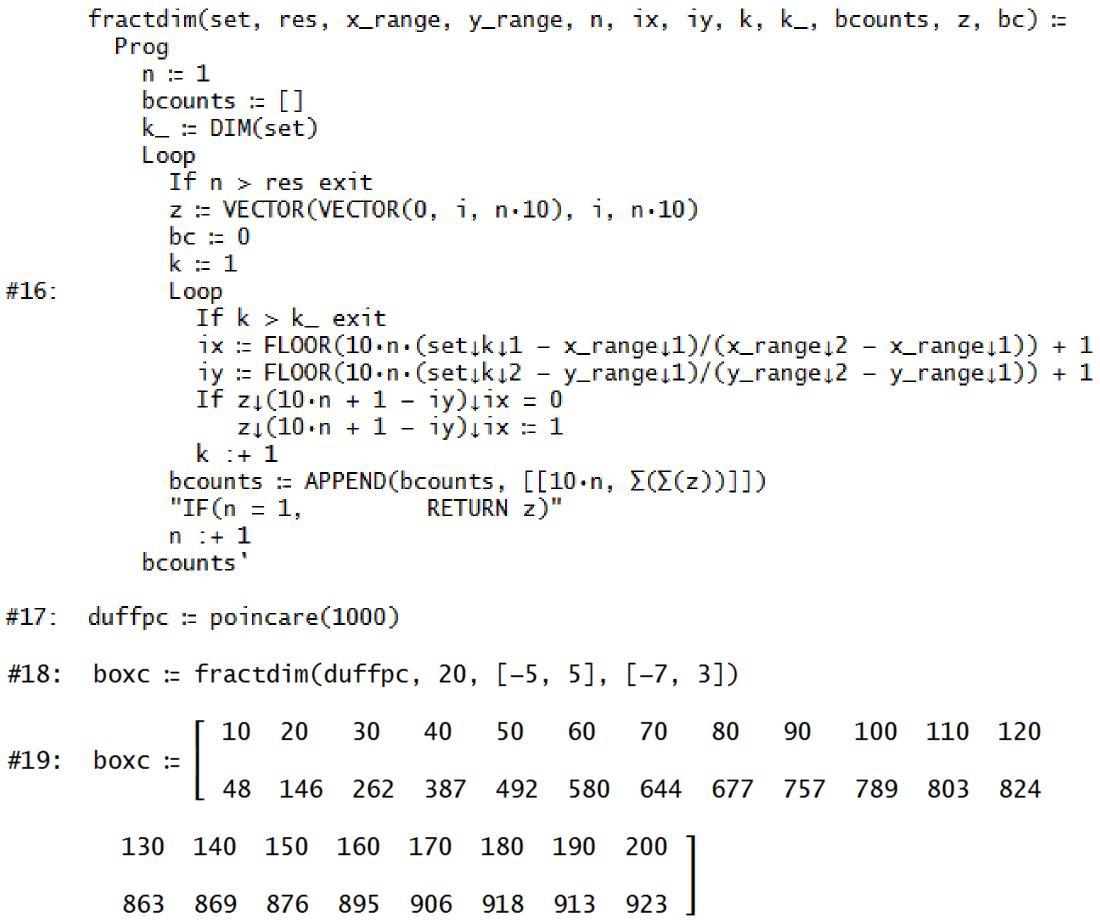
The authors propose – according to some resources – to eliminate the first seven and the last two points in order to give a suitable estimation for the fractal dimension:



```
(%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
```

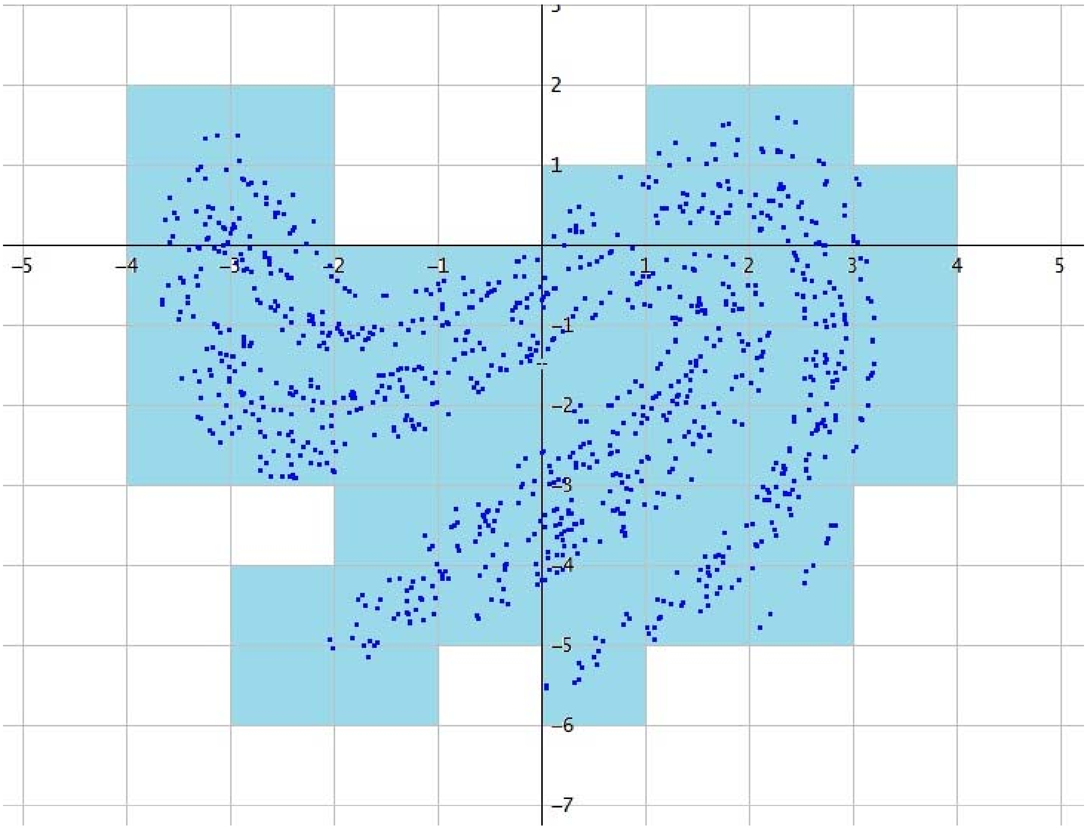
In the authors' opinion the fractal dimension of the Poincaré section of the Duffing oscillator is approximately 0.68 (slope of the regression line). I must admit that I have my concerns because I cannot explain their use log(n)/log(10) as argument for the log of the counted boxes instead of log(10n)?

However, let's try transferring this to DERIVE:

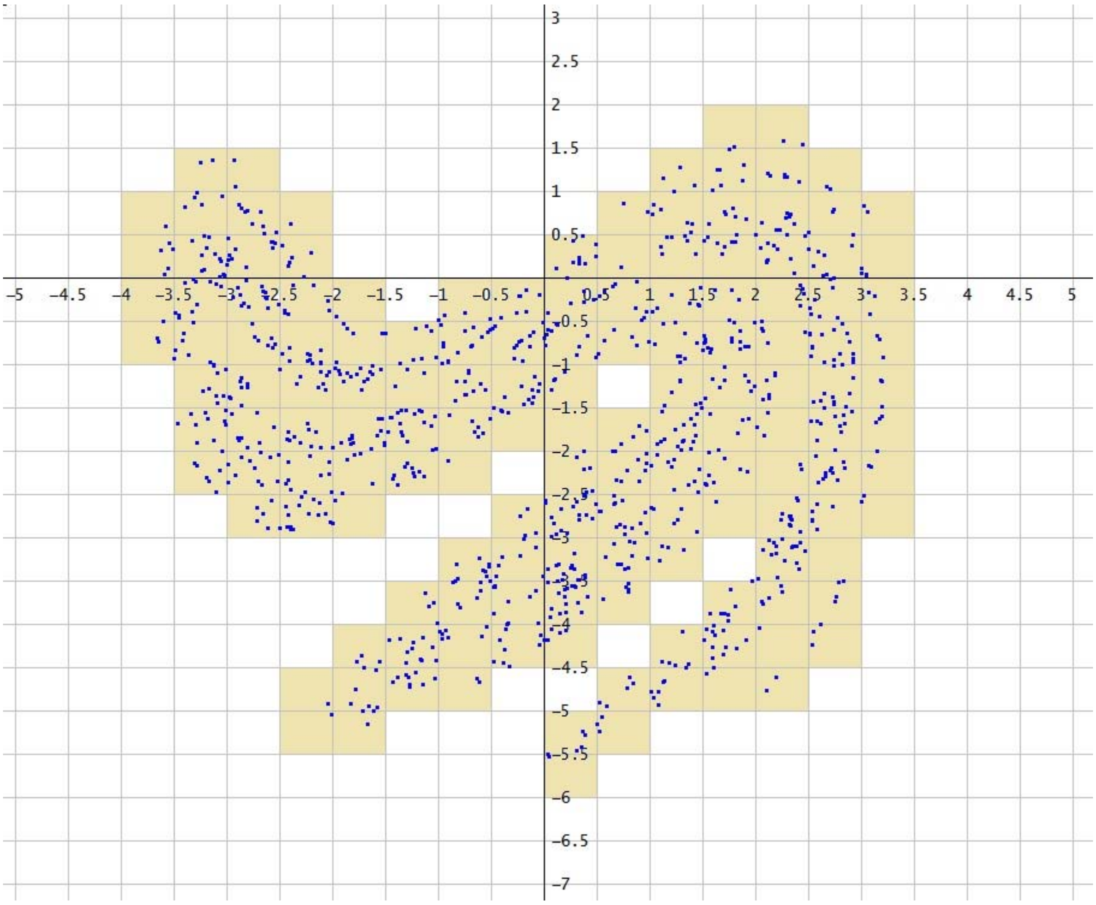


```
       fractdim(set, res, x_range, y_range, n, ix, iy, k, k_, bcounts, z, bc) :=
         Prog
           n := 1
           bcounts := []
           k_ := DIM(set)
           Loop
             If n > res exit
             z := VECTOR(VECTOR(0, i, n·10), i, n·10)
             bc := 0
             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])

               ⎡ 10   20   30   40   50   60   70   80   90   100  110  120
#19:   boxc := ⎢
               ⎣ 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: fractdim(duffpc, 20, [-5, 5], [-7, 3]) =

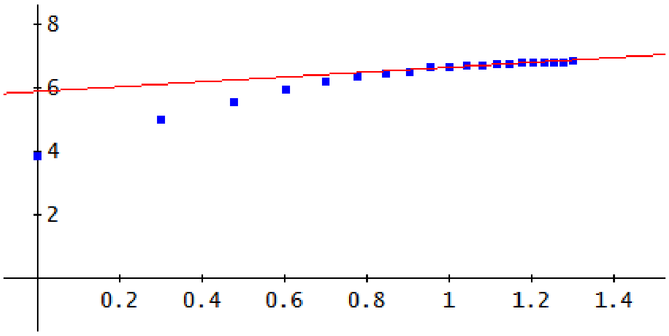
$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

You are invited to compare this matrix with the first plot on page 24.

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

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

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



The fractal dimension here is ~ 0.76.

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

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

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

On page 22 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.

4 What about 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]
```

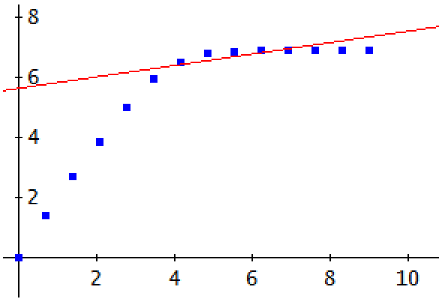
And indeed $scale^{1.29}$ gives approximately the numbers of the boxes …

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'
```

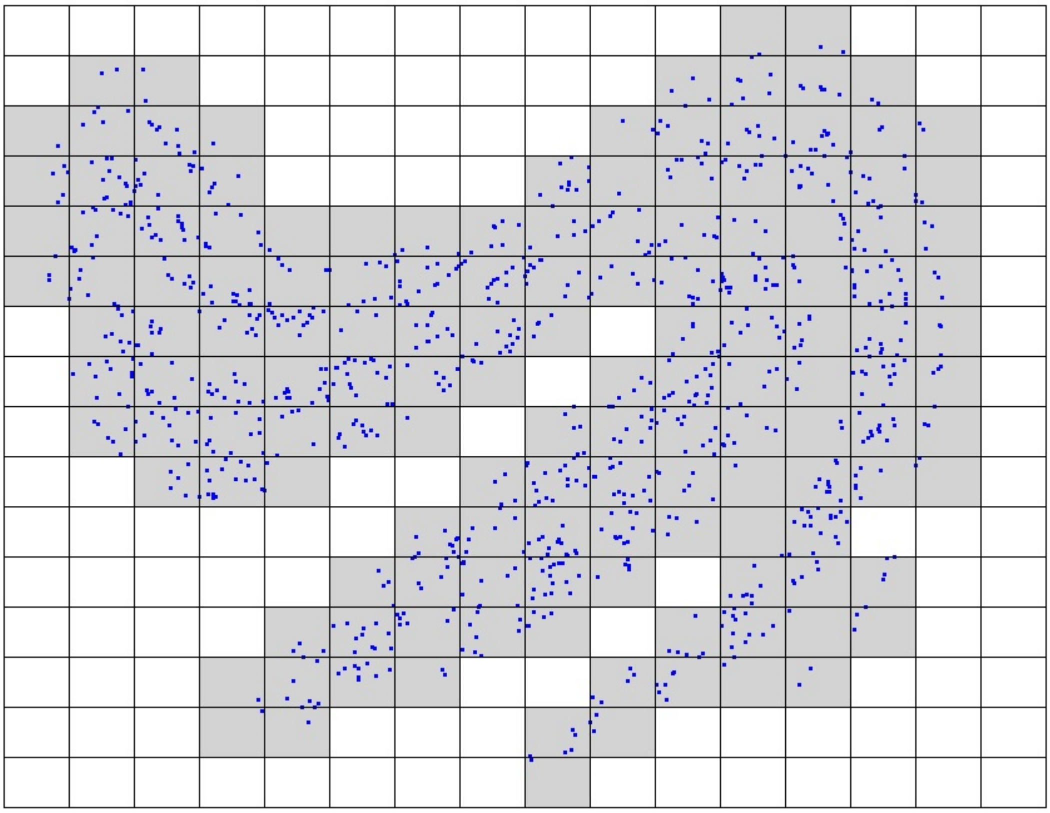
I performed all calculations from above using this new scaling for both ranges.:

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

               ┌ 1  2   4   8  16  32   64   128  256  512  1024  2048  4096  8192 ┐
#38:  boxc2 := │                                                                     │
               └ 1  4  14  32  98  286  617  856  952  985   988   991   993   996 ┘

      FIT([x, a·x + b], VECTOR([LOG(boxc2    ), LOG(boxc2    )], i, 6, 12)) = 0.24784·x + 5.2512
#39:                                      1,i             2,i

               ⎛ LOG(boxc2    )            ⎞
               ⎜          2,i              ⎟
#40:  VECTOR⎜ ─────────────────, i, 4, 12 ⎟
               ⎜ LOG(boxc2    )            ⎟
               ⎝          1,i              ⎠

#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])

                ┌ 1  2   4   8   16   32   64   128  256  512  1024  2048  4096  8192 ┐
#43:  boxc2_ := │                                                                      │
                └ 1  4  15  48  146  387  677  895  966  987   992   993   993   994 ┘

      FIT([x, a·x + b], VECTOR([LOG(boxc2_    ), LOG(boxc2_    )], i, 6, 12)) = 0.19006·x + 5.6376
#44:                                       1,i              2,i

               ⎛ LOG(boxc2_    )           ⎞
               ⎜           2,i             ⎟
#45:  VECTOR⎜ ──────────────────, i, 3, 9 ⎟ = [1.9534, 1.8616, 1.7974, 1.7192, 1.5671, 1.4008, 1.2394]
               ⎜ LOG(boxc2_    )           ⎟
               ⎝           1,i             ⎠
```

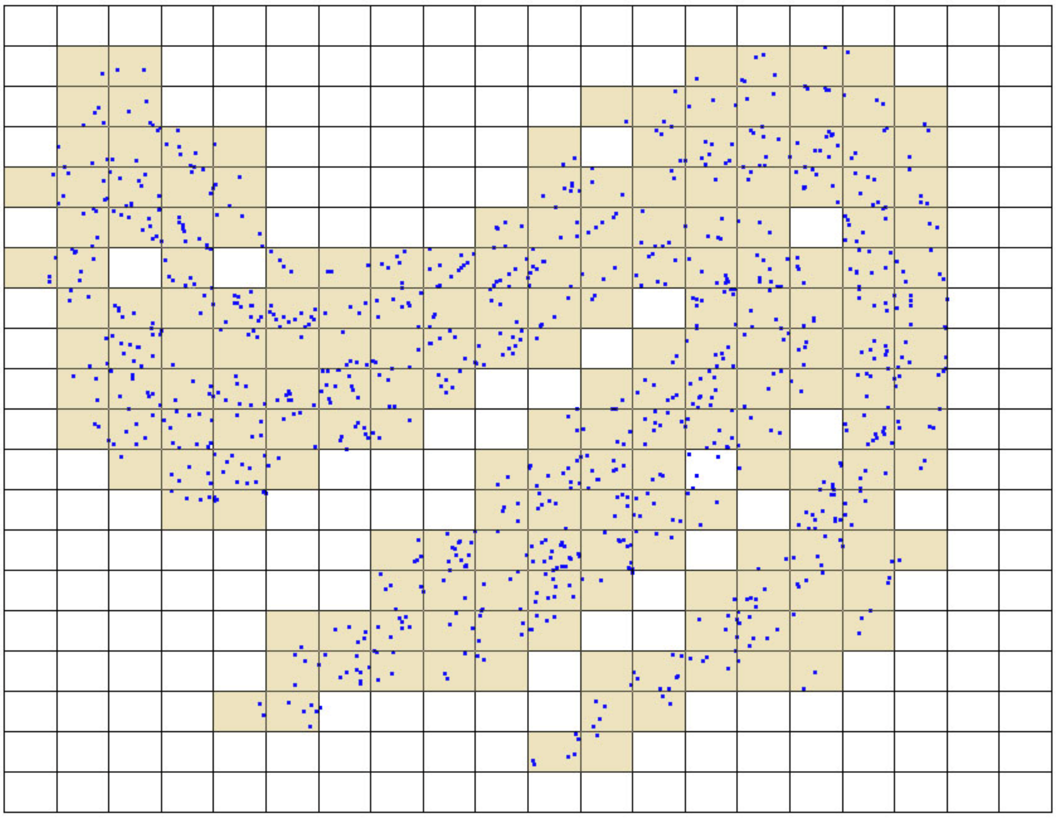


The quotient $\frac{\log N(k)}{\log \varepsilon_k}$ tends again very similar to approximately 1.30.

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!

5 Another kind of fractal dimension

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

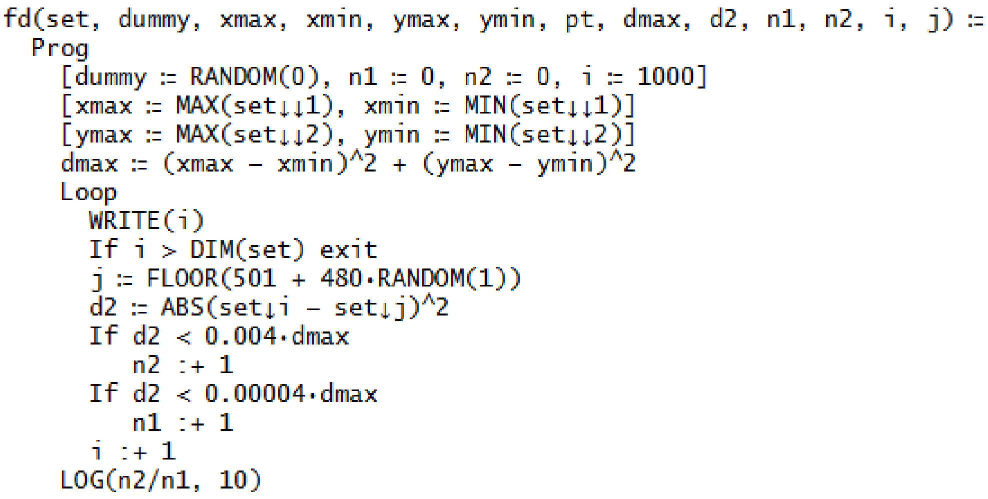
> *One method is to draw a small circle somewhere on the plane that surrounds at least one of the points. We then draw a second circle with the same center but with twice the radius. Now we count the number of points inside each circle. Let's say the smaller circle encloses n_1 points and the larger 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.*

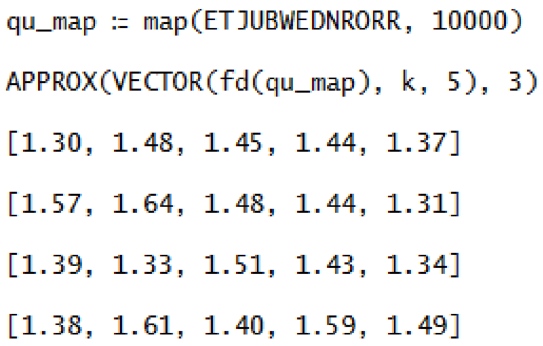
You will find more details in Sprott's book [1].

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.

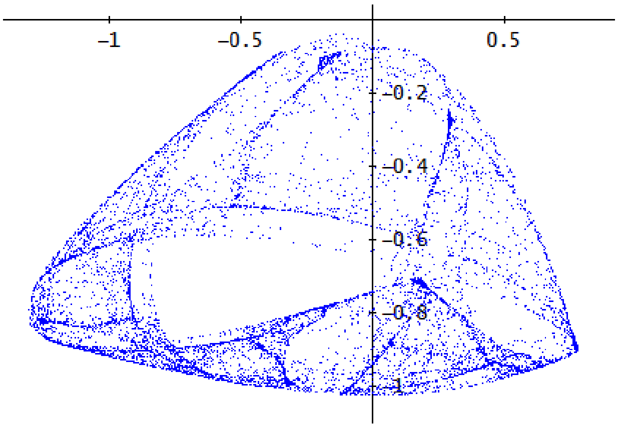


```
fd(set, dummy, xmax, xmin, ymax, ymin, pt, dmax, d2, n1, n2, i, j) :=
  Prog
    [dummy := RANDOM(0), n1 := 0, n2 := 0, i := 1000]
    [xmax := MAX(set↓↓1), xmin := MIN(set↓↓1)]
    [ymax := MAX(set↓↓2), ymin := MIN(set↓↓2)]
    dmax := (xmax - xmin)^2 + (ymax - ymin)^2
    Loop
      WRITE(i)
      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 DNL#107 to create one of Sprott's attractors and calculated the correlation dimension 20 times:

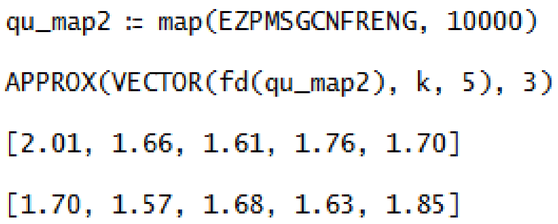


```
qu_map := map(ETJUBWEDNRORR, 10000)

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]
```

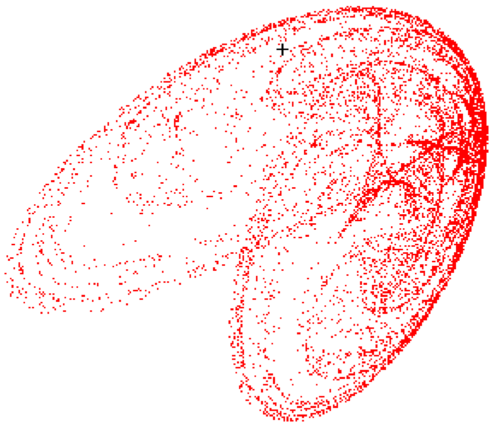


A screen shot of the BASIC program containing much more iterations gives a dimension of 1.42. Thus, my program seems to work correctly (L is the respective Lyapunov-exponent). See page 11.

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]
```



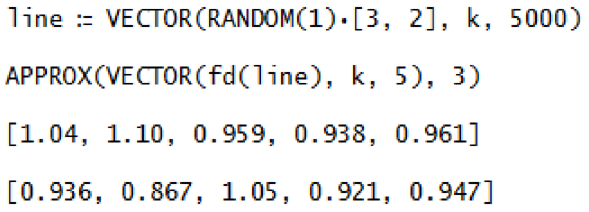
Sprott's value: F = 1.67

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

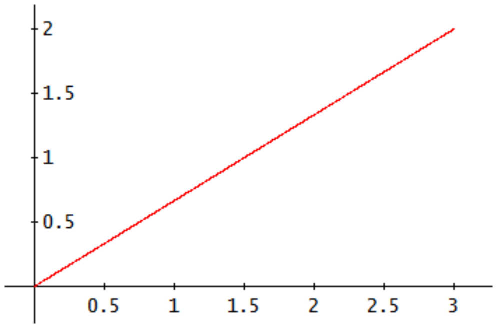
6 Dimension 1, dimension 2

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:



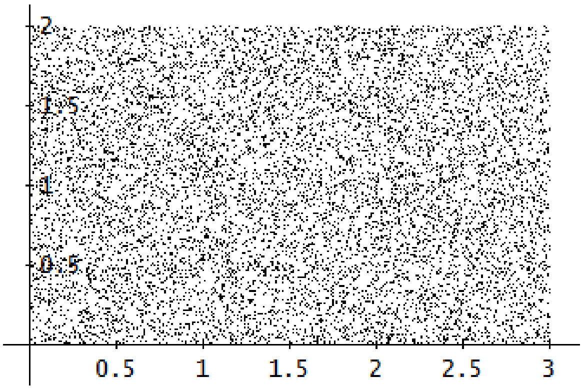
```
line := VECTOR(RANDOM(1)·[3, 2], k, 5000)

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]
```



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.

I did not transfer the dimension programs on the TI-Nspire because of lack of time, maybe later.

There are lots of respective websites and books, but I didn't find one of them performing the box-counting method step by step – there were just descriptions – many of them pretty the same!!

References (in addition to the references from page 14)

[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] Dietmar Herrmann, *Algorithmen für Fraktale*, Addison-Wesley 1994

[6] 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

More websites:

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

wonderfl.net/tag/Chaos

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

arxiv.org/

en.wikipedia.org/wiki/Fractal_dimension

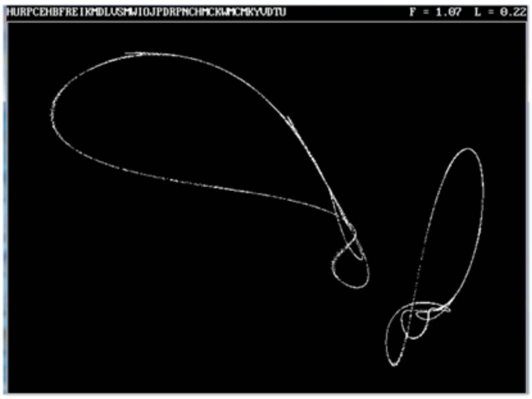
www.fast.u-psud.fr/~moisy/ml/boxcount/html/demo.html

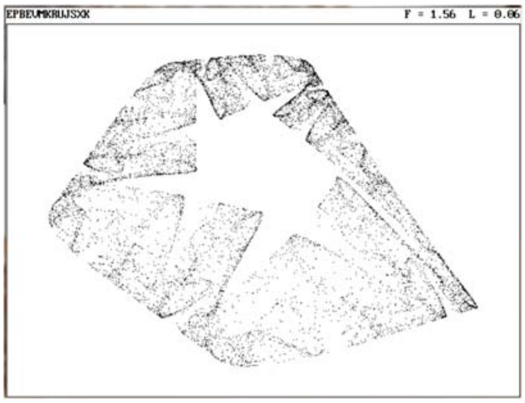
paulbourke.net/fractals/fracdim/

paulbourke.net/papers/shier2013/paper.pdf

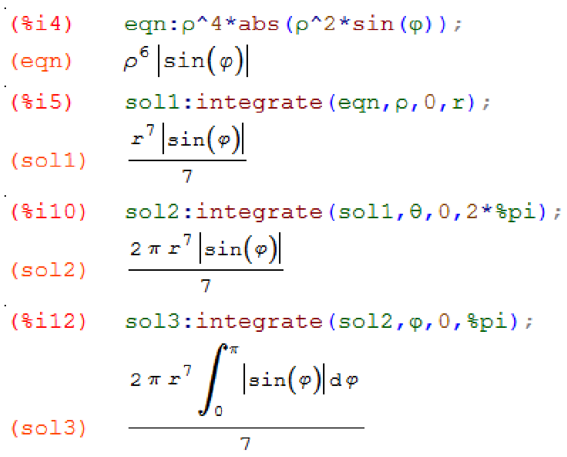
The screenshots below are two results produced by Sprott's SA.EXE which is a compiled BASIC program. It does not run under WIN 7/10. But it does work using a "DOSBox"-program.

Download DOSBox: https://sourceforge.net/projects/dosbox/





Leon Magiera forwarded a MAXIMA problem (no problem with MAPLE!). As you can see there are no problems for DERIVE and for TI-NspireCAS as well to find this triple integral stepwise or directly.



(%i4) eqn:ρ^4*abs(ρ^2*sin(φ));

(eqn) $\rho^6 \left|\sin(\varphi)\right|$

(%i5) sol1:integrate(eqn,ρ,0,r);

(sol1) $\frac{r^7 \left|\sin(\varphi)\right|}{7}$

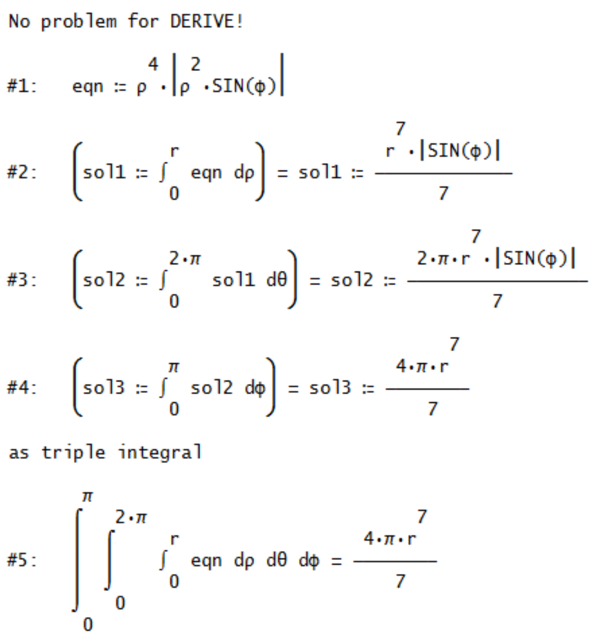
(%i10) sol2:integrate(sol1,θ,0,2*%pi);

(sol2) $\frac{2\,\pi\,r^7 \left|\sin(\varphi)\right|}{7}$

(%i12) sol3:integrate(sol2,φ,0,%pi);

(sol3) $\frac{2\,\pi\,r^7 \int_0^{\pi} \left|\sin(\varphi)\right| d\varphi}{7}$

Leon's reaction: Great DERIVE !!



No problem for DERIVE!

#1: $\text{eqn} := \rho^4 \cdot \left|\rho^2 \cdot \text{SIN}(\phi)\right|$

#2: $\left(\text{sol1} := \int_0^r \text{eqn}\ d\rho\right) = \text{sol1} := \frac{r^7 \cdot \left|\text{SIN}(\phi)\right|}{7}$

#3: $\left(\text{sol2} := \int_0^{2\cdot\pi} \text{sol1}\ d\theta\right) = \text{sol2} := \frac{2\cdot\pi\cdot r^7 \cdot \left|\text{SIN}(\phi)\right|}{7}$

#4: $\left(\text{sol3} := \int_0^{\pi} \text{sol2}\ d\phi\right) = \text{sol3} := \frac{4\cdot\pi\cdot r^7}{7}$

as triple integral

#5: $\int_0^{\pi} \int_0^{2\cdot\pi} \int_0^r \text{eqn}\ d\rho\ d\theta\ d\phi = \frac{4\cdot\pi\cdot r^7}{7}$

I am very happy to have a paper from earlier times written by a wonderful colleague and close friend. I hope that Carl will read his contribution in good shape and he will remember so many great times we could spend together in the DERIVE community, Noor and Josef

Deciding About Planting & Harvesting – An Application of the Definite Integral

Carl Leinbach, USA

NOTE: While working with this lesson, we recommend that you open a 2D-plot Window and then choose the option Window > Tile Vertically. Return to the Plot Window and choose the option Set > Plot Range. Choose a range of -50 to 400 with 9 intervals for *x* and -3 to 24 with 9 intervals for *y.* Now you will be able to follow the lesson and display your graphical results side by side.

Background

A farmer is planning to "rest" one of his fields by planting it in clover for hay. The farmer would like to get three cuttings of hay from the field. Because of the need to perform other duties around the farm, it is best if the last cutting can be made close to September 15th. The main question for the farmer is when to plant the field and when to plan for the first two cuttings.

Your task is to help the farmer plan. You do some research by contacting a local seed supplier and find that the hay requires, assuming normal weather conditions, approximately 600 hours of sunlight during the growing season to reach a maturity that would yield a proper level of nourishment for the larger farm animals (cattle and horses).

Your next step is to find a way of calculating the hours of sunlight during each day of the year. You make the assumption that this is a periodic phenomenon with a period of one year (= 365.242 days). Thus, a graph for the number of hours of daylight *t* days after the beginning of the year should follow a curve of the form:

$$a + b \cdot \sin\left(\frac{2\pi}{365.242} \cdot (t - \alpha)\right). \qquad (1)$$

The assumption is that the number of hours of daylight will follow a sinusoidal curve. The constant 2π/365.242 controls the length of the period of the curve.

Exercise 1

Substitute values for *a*, *b* and α in the formula given above. Graph the results. Can you determine what aspects of the curve are controlled by each of these parameters?

As a result of your investigations in exercise #1, you may have determined that α is the offset or phase shift of the sine curve. Thinking about the course of the sun we can set the values for our parameters. After the Vernal Equinox on March 21 the days get longer until the Summer Solstice on June 21; then they get shorter returning to equal hours of sunlight on the Autumnal Equinox on September 21, and continue getting shorter with less daylight than darkness until the Winter Solstice on December 21 when the hours of daylight are a minimum. Finally, the days become longer until the cycle repeats itself. The phase shift for the cycle must be to the position of the Vernal Equinox, or March 21, the 80th day of the year. Thus, α = 80.

$$a + b \cdot \sin\left(\frac{2\pi}{365.242} \cdot (t - 80)\right). \qquad (2)$$

The values for the other parameters depend on one's location on the globe. The farmer in our example is at Latitude 39° 55' 49" and Longitude 77° 14' 54". You can go to the web site located at the URL:

http://triton.srrb.noaa.gov/highlights/sunrise/sunrise.html (is not valid now).

https://www.esrl.noaa.gov/gmd/grad/solcalc/sunrise.html (this is the correct URL for 2018).

which is run by NOAA, to find the following information for this location.

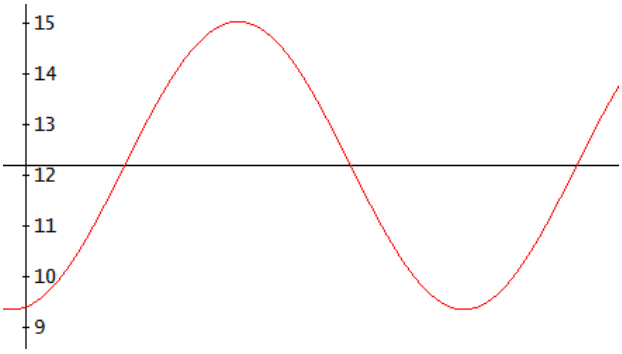
Date	Sunrise	Sunset	Hours
Mar 21	6:11	18:22	12:11
Jun 21	5:40	20:41	15:01
Sep 21	6:56	19:08	12:12
Dec 21	7:27	16:47	9:20

Exercise 2

The website mentioned above will help you to find Latitude and Longitude (Sunrise & Sunset) for your location. Using this information construct a table similar to the one given above. (see appropriate websites at the end of the article, Josef)

From the information in the table we can substitute into formula (2) for the hours of daylight in our area. Note that the hours of daylight for the Vernal and Autumnal Equinoxes differs by a minute. We will use their average as our baseline. We will also need to make a slight adjustment for the two solstices. This results from our model being an approximation to the physical phenomenon. All of the adjustments are in the second decimal place.

$$\#1:\quad 12.192 + 2.84\cdot\mathrm{SIN}\left(\frac{2\cdot\pi}{365.242}\cdot(t - 80)\right)$$



Exercise 3

Using the information from your area, write the expression for the hours of daylight on day, *t*, that is analog of expression #1 for your location.

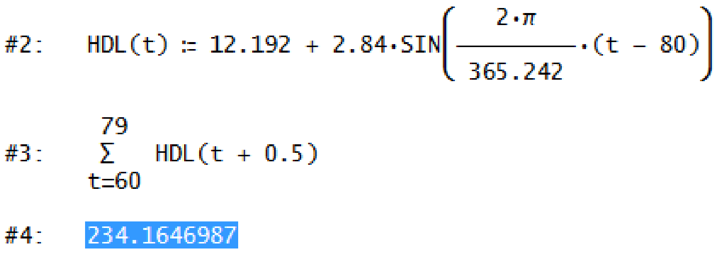
Now we are ready to solve the farmer’s problem.

Relating Total Hours of Daylight to the Integral

What is this assignment doing in a section on the Definite Integral? Stated simply, the integral is an extension of the idea of summing the values of a function over an interval. Let's take an example.

In the plot window restrict the range of the t-variable to 50 to 100 with 5 intervals and the y-variable to -3 to 15 with 6 intervals. This is done using the Set > Plot Range dialog box. If we want to calculate the total number of hours of daylight from day 60 (March 1) to day 80 (March 21), we can take a representative value for the hours of daylight function for each day and add those values. In expression #3 we have done just that. We divided the interval from 20 to 80 into subintervals of one day and chose the midpoint of the interval as the representative value.

Approximate expression #3.

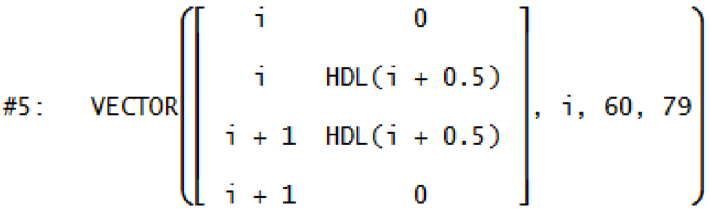


#2: $\mathrm{HDL}(t) := 12.192 + 2.84 \cdot \mathrm{SIN}\left(\dfrac{2 \cdot \pi}{365.242} \cdot (t - 80)\right)$

#3: $\displaystyle\sum_{t=60}^{79} \mathrm{HDL}(t + 0.5)$

#4: 234.1646987

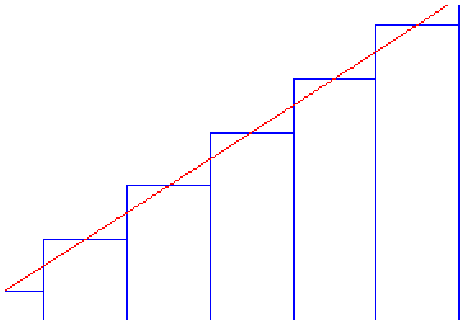
We can represent this sum graphically by plotting the following points and connecting them.

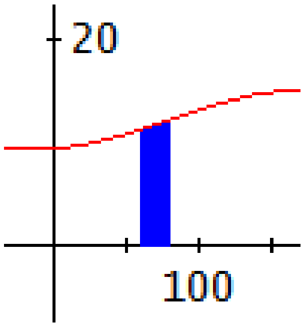


#5: $\mathrm{VECTOR}\left(\begin{bmatrix} i & 0 \\ i & \mathrm{HDL}(i + 0.5) \\ i + 1 & \mathrm{HDL}(i + 0.5) \\ i + 1 & 0 \end{bmatrix}, i, 60, 79\right)$

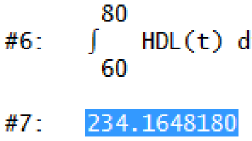
Plot this expression and enlarge the plot about the tops of the rectangles. Then return plot setting to those given prior to expressions.

When you looked at a close up of the tops of the rectangles, you saw that essentially we had as much area under the curve lying above of the rectangle as we had area of the rectangle lying above of the curve. In other words, the area under the curve should be the same as the sum of the areas of the rectangles. But the sum of the areas of the rectangles (remember each rectangle has width 1) is the va1ue of expression #6, the total number of hours of daylight. Also, the graph of expression #7 looks very much like an approximating sum for the definite integral. Thus, the value of the sum and the value of the definite integral should be approximately the same.





Check this statement by simplifying the following expression. Compare your result with that of simplifying expression #3.



#6: $\int_{60}^{80} \text{HDL}(t)\,dt$

#7: 234.1648180

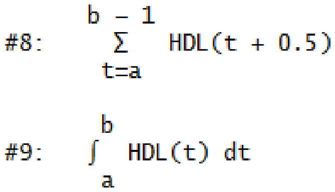
The agreement is not quite perfect, but it is quite close.

Exercise 4

In the example we choose an interval where the graph of the function was rather straight. Choose an interval of about 20 days that contains one of the solstices in its interior and repeat the comparison of the number of hours of daylight over the interval and the value of the definite integral. Comment on the accuracy of the approximation and issues affecting the integral as an estimate.

Helping the Framer Plan

We have seen that in order to calculate the number of days of daylight for a group of consecutive days from $t = a$ to $t = b$ can be done in two ways:

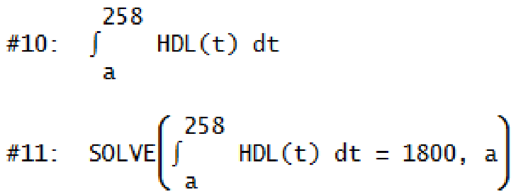


#8: $\sum_{t=a}^{b-1} \text{HDL}(t + 0.5)$

#9: $\int_{a}^{b} \text{HDL}(t)\,dt$

If the goal is to simply find the total hours of daylight between day a and day b, then expression #8 will give the best answer. The time to perform the calculation is not appreciably longer than using the approximation given by expression #9. However, the farmers question is of a different nature. We know b which is 258 (the value of t corresponds to September 15). What we want to do is find the value for a that will yield a value approximately 1800 = 3 ×600. In this situation the form of expression #9 makes sense. The answer will be the solution to expression #11.

Simplify expression # 10 and then find the solution to the farmer's problem using expression #11.



#10: $\int_{a}^{258} \text{HDL}(t)\,dt$

#11: $\text{SOLVE}\left(\int_{a}^{258} \text{HDL}(t)\,dt = 1800,\ a\right)$

What day of this year is this? It remains for you to complete the farmer's schedule by determining the dates when the first and second cuttings should be done.

Further Exercises

Find the dates for the first and second cuttings for the farmer. Remember that you need 600 hours off daylight between cuttings and also 600 hours from planting to the first cutting. You can do this either by working backwards from September 14 or forwards from the date found as a result of simplifying expression #11.

Work out a planting and harvesting schedule for clover for a farmer in your location.

Project: Another phenomenon that is cyclic and related to planting is called Growing Degree Days (= Wachstumsgradtage WGT in German). The GDD for any day is given by taking the average Maximum Temperature for the day and the Minimum Temperature (using degrees Fahrenheit) and subtracting 50. Using a weather page (your local TV or radio station may have such a web page, NOAA is another source, or one of the channels that specialize in weather forecasts may have one) that archives the average highs and lows for each date, choose the 15th of each month and plot the average GDD for these dates. Fit a sine curve such as formula (1) to the date by approximating the maximum and minimum GDD values for the year. Now go to a local seed dealer and get a seed catalogue for corn or another crop of interest. These catalogs list the total number of GGD from planting to harvest. Work out a planting schedule for the farmer to plant this crop in your area.

Carl Leinbach: leinbachgettysburg.edu

References and Resources

https://www.zamg.ac.at/cms/de/klima/klimauebersichten/ephemeriden

http://www.timeadate.eu/pages/en/index-en.html

https://en.wikipedia.org/wiki/Growing_degree-day

https://de.wikipedia.org/wiki/Wachstumsgradtag

http://www.wetterstation-goettingen.de/dokumente/wachstumsgradtag.pdf

http://www.farmwest.com/node/936

https://mrcc.illinois.edu/gismaps/gddinfo.htm

Another root-finding algorithm for polynomials

Francisco Marcelo Fernández from Argentina sent a request which let me know another numerical method for solving univariate polynomial equations. He wrote:

Dear Josef,
I would like to know algorithm is built in Derive for the calculation of the roots of a polynomial. I am attaching an example. I can produce many others of that kind.

Best regards,
Marcelo

Marcelo's example containing the reason for his question is given after the respective mails. I sent Marcelo's request to David Stoutemyer and he answered very soon and very competent:

David Stoutemyer (1)

Aloha Josef!,

The numerical polynomial zero approximator is based on Laguerre's method, as described in "A first course in numerical analysis" by Anthony Ralston.

It has better global convergence properties than Newton's method, the code is compact, and unlike the popular QR method, it doesn't require storing n by n matrices for a degree n -1 polynomial.

There are better methods for when you only want real roots or only the closest root to a starting guess. Akritas, who you might have met at some ACA meeting, has written a series of articles about one of them based on an idea of Vincent.

-- best regards, david

Francisco Marcelo Fernández, Argentina

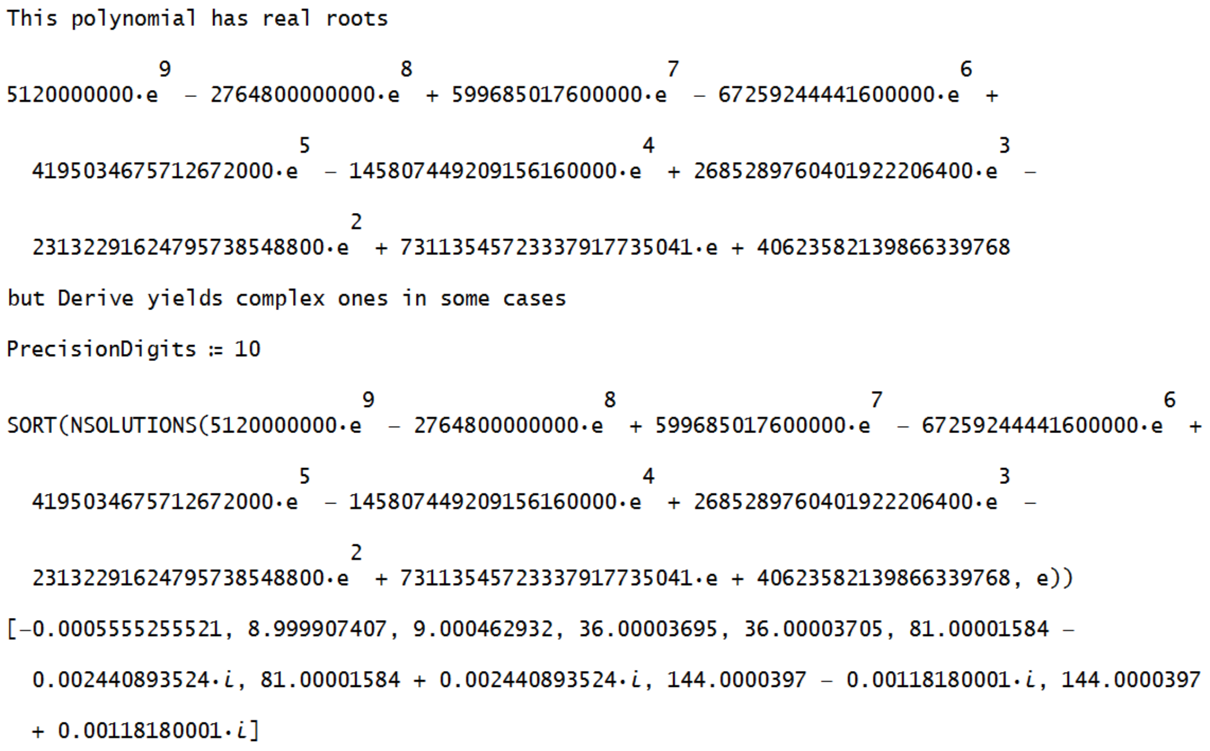
Dear Josef,

Thank you very much for your answer. As far as I understand, such an algorithm assumes that there are no roots close together which may be the reason of the failure in this case.

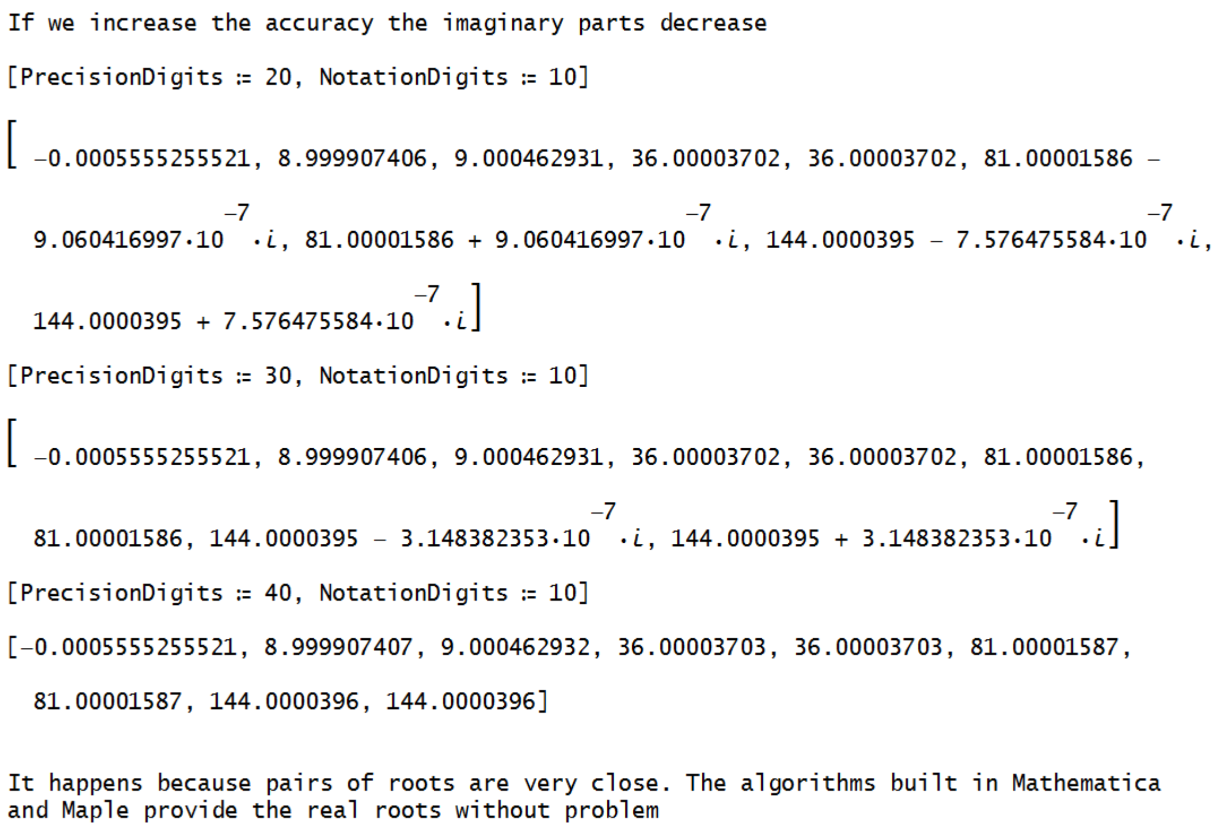
In Ralston, page 382 it is said that "When all zeros are real, we can show that H is always nonnegative". The square root of H(xi) appears to be the only source of complex eigenvalues. However, the polynomial I sent you has only real roots and the Laguerre's method yields complex roots if the precision is not sufficiently great.

Best regards,

Marcelo



```
This polynomial has real roots

            9                  8                    7                      6
5120000000·e  - 2764800000000·e  + 599685017600000·e  - 67259244441600000·e  +

                      5                         4                         3
  4195034675712672000·e  - 1458074492091561600000·e  + 268528976040192206400·e  -

                          2
  23132291624795738548800·e  + 73113545723337917735041·e + 40623582139866339768

but Derive yields complex ones in some cases

PrecisionDigits := 10

                            9                  8                    7                      6
SORT(NSOLUTIONS(5120000000·e  - 2764800000000·e  + 599685017600000·e  - 67259244441600000·e  +

                      5                         4                         3
  4195034675712672000·e  - 1458074492091561600000·e  + 268528976040192206400·e  -

                          2
  23132291624795738548800·e  + 73113545723337917735041·e + 40623582139866339768, e))

[-0.0005555255521, 8.999907407, 9.000462932, 36.00003695, 36.00003705, 81.00001584 -

  0.002440893524·ⅈ, 81.00001584 + 0.002440893524·ⅈ, 144.0000397 - 0.00118180001·ⅈ, 144.0000397

  + 0.00118180001·ⅈ]
```



If we increase the accuracy the imaginary parts decrease

[PrecisionDigits := 20, NotationDigits := 10]

$[-0.0005555255521, 8.999907406, 9.000462931, 36.00003702, 36.00003702, 81.00001586 - 9.060416997\cdot10^{-7}\cdot i, 81.00001586 + 9.060416997\cdot10^{-7}\cdot i, 144.0000395 - 7.576475584\cdot10^{-7}\cdot i, 144.0000395 + 7.576475584\cdot10^{-7}\cdot i]$

[PrecisionDigits := 30, NotationDigits := 10]

$[-0.0005555255521, 8.999907406, 9.000462931, 36.00003702, 36.00003702, 81.00001586, 81.00001586, 144.0000395 - 3.148382353\cdot10^{-7}\cdot i, 144.0000395 + 3.148382353\cdot10^{-7}\cdot i]$

[PrecisionDigits := 40, NotationDigits := 10]

$[-0.0005555255521, 8.999907407, 9.000462932, 36.00003703, 36.00003703, 81.00001587, 81.00001587, 144.0000396, 144.0000396]$

It happens because pairs of roots are very close. The algorithms built in Mathematica and Maple provide the real roots without problem

I (Josef) must admit that I didn't even know about the existence of *Laguerre*'s method. The respective algorithm runs as follows:

We try to find one root of polynomial $p(x)$ of degree n.

Start with an initial guess x_0

Start a loop for $k = 0, 1, 2, \ldots$

- If $p(x_k)$ is sufficiently small, exit the loop
- Calculate $G = \dfrac{p'(x_k)}{p(x_k)}$
- Calculate $H = G^2 - \dfrac{p''(x_k)}{p(x_k)}$
- Calculate $a = \dfrac{n}{G \pm \sqrt{(n-1)(nH-G^2)}}$. Take the sign which gives the denominator the larger absolute value
- Set $x_{k+1} = x_k - a$
- If a is small enough or if the maximum number of iterations has been reached, a root has been found.

If a root has been found, then use the corresponding linear factor to deflate the degree of the polynomial by one. So, step by step you can find approximations of all roots. Round off errors are least if the roots are found in order of increasing magnitude.

More details including the proof can be found at:

https://en.wikipedia.org/wiki/Laguerre%27s_method
http://mathworld.wolfram.com/LaguerresMethod.html
http://www.aip.de/groups/soe/local/numres/bookfpdf/f9-5.pdf
http://mathfaculty.fullerton.edu/mathews/n2003/laguerresethod/LaguerreMethodBib/Links/LaguerreMethodBib_lnk_1.html

https://en.wikipedia.org/wiki/Vincent%27s_theorem
ftp://ftp.pdmi.ras.ru/pub/publicat/znsl/v373/p005.pdf
https://faculty.e-ce.uth.gr/akritas/publications.htm
https://people.mpi-inf.mpg.de/~mehlhorn/ftp/VikramContinuedFractions.pdf

http://www.math.cornell.edu/~hubbard/NewtonInventiones.pdf

Maybe that the following procedure might be a good way to introduce this method in class room. Let’s start performing the algorithm step by step (using DERIVE or any other CAS):

We try to find all roots of the polynomial given in expression #10.

#1: [Precision := Approximate, PrecisionDigits := 20]

#2: [Notation := Scientific, NotationDigits := 20]

#3: p(x) :=

#4: $\left[\mathrm{p1}(x) :\!= \frac{d}{dx}\, p(x),\ \mathrm{p2}(x) :\!= \left(\frac{d}{dx}\right)^2 p(x)\right]$

#5: $\left[G(x) :\!= \frac{\mathrm{p1}(x)}{p(x)},\ H(x) :\!= G(x)^2 - \frac{\mathrm{p2}(x)}{p(x)}\right]$

#6: $\mathrm{den1}(x, n) :\!= G(x) + \sqrt{((n - 1)\cdot(n\cdot H(x) - G(x)^2))}$

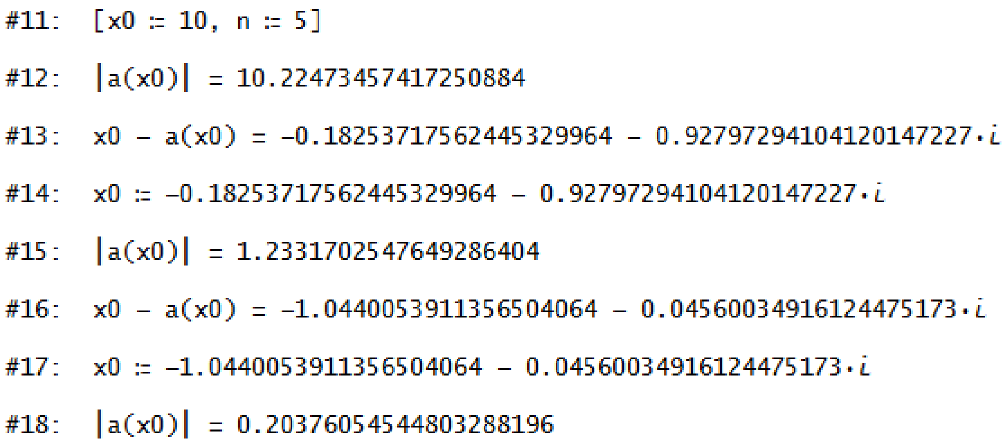
#7: $\mathrm{den2}(x, n) :\!= G(x) - \sqrt{((n - 1)\cdot(n\cdot H(x) - G(x)^2))}$

```
      a(x, n) :=
        If ABS(den1(x, n)) > ABS(den2(x, n))
#8:        n/den1(x, n)
           n/den2(x, n)
           n/den1(x, n)
```

#9: ε = 10^-10

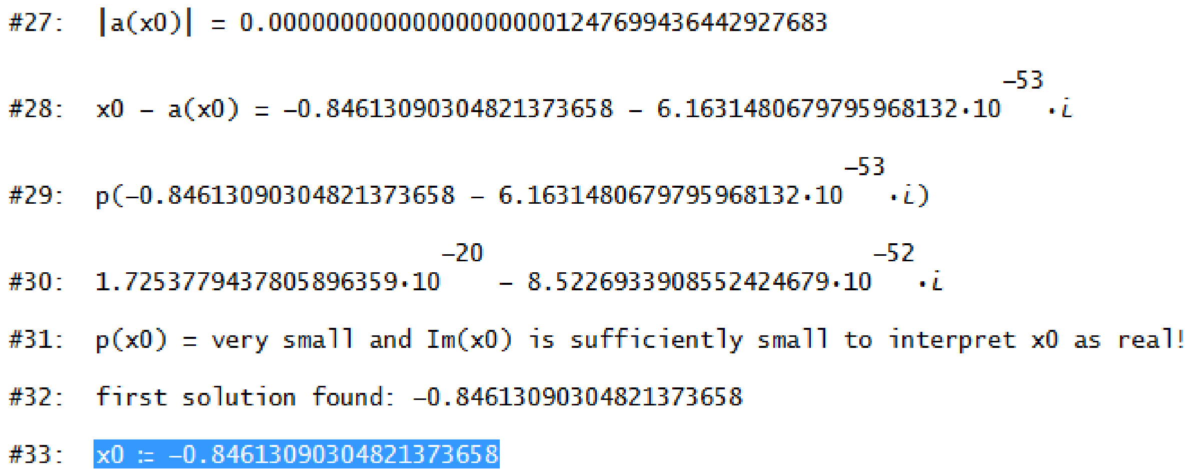
#10: $p(x) :\!= 0.3\cdot x^5 + 0.4\cdot x^4 + 0.3\cdot x^3 - 2\cdot x^2 + 10\cdot x + 10$

It’s good practice to take the constant of the polynomial as initial value.



```
#11:  [x0 := 10, n := 5]

#12:  |a(x0)| = 10.22473457417250884

#13:  x0 - a(x0) = -0.18253717562445329964 - 0.92797294104120147227·ί

#14:  x0 := -0.18253717562445329964 - 0.92797294104120147227·ί

#15:  |a(x0)| = 1.2331702547649286404

#16:  x0 - a(x0) = -1.0440053911356504064 - 0.04560034916124475173·ί

#17:  x0 := -1.0440053911356504064 - 0.04560034916124475173·ί

#18:  |a(x0)| = 0.20376054544803288196
```

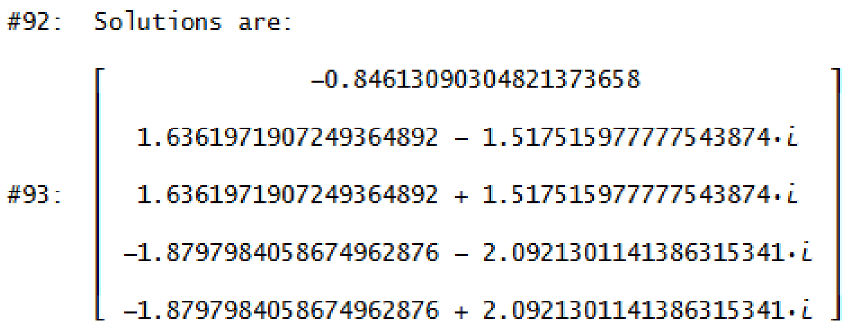
After several steps:



```
#27:  |a(x0)| = 0.000000000000000000001247699436442927683

                                                                  -53
#28:  x0 - a(x0) = -0.84613090304821373658 - 6.1631480679795968132·10    ·ί

                                                          -53
#29:  p(-0.84613090304821373658 - 6.1631480679795968132·10    ·ί)

                                     -20                         -52
#30:  1.7253779437805896359·10     - 8.5226933908552424679·10    ·ί

#31:  p(x0) = very small and Im(x0) is sufficiently small to interpret x0 as real!

#32:  first solution found: -0.84613090304821373658

#33:  x0 := -0.84613090304821373658
```

We use the DERIVE-command `QUOTIENT(p(x), x - x0)` to divide *p*(*x*) by the linear factor in order to obtain a polynomial of degree 4 and repeat the procedure with `[x0 := 11.818502271899866827, n := 4]`. This leads to the first complex root, the next solution – which is our third now – is its conjugate. Then we can solve the quadratic or apply the algorithm once more …

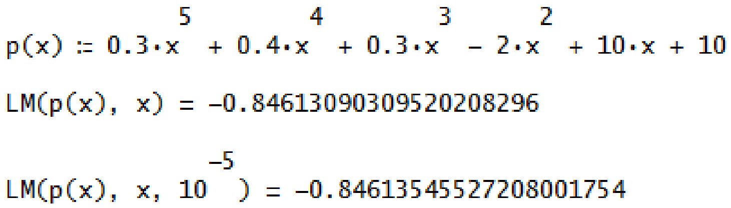
This is the table of all “hand-made” roots:



```
#92:  Solutions are:

      ┌               -0.84613090304821373658               ┐
      │                                                     │
      │  1.6361971907249364892 - 1.517515977777543874·ί     │
      │                                                     │
#93:  │  1.6361971907249364892 + 1.517515977777543874·ί     │
      │                                                     │
      │ -1.8797984058674962876 - 2.0921301141386315341·ί    │
      │                                                     │
      └ -1.8797984058674962876 + 2.0921301141386315341·ί    ┘
```

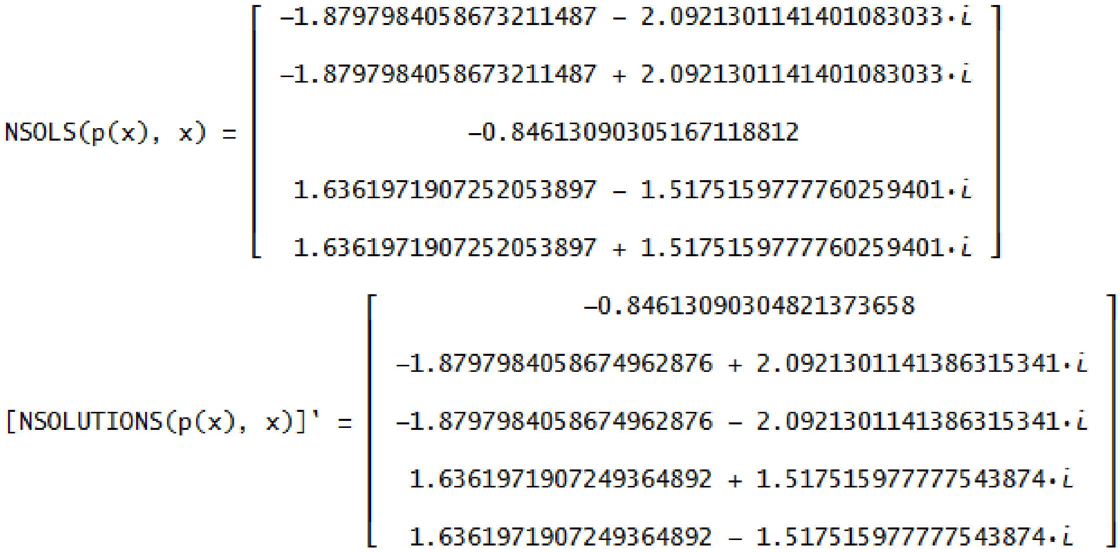
(DERIVE-file Laguerre.dfw)

You will imagine the next step: gathering all steps in a program. Right guess!
But I will not bore you with the program code, I will just show how it works.



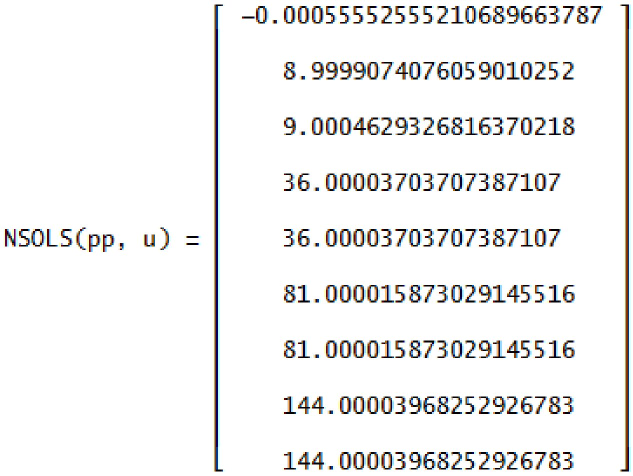
```
                5          4          3       2
p(x) := 0.3·x  + 0.4·x  + 0.3·x  - 2·x  + 10·x + 10

LM(p(x), x) = -0.84613090309520208296

                 -5
LM(p(x), x, 10   ) = -0.84613545527208001754
```

My DERIVE-program `NSOLS(polynomial, variable, [epsilon])` performs the complete calculation. Epsilon is 10^{-10} by default.



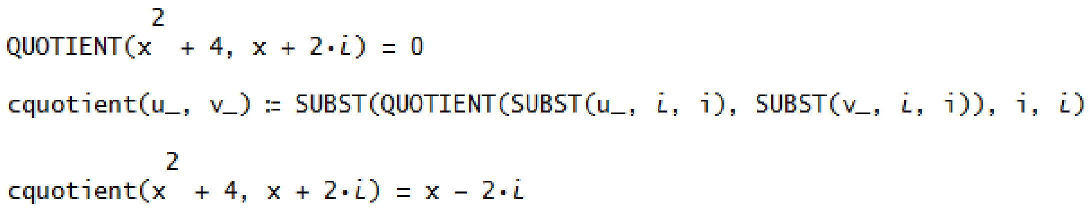
```
                  ┌ -1.8797984058673211487 - 2.0921301141401083033·ί ┐
                  │                                                  │
                  │ -1.8797984058673211487 + 2.0921301141401083033·ί │
                  │                                                  │
NSOLS(p(x), x) =  │             -0.84613090305167118812              │
                  │                                                  │
                  │  1.6361971907252053897 - 1.5175159777760259401·ί │
                  │                                                  │
                  └  1.6361971907252053897 + 1.5175159777760259401·ί ┘

                       ┌                -0.84613090304821373658                ┐
                       │                                                       │
                       │ -1.8797984058674962876 + 2.0921301141386315341·ί      │
                       │                                                       │
[NSOLUTIONS(p(x), x)]' = │ -1.8797984058674962876 - 2.0921301141386315341·ί  │
                       │                                                       │
                       │  1.6361971907249364892 + 1.517515977777543874·ί       │
                       │                                                       │
                       └  1.6361971907249364892 - 1.517515977777543874·ί       ┘
```

This is NSOLS applied on Marcelo’s polynomial of degree 9:



```
                ┌ -0.00055552555210689663787 ┐
                │                            │
                │    8.9999074076059010252   │
                │                            │
                │    9.0004629326816370218   │
                │                            │
                │    36.00003703707387107    │
                │                            │
NSOLS(pp, u) =  │    36.00003703707387107    │
                │                            │
                │    81.000015873029145516   │
                │                            │
                │    81.000015873029145516   │
                │                            │
                │    144.00003968252926783   │
                │                            │
                └    144.00003968252926783   ┘
```

I faced one problem: DERIVE’s `QUOTIENT` function (needed for dividing by the linear factors) does not work for complex solution. This is not necessary for polynomial with all real coefficients but I wanted to apply NSOLS on complex polynomials, too. So, let’s work around!



```
           2
QUOTIENT(x  + 4, x + 2·ί) = 0

cquotient(u_, v_) := SUBST(QUOTIENT(SUBST(u_, ί, i), SUBST(v_, ί, i)), i, ί)

            2
cquotient(x  + 4, x + 2·ί) = x - 2·ί
```

This is another example to compare with NSOLUTIONS followed by a complex polynomial and its roots.

$$\mathrm{NSOLS}(x^3 + 3\cdot x^2 - 4\cdot x + 10,\ x)$$

$$\begin{bmatrix} -4.417802089290254806 \\ 0.70890104464890047869 - 1.3270374302005337394\cdot \imath \\ 0.70890104464890047869 + 1.3270374302005337394\cdot \imath \end{bmatrix}$$

$$\left[\mathrm{NSOLUTIONS}(x^3 + 3\cdot x^2 - 4\cdot x + 10,\ x)\right]'$$

$$\begin{bmatrix} 0.708901044645127403 + 1.3270374302059543505\cdot \imath \\ 0.708901044645127403 - 1.3270374302059543505\cdot \imath \\ -4.417802089290254806 - 1.3483072590143257929\cdot 10^{-45}\cdot \imath \end{bmatrix}$$

$$\mathrm{mysols} := \mathrm{NSOLS}(x^3 + 2\cdot x - 3\cdot x\cdot \imath + 2 - \imath,\ x)$$

$$\mathrm{mysols} := \begin{bmatrix} -0.58025547748239283160 - 0.25137126967171118897\cdot \imath \\ -0.52968858909384196706 - 1.5705544688536791818\cdot \imath \\ 1.1099440665746921927 + 1.8219257385072329862\cdot \imath \end{bmatrix}$$

$$\mathrm{Dsols} := \left[\mathrm{NSOLUTIONS}(x^3 + 2\cdot x - 3\cdot x\cdot \imath + 2 - \imath = 0,\ x)\right]'$$

$$\mathrm{Dsols} := \begin{bmatrix} 1.1099440665811726488 + 1.8219257385105477492\cdot \imath \\ -0.52968858908289602246 - 1.5705544688604104047\cdot \imath \\ -0.58025547749827662634 - 0.25137126965013734448\cdot \imath \end{bmatrix}$$

Let's compare the accuracy! DERIVE's internal implementation performs much better but this does not surprise me.

$$\mathrm{VECTOR}\left(\left|x^3 + 2\cdot x - 3\cdot x\cdot \imath + 2 - \imath\right|,\ x,\ \mathrm{mysols}'_1\right)$$

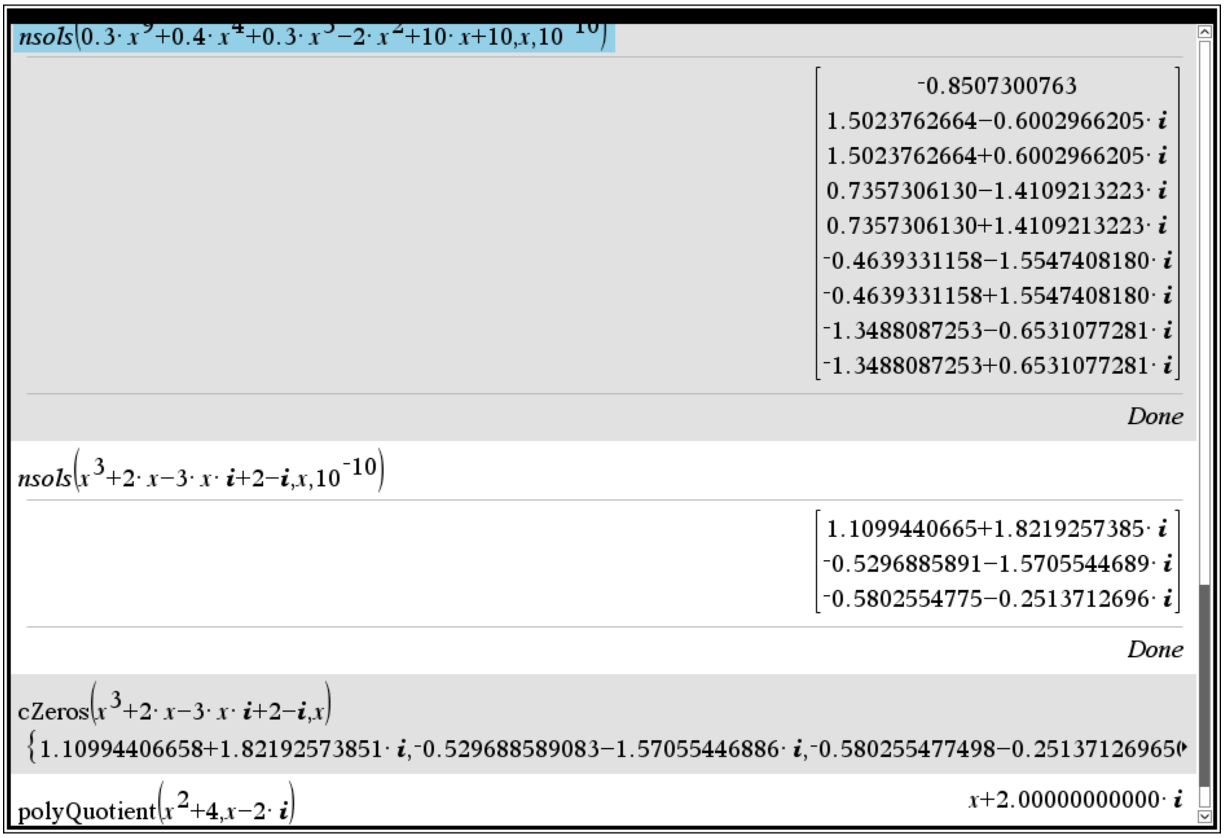
$$\left[9.4605889868321699247\cdot 10^{-11},\ 6.3919144483520131172\cdot 10^{-11},\ 7.3365241581414797116\cdot 10^{-11}\right]$$

$$\mathrm{VECTOR}\left(\left|x^3 + 2\cdot x - 3\cdot x\cdot \imath + 2 - \imath\right|,\ x,\ \mathrm{Dsols}'_1\right)$$

$$\left[3.2733854962417922621\cdot 10^{-38},\ 3.2767157315353793868\cdot 10^{-38},\ 3.2751527584844037062\cdot 10^{-38}\right]$$

(DERIVE-file Laguerre_prog.dfw)

nsols works also on TI-NspireCAS. polyQuotient performs the division of complex polynomials.



```
nsols(0.3·x^9+0.4·x^4+0.3·x^3−2·x^2+10·x+10,x,10^-10)
                                              ⎡            -0.8507300763 ⎤
                                              ⎢1.5023762664−0.6002966205·i⎥
                                              ⎢1.5023762664+0.6002966205·i⎥
                                              ⎢0.7357306130−1.4109213223·i⎥
                                              ⎢0.7357306130+1.4109213223·i⎥
                                              ⎢-0.4639331158−1.5547408180·i⎥
                                              ⎢-0.4639331158+1.5547408180·i⎥
                                              ⎢-1.3488087253−0.6531077281·i⎥
                                              ⎣-1.3488087253+0.6531077281·i⎦
                                                                        Done
nsols(x^3+2·x−3·x·i+2−i,x,10^-10)
                                              ⎡1.1099440665+1.8219257385·i⎤
                                              ⎢-0.5296885891−1.5705544689·i⎥
                                              ⎣-0.5802554775−0.2513712696·i⎦
                                                                        Done
cZeros(x^3+2·x−3·x·i+2−i,x)
 {1.10994406658+1.82192573851·i,-0.529688589083−1.57055446886·i,-0.580255477498−0.25137126965
polyQuotient(x^2+4,x−2·i)                               x+2.00000000000·i
```

As you can see it is possible to find roots of an order 9 polynomial. But I was not able to apply *nsols* on Marcelo's order 9 polynomial on the TI. I believe that the coefficients are too great, sorry. I sent my program to David and he provided a lot of useful hints. I could build in some of them.

Many thanks, David

Dear Josef,

Actually, Laguerre's algorithm addresses only refining an initial guess for one zero. The algorithm implemented in Derive also entails:

1. 1st doing a (not always successful) square-free decomposition to avoid multiple roots.
2. deflating the polynomial after each zero has converged so as not to reconverge to a zero that has already been found,
3. "polishing" each zero with the original polynomial to overcome inaccuracies due to the deflation.
4. making each initial guess at an estimated centroid of the remaining smallest-magnitude zeros (because deflation is less damaging if the small-magnitude zeros are done first.
5. When the coefficients are all real, immediately computing the conjugate zero after each non-real zero.
6. Probably some other tricks that I have long forgotten.

-- aloha, david