THE DERIVE - NEWSLETTER #117

ISSN 1990-7079

THE BULLETIN OF THE DERIVE USER GROUP

+ CAS-TI

Contents:

March 2020

Some Links and one recommended reading:

John Hanna is an expert in the use of TI-Nspire. You can find a rich collection of applications together with the respective tns-files. Enjoy it.

http://www.johnhanna.us/

A bundle of resources in English and German (Try Nordvik and Sousa …)

https://wiki.zum.de/wiki/TI-Nspire/freies_Material

Prof. Hans Humenberger at the University of Vienna offers a huge collection of papers (in German and in English) on his website (~ 130). Most of his articles have a didactical background.

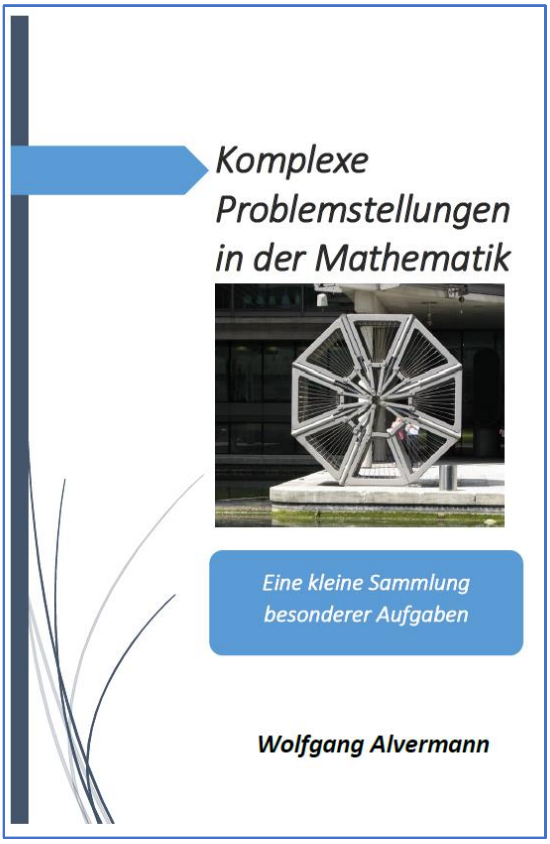
https://homepage.univie.ac.at/hans.humenberger/publikationen.html

Unser langjähriges DUG-Mitglied Wolfgang Alvermann, der uns schon viele schöne Beiträge geliefert hat, hat eine Sammlung von 28 Aufsätzen zusammengestellt und diese den Mitgliedern der DUG zur Verfügung gestellt.

Lieber Wolfgang, herzlichen Dank dafür und weiterhin frohes Schaffen
wünscht im Namen der DUG
Josef

Our long-time DUG-member Wolfgang Alvermann, who has contributed many great articles has collected 28 "*Complex Problems in Mathematics*" (A small collection of special problems).

The German version can be downloaded from our website. I intend to include translations of his problems by and by in future newsletter.



Many thanks to Wolfgang and we wish happy work for the future,
On behalf of the DUG community
Josef

Dear DUG Members,

This is a very short letter in difficult Corona-times. We wish you and your family to stay healthy. (Follow the rules and recommendations of your authorities!) Better times will come.

Let's look ahead to another DUG year (it will be year 30 of its existence, which not so bad?).

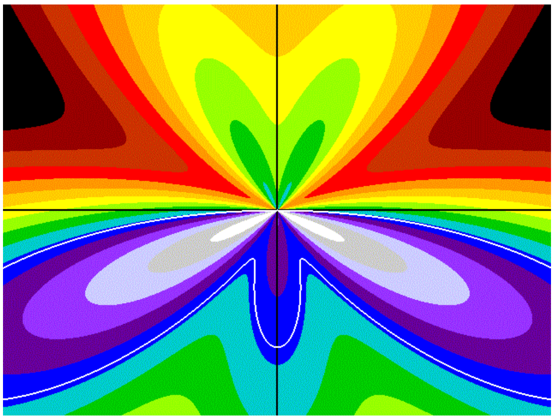
Best regards and wishes to all of you

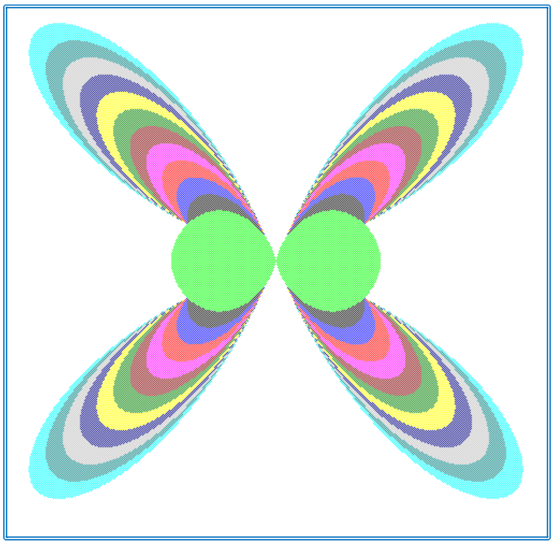
Noor and Josef

It is springtime and the first butterflies are around us:

http://old.nationalcurvebank.org/home/home.htm

http://old.nationalcurvebank.org////povray/povray.htm





Papilio derivia

More butterflies are fluttering on page 35.

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: June 2020

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
Simulating a Graphing Calculator in *DERIVE*, J. Böhm, AUT
Cubics, Quartics – Interesting features, T. Koller & J. Böhm, AUT
Logos of Companies as an Inspiration for Math Teaching
Exciting Surfaces in the FAZ
BooleanPlots.mth, P. Schofield, UK
Old traditional examples for a CAS – What´s new? J. Böhm, AUT
Mandelbrot and Newton with *DERIVE*, Roman Hašek, CZ
Tutorials for the NSpireCAS, G. Herweyers, BEL
Dirac Algebra, Clifford Algebra, Vector-Matrix-Extension, D. R. Lunsford, USA
A New Approach to Taylor Series, D. Oertel, GER
Statistics of Shuffling Cards, Charge in a Magnetic Field, H. Ludwig, GER
Selected Lectures from TIME 2016
More Applications of TI-Innovator™ Hub and TI-Innovator™ Rover
Surfaces and their Duals, Cayley Symmetroid, J. Böhm, AUT
Affine Mappings –Treated Systematically, H. Nieder, GER
Investigations of Lottery Game Outcomes, W Pröpper, GER
A Collection of Special Problems, W. Alvermann, GER
DERIVE Bugs?, D. Welz, GER
Tweening & Morphing with TI-NspireCX-II-T, J. Böhm. AUT
Why did the Tacoma-Narrows-Bridge Collapse? K-H. Keunecke, GER
The Gap between Poor and Rich, J. Böhm, AUT
Tumbling Tour in the Amusement Park, W. Alvermann, GER

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

Information from Prof. Simon Plouffe:

Hello Mr Böhm,
the English version is here:

http://plouffe.fr/NEW/a%20formula%20for%20primes.pdf

The calculation of p_n and $\pi(n)$

Simon Plouffe
Feb. 23 2020

Abstract

A new approach is presented for the calculation of p_n and $\pi(n)$ which uses the Lambert W function. An approximation is first found and using a calculation technique it makes it possible to have an estimate of these two quantities more precise than those known from Cipolla and Riemann. The calculation of p_n uses an approximation using the Lambert W function and an estimate based on a logarithmic least square curve (LLS) $c(n)$. The function $c(n)$ is the same in both cases. The two formulas are:

$$p_n \approx -nW_{-1}\left(\frac{-e}{n}\right) - \frac{n\,c(n)}{W_0(n)} \qquad 1$$

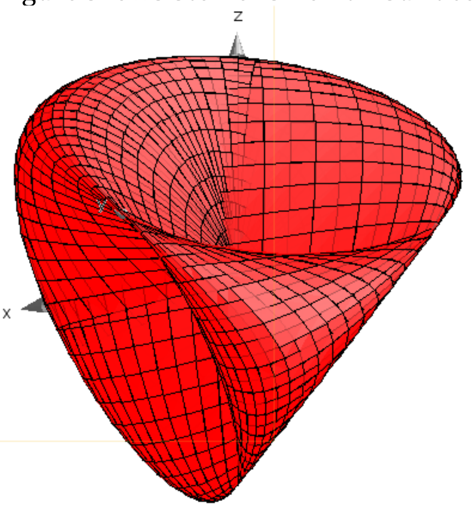
$$\pi(n) \approx \left\{-nW_{-1}\left(\frac{-e}{n}\right) - \frac{n\,c(n)}{W_0(n)}\right\}^{-1} \qquad 2$$

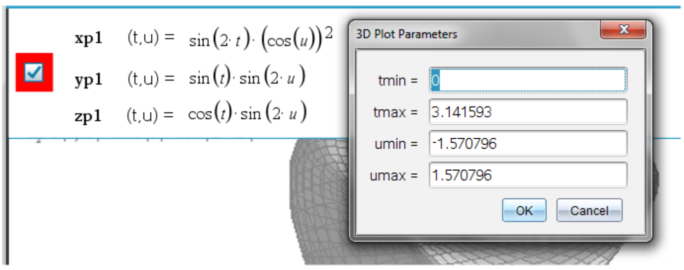
The results presented are empirical and apply up to $n \approx 10^{16}$.

If you prefer the French original version, then go to:

http://plouffe.fr/NEW/Une%20formule%20pour%20les%20nombres%20premiers%20II.pdf

The figure shows Steiner's Roman Surface. It belongs to page 17.





This is the TI-Nspire 3D plot

Hyperoperations with *DERIVE*

Julius Angres, Neumünster, Germany

1 Abstract

This paper deals with the order of arithmetic operations and the hyperoperations *tetration*, *pentation* etc. and their implementation in DERIVE. We present recursive implementations of basic arithmetic operations on the set of natural numbers and have a look at the relationship between the hierarchy of arithmetic operations, hyperoperations and the well-known ACKERMANN function[1].

2 Ordinary Arithmetic Operations

In this section we will only study arithmetic operations on the naturals. Hence, we sometimes leave out the attribute 'natural' and only use the term *number*. We put them into an order regarding their complexity. Starting with the successor function, binary addition, multiplication and exponentiation, we extend this hierarchy with the hyperoperations as defined in the up-arrow notation by Donald KNUTH[2].

2.1 Counting (Level 0)

The most basic operation in this context are functions that return the successor resp. predecessor of a number. Thus, we can move along the number line in steps of one. We define the corresponding functions *succ* and *pred* as follows.

$$succ : \mathbb{N} \to \mathbb{N}, succ(a) = a + 1 \tag{1}$$

$$pred : \mathbb{N} \setminus \{0\} \to \mathbb{N}, pred(a) = a - 1 \tag{2}$$

Defining these functions in DERIVE is straight forward. For the sake of readability, we will continue to use *succ*, but write $a - 1$ instead of *pred*.

2.2 Addition (Level 1)

Addition is repeated application of *succ*. Hence, we can define the addition of two naturals recursively. Examples:

$$3 + 4 = 3 + \underbrace{1 + 1 + 1 + 1}_{4\text{ times}} = 7 \tag{3}$$

$$1 + 2 = 1 + \underbrace{1 + 1}_{2\text{ times}} = 3 \tag{4}$$

$$a + b = a + \underbrace{1 + 1 + \cdots + 1}_{b\text{ times}} \tag{5}$$

Addition can be interpreted in a graphical way using the standard number line. The first operand indicates the starting point of the construction and the second one denotes the number of arrows of length one (applications of *succ*) that are required. The last arrow now points to the resulting number.

Using these examples, we can define recursive addition *addr* as follows.

$$addr : \mathbb{N} \times \mathbb{N} \to \mathbb{N}, addr(a,b) = \begin{cases} a & \text{:if } b = 0 \\ succ(addr(a, b-1)) & \text{:else} \end{cases} \tag{6}$$

---

[1] The ACKERMANN function was first defined by mathematician Wilhelm ACKERMANN (1896-1962) in 1928 in a proof concerning computability problems on primitive-recursive functions.

[2] Donald KNUTH (*1938) is an American computer scientist.

This almost directly translates into *DERIVE* code. The line above the definition contains the signature of the function in HASKELL style in all our code listings involving function definitions (comments).

```
#1:    succ :: Int -> Int

#2:    succ(a) := a + 1

#3:    addr :: Int -> Int -> Int

       addr(a, b) :=
         If b = 0
#4:         a
            succ(addr(a, b - 1))
```

A call of *addr*(3,2) for example would be evaluated like this:

addr(3,2)	=	*succ*(*addr*(3,2))	Case 2
	=	*succ*(*succ*(*addr*(3,0)))	Case 2
	=	*succ*(*succ*(3))	Case 1
	=	*succ*(3+1)	Def. *succ*
	=	(3 + 1) + 1	Def. *succ*
	=	5	

We see that the function evaluation consists of some recursive calls that expand the term before the terminating case of recursion is reached and the terms can be evaluated to concrete values. This behavior of expanding and collapsing is typical for recursive functions.

The experienced programmer knows that the evaluation of recursive functions can be speeded up by using a so-called accumulator that computes the result of a function while the function's body is expanded by the recursive calls. Thus no 'running back' is required. A recursive definition of addition using an accumulator can be implemented in *DERIVE* like this:

```
#5:    addrc :: Int -> Int -> Int -> Int

       addrc(a, b, n) :=
         If b = 0
#6:         n
            addrc(a, b - 1, succ(n))
```

In this function the accumulator n must have an initial value of a to produce correct results. An initial value of 0 for the accumulator would require the function to return $n + a$ and would thus rely on addition.

However, both recursive functions are closely related to the mathematical notation. By contrast to this functional approach, a conventional imperative version of addition using a loop can be realized in *DERIVE* like this:

```
#7:    addi :: Int -> Int -> Int

       addi(a, b) :=
         Loop
           If b = 0
#8:           RETURN a
           a := succ(a)
           b := b - 1
```

In the listing above we see that one big advantage of functional programming is the direct transformation from the mathematical notation into code.

2.3 Multiplication (Level 2)

In the same sense that addition is repeated application of the successor function, we can think of multiplication as repeated addition. Examples:

$$3 \cdot 4 = \underbrace{3+3+3+3}_{4\text{ times}} = 12 \tag{7}$$

$$1 \cdot 2 = \underbrace{1+1}_{2\text{ times}} = 2 \tag{8}$$

$$a \cdot b = \underbrace{a+a+\cdots+a}_{b\text{ times}} \tag{9}$$

Again, the first operand indicates what number to start with and the second one denotes the number of repetitions needed. Note that multiplication usually (aside from some cases involving very small numbers) produces bigger numbers than addition.
We can thus realize multiplication in a recursive way using the aforementioned *succ* function. We start with the mathematical definition:

$$multr : \mathbb{N} \times \mathbb{N} \to \mathbb{N}, multr(a,b) = \begin{cases} 0 & :\text{if } b = 0 \\ addr(a, multr(a, b-1)) & :\text{else} \end{cases} \tag{10}$$

And again, using functional style we can directly convert this definition into *DERIVE* code:

```
#9:    multr :: Int -> Int -> Int

       multr(a, b) :=
         If b = 0
#10:        0
            addr(a, multr(a, b - 1))
```

Please note, that any call of *multr* will not only call *addr* but also *succ* as we have defined addition through the successor function. This forces us to change our implementation a little when advancing to higher order operations to improve performance and avoid huge amounts of recursive calls that all have to be held in memory waiting for evaluation.

2.4 Exponentiation (Level 3)

The most complex operation we usually see in ordinary mathematics is exponentiation. Yet again, we can use exponentiation to abbreviate an expression of lower level operations. Exponentiation is in fact repeated multiplication as the following examples show.

$$3^4 = \underbrace{3 \cdot 3 \cdot 3 \cdot 3}_{4\text{ times}} = 81 \tag{11}$$

$$1^2 = \underbrace{1 \cdot 1}_{2\text{ times}} = 1 \tag{12}$$

$$a^b = \underbrace{a \cdot a \cdot \ldots \cdot a}_{b\text{ times}} \tag{13}$$

This time we will skip the mathematical function definition and directly proceed with the *DERIVE* implementation.

```
#11:  powr :: Int -> Int -> Int

      powr(a, b) :=
        If b = 0
#12:       1
           multr(a, powr(a, b - 1))
```

We can verify the correctness of the function by testing it with some self-chosen input.

```
powr(3, 4) = 81

powr(1, 2) = 1

powr(2, 0) = 1

powr(2, 10) = 1024
```

This is the output we get, if we enter our running examples and simplify the expression. Our hierarchy of recursively defined functions from ordinary mathematics behaves just as expected so far.

3 Beyond the Ordinary

3.1 Tetration (Level 4)

At this point we leave the common arithmetic operations behind and continue with the so-called hyperoperations, i.e. operations which are of even higher levels than exponentiation. In fact, exponentiation is defined as first level hyperoperation in *KNUTH*'s up-arrow notation. Having a careful look at the code snippets above, one might already spot the pattern that leads us further up the ladder: The first number is a starting point or *base* as it is called in exponential terms and the second one is the number of repetitions for the operation. The implementations also reflect this pattern as each operation makes a recursive call to the operation of the level directly below. That way we can define multiplication as repeated addition and exponentiation as repeated multiplication. Following this pattern, the next operation, the second hyperoperation must be repeated exponentiation. This operation is called *tetration*. On paper there exist several notations for titration and the big numbers it produces. We will stick with KNUTH's up-arrow notation. Using this notation, the tetration of two numbers a (the base) and b (the height of the power tower) can be written as $a\uparrow\uparrow b$ resp. $a\uparrow^2 b$.

$$1 \uparrow\uparrow 2 = 1^1 = 1 \tag{14}$$

$$2 \uparrow\uparrow 4 = 2^{2^{2^2}} = 2^{16} = 65536 \tag{15}$$

$$3 \uparrow\uparrow 4 = 3^{3^{3^3}} = 3^{7622597484987} \tag{16}$$

$$a \uparrow\uparrow b = \underbrace{a^{a^{\cdot^{\cdot^{a}}}}}_{b\text{ times}} \tag{17}$$

Now tetration can be defined in *DERIVE* as follows:

```
#13:  tetr :: Int -> Int -> Int

      tetr(a, b) :=
        If b = 0
#14:        1
            powr(a, tetr(a, b - 1))
```

As the examples show, the results of tetration rapidly become mindboggling. Playing around a little with some small values (numbers smaller than 3 in fact) already causes *DERIVE* to calculate several seconds before presenting the result. Using the implementation above this is also due to the fact that all our recursive functions call their predecessors from lower levels in the hierarchy meaning that a call of *tetr* will involve lots of calls of *succ* in the end. We overcome this shortcoming by using a modified implementation of tetration that uses *DERIVE*'s built-in exponentiation.

```
#15:  tetr2 :: Int -> Int -> Int

      tetr2(a, b) :=
        If b = 0
#16:        1
            a^tetr2(a, b - 1)
```

Using *tetr2* we can obtain some more results of tetration operations, at least on small integers. The reason why the numbers are getting so huge quickly is the definition of tetration as repeated exponentiation.

Each tetration defines a *power tower* that is evaluated from the top to the bottom (tetration is right-associative). Below is some example output of *DERIVE* calculating tetrations. Using `VECTOR` we can quickly find out where the borders of *DERIVE*'s ability are for any given base number. Let's have a look at the tetration of 2.

```
VECTOR(tetr2(2, k), k, 1, 5)

[2, 4, 16, 65536,

  20035299304068464649790723515602557504478254755697514192650169737
  64761547029165041871916351587966347219442930927982084309104855990
  50020667156370236612635974714480711177481588091413574272096719015
  09057075603140350761625624760318637931264847037437829549756137709
  41529463842244845292537361442533614373729088303794601274724958414
  90113423778270556742108007006528396332215507783121428855167555443
  20000241419637068135598404640394721940160695176901561197269823376
  61100640362119796101859534802787167200122604642492385111393400464
  97333357615955239488529757995402847194352991354376370598692891375
  77031380647813423095961909606545913008901888875880847336259560654
  91403236328496233046421066136200220175787851857409162050489711781
  37620399920349202390662626449190916798546151577883906039772075927
  36384083847782637904596071868767285097634712719888906804782432303
  18301158780701975535722241400019548102005661773589781499532325208
  86730653993164072049223847481528061916690093380573212081635070763
  10058924766554458408383347905441448176842553272073155863493476061
  38104076468878471647552945326947661700424461063311238021134588694
```

The results for $k \le 4$ are fine, but $2\uparrow\uparrow 5$ (only a few digits of which are shown in the screenshot) is an enormous number since it equals 2^{65536}. We can use the `DIM` command to count the number of digits.

```
DIM((VECTOR(tetr2(2, k), k, 1, 5)) ) = 19729
                                  5
```

Trying tetrations with base 3 also comes to a quick end.

$$\text{VECTOR(tetr2(3, k), k, 1, 4)} = \left[3,\ 27,\ 7625597484987,\ 3^{7625597484987}\right]$$

The number 3↑↑4 is already so large that *DERIVE* can only provide us with a symbolic result but is unable to count even the digits of it[3]. Recalling the definition of tetration, we see that 3↑↑4 equals 7625597484987 times 3 multiplied by itself.

3.2 Pentation (Level 5) and above

Of course, the pattern for the lower level operations can be repeated itself to define an infinite hierarchy of hyperoperations starting with the exponentiation as level one of this hierarchy. The next step after tetration, would be *pentation* which is defined as repeated tentration. All these hyperoperations can easily be implemented in *DERIVE* using the built-in exponentiation and the already defined hyperoperations. As the pattern is always the same and the number produced by pentation, hexation, heptation, etc. almost immediately become so large that they cannot even be computed (or even displayed) by supercomputers we will provide the implementation of pentation as our final example.

```
#17:  penr :: Int -> Int -> Int

      penr(a, b) :=
        If b = 0
#18:       1
           tetr2(a, penr(a, b - 1))
```

Note that we are using *tetr2* for the recursive call due to the aforementioned performance issues wit the generic implementation of *tetr*.

And this is what pentation of small numbers looks like:

```
penr(2, 2) = 4

penr(2, 3) = 65536

penr(3, 1) = 3

penr(3, 2) = 7625597484987
```

3.3 KNUTH's Up-arrow Notation

One possible mathematical way to define hyperoperations of arbitrary level is the up-arrow notation which only consists of three simple rules. The recursive definition for it is the following[4]:

$$a \uparrow^n b = \begin{cases} a^b & : \text{if } n = 1 \\ 1 & : \text{if } n \geq 1 \wedge b = 0 \\ a \uparrow^{n-1} (a \uparrow^n (b-1)) & : \text{else} \end{cases} \tag{18}$$

Here, $\uparrow^n$ stands for n arrows, so for example $2\uparrow^4 3 = 2\uparrow\uparrow\uparrow\uparrow 3$. This notation can be used to express some of the largest integer numbers that proved to be relevant in mathematics. One of the largest of them is the so-called GRAHAM's number G that originates from a proof about a problem in graph theory. To get an idea of its sheer size, consider the following function.

[3] See my comment on page 10 (bottom)

[4] Definition taken from https://en.wikipedia.org/wiki/Knuth%27s_up-arrow_notation

$$g_n = \begin{cases} 3 \uparrow^4 3 & : \text{if } n = 1 \\ 3 \uparrow^{g_{n-1}} 3 & : \text{else} \end{cases} \tag{19}$$

Now by definition GRAHAM's number $G = g_{64}$.

Ordinary and higher order operations can be combined in the hyperfunction *hyper* defined as follows.

$$hyper(a,n,b) = \begin{cases} a & : \text{if } n > 1 \wedge b = 1 \\ a+1 & : \text{if } n = 0 \\ a+b & : \text{if } n = 1 \\ a \cdot b & : \text{if } n = 2 \\ a^b & : \text{if } n = 3 \\ a^{hyper(a,4,b-1)} & : \text{if } n = 4 \\ hyper(a,n-1,hyper(a,n,b-1)) & : \text{if } n > 4 \end{cases} \tag{20}$$

It's easy to see that the parameter *n* corresponds to the level we have assigned to the operations in this paper. We can define the hyperoperations function in *DERIVE*.

```
#19:  hyper :: Int -> Int -> Int -> Int

      hyper(a, n, b) :=
        If n > 1 ∧ b = 1
           a
           If n = 0
              a + 1
              If n = 1
                 a + b
#20:             If n = 2
                    a·b
                    If n = 3
                       a^b
                       If n = 4
                          a^hyper(a, 4, b - 1)
                          hyper(a, n - 1, hyper(a, n, b - 1))
```

Note, that KNUTH's up-arrow notation is a part of the hyperoperation function. More precisely, it holds that $a\uparrow^n b = hyper(a,n+2,b)$. We can now use our hyperoperation function to test it with our running example where $a = 3$ and $b = 4$.

hyper(3, 0, 4) = 4

hyper(3, 1, 4) = 7

hyper(3, 2, 4) = 12

hyper(3, 3, 4) = 81

hyper(3, 4, 4) = $3^{7625597484987}$

The results prove that the function *hyper* is indeed a summary of all the functions presented.

Comment (JB): $\text{LOG}(3^{7625597484987}, 10) = 3.6383346400240996855 \cdot 10^{12}$ Gives the number of digits!

4 The ACKERMANN Function revisited

When investigating computable functions and fast-growing sequences one almost certainly comes across the ACKERMANN function sooner or later. It is an example of a computable non-primitive recursive function that is closely related to the hyperoperation discussed in the last chapter. The modified ACKERMANN function *a* (the original one had three parameters) has the following definition.

$$\begin{aligned} a(0,m) &= m+1 \\ a(n+1,0) &= a(n,1) \\ a(n+1,m+1) &= a(n,a(n+1,m)) \end{aligned}$$

Using a functional programming style in *DERIVE* we can easily implement the ACKERMANN function.

```
#21:   ackermann :: Int -> Int -> Int

       ackermann(n, m) :=
         If n = 0
            m + 1
#22:        If m = 0
               ackermann(n - 1, 1)
               ackermann(n - 1, ackermann(n, m - 1))
```

Playing around with some small input values gives us a small lookup table for n = 1,2,3 and m = 0,1,2,3,4,5.

```
APPEND([[m, a(1,m), a(2,m), a(3,m)]], TABLE([ackermann(1, k), ackermann(2, k),

  ackermann(3, k)], k, 0, 5))
```

$$\begin{bmatrix} \text{m} & \text{a(1,m)} & \text{a(2,m)} & \text{a(3,m)} \\ 0 & 2 & 3 & 5 \\ 1 & 3 & 5 & 13 \\ 2 & 4 & 7 & 29 \\ 3 & 5 & 9 & 61 \\ 4 & 6 & 11 & 125 \\ 5 & 7 & 13 & 253 \end{bmatrix}$$

However, the next column is impossible to calculate for the presented values of *m*. DERIVE easily calculates *ackermann*(4,0) = 13, but struggles to evaluate *ackermann*(4,1) = 65535 (really takes some time on my machine). The values of *ackermann*(4,*m*) with $m \geq 2$ cannot be computed be computed as the numbers grow incredibly large. In fact, columns of ACKERMANN function are related to the hierarchy of operations presented in the last chapter. The following table shows the relationship.

n	m	Operation
0	$m+1$	successor
1	$m+2$	addition
2	$2m+3$	multiplication
3	$8\cdot 2^m-3$	exponentiation
4	$2\uparrow\uparrow(m+3)-3$	tetration

We notice that each column produces results that represent a certain level in the hierarchy of arithmetic operations. In fact, the parameter *n* of the ACKERMANN function matches with the operation's level as presented in this paper. The values in the fifth column would therefore be related to pentation ($\uparrow^3$) and thus are almost impossible to compute except for $m = 0$, as *Ackermann*(5,0) = 65533.

5 Upshot

The hierarchy of standard arithmetic operations and hyperoperations proved to be a good use case for programming and exploring with *DERIVE*. If discussed with students the focus can be set on different levels. For beginners just provide them with some functions and let them explore the boundaries of computability by creating lookup tables etc. For more advanced classes the hyperoperations can be used to work on the topic of recursion and recursive definitions in an abstract way using Derive to make results visible.

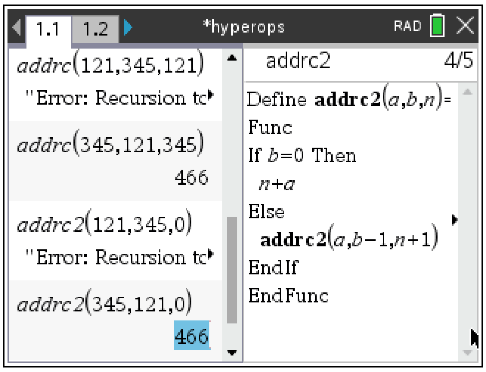
The big integers can also be used to initiate discussions about metamathematics, (ultra-) finitism, constructivism as well as philosophy.

Finally, the code snippets in this paper show that a functional programming style can be practiced with students using *DERIVE*. Of course, a fully-fledged purely functional language as HASKELL offers way more possibilities for professional programming, e.g. pattern matching, but in my opinion *DERIVE* can be used as an introductory tool to explain the concepts. This is especially beneficial if the students are used to working with *DERIVE* in their math lessons.

You know that I like to "translate" *DERIVE*-code into TI-Nspire-code and vice versa. So, we can ask ourselves how to realize Julius Angres' didactical concern and concept using TI-Nspire technology.

Recursion technique is available, so it is not surprising that this is not so difficult.

But I must admit that I came across one problem. In *DERIVE*-code function *addrc*(*a*,*b*,*n*) needs three arguments– *a*, *b*, *n* – with *n* being a local variable and we need only entering *a* and *b*. This is not possible in TI-Nspire-syntax. I found no other way to circumvent this problem as to enter *a* as third argument (playing the role of *n*). Using the command local n did not help. I wonder if there is another way to transfer the *DERIVE* code into TI-Nspire code?



I tried a second way. It works, but for "larger" numbers Recursion is too deep. (Same happens with addr(121,345), but reversing the summands gives the expected result.)

Find more about HASKELL on page 15.

Functions *addr* and *addi* for TI-Nspire:

addrc(*a,b,n*) needs to enter the value of *a* for parameter *n*.

addr(23,47)	70
addr(125,203)	328
addrc(47,23,47)	70
addrc(125,203,125)	328
addi(23,47)	70
addi(125,203)	328

addr 1/1

```
Define addr(a,b)=
when(b=0,a,addr(a+1,b−1))
```

addrc 1/5

```
Define addrc(a,b,n)=
Func
If b=0 Then
  n
Else
  addrc(a,b−1,n+1)
EndIf
EndFunc
```

addi 1/5

```
Define addi(a,b)=
Func
Loop
  If b=0:Return a
  a:=a+1
  b:=b−1
EndLoop
EndFunc
```

Functions *multr*, *powr* and *tetr2* for TI-Nspire:

multr(4,7)	28
powr(3,4)	81
powr(11,0)	1
powr(1,10)	1
tetr2(2,4)	65536
tetr2(3,3)	7625597484987
tetr2(3,4)	∞

"multr" stored successfully

```
Define multr(a,b)=
Func
when(b=0,0,addr(a,multr(a,b−1)))
EndFunc
```

"tetr2" stored successfully

```
Define tetr2(a,b)=
Func
when(b=0,1,a^tetr2(a,b−1))
EndFunc
```

"powr" stored successfully

```
Define powr(a,b)=
Func
when(b=0,1,multr(a,powr(a,b−1)))
EndFunc
```

Here are the remaining functions for TI-Nspire:

penr(2,3)	65536
penr(3,1)	3
penr(3,2)	7625597484987
hyper(3,4,0)	4
hyper(3,4,1)	7
hyper(3,4,2)	12
hyper(3,4,4)	∞
hyper(3,2,5)	7625597484987

```
penr                                   1/1
Define penr(a,b)=
Func
when(b=0,1,tetr2(a,penr(a,b-1)))
EndFunc
```

```
hyper                                  0/20
Define hyper(a,b,n)=
Func
If n>1 and b=1 Then
  a
  Else
    If n=0 Then
    a+1
    Else
      If n=1 Then
        a+b
      Else
        If n=2 Then
          a·b
        Else
          If n=3 Then
            a^b
          Else
            If n=4 Then
              a^hyper(a,b-1,4)
            Else
                hyper(a,hyper(a,b-1,n),n-1)
EndIf:EndIf:EndIf:EndIf:  EndIf:EndIf
```

Finally, the ACKERMANN function:

ackerm(0,10)	11
ackerm(2,2)	7
ackerm(3,2)	29
ackerm(3,5)	253
ackerm(4,3)	"Error: Recursion too deep"

```
ackerm
Define ackerm(n,m)=
Func
If n=0:
  Return m+1
If n>0 and m=0:
  Return ackerm(n-1,1)
  Return ackerm(n-1,ackerm(n,m-1))
EndFunc
```

When I asked Mr Angres to send the Haskell-code for his hyperoperations he was so friendly to fulfill my request within a few days. (Maybe that I will – having some leisure time?? – install Haskell on my PC. There is a Windows, a Mac and a Linux distribution for download.

If you like to inform about Haskell which is mentioned in Julian Angres' contribution then go to

https://www.haskell.org/

Haskell
An advanced, purely functional programming language

for information, examples, download, documentation and lots of resources.

See also https://wiki.haskell.org/Introduction#Why_use_Haskell.3F

This is the Haskell code provided by Juilus Angres:

```
-- Successor function is built-in as Prelude.succ

-- Addition
add :: Integer -> Integer -> Integer
add a 0 = a
add a b = succ $ add a (b-1)

-- Multiplication
mult :: Integer -> Integer -> Integer
mult a 0 = 0
mult a b = add a $ mult a (b-1)

-- Exponentiation
pow :: Integer -> Integer -> Integer
pow a 0 = 1
pow a b = mult a $ pow a (b-1)

-- Tetration (using previously defined arithmetic)
tetr :: Integer -> Integer -> Integer
tetr a 0 = 1
tetr a b = pow a $ tetr a (b-1)

-- Tetration (using built-in arithmetic)
tetr2 :: Integer -> Integer -> Integer
tetr2 a 0 = 1
tetr2 a b = (^) a $ tetr2 a (b-1)

-- Generalized hyperoperation
hyper :: Integer -> Integer -> Integer -> Integer
hyper a n 1 = a
hyper a 0 b = a + 1
hyper a 1 b = a + b
hyper a 2 b = a * b
hyper a 3 b = a ^ b
hyper a 4 b = a ^ hyper a 4 (b-1)
hyper a n b = hyper a (n-1) $ hyper a n (b-1)
```

https://en.wikipedia.org/wiki/Hyperoperation

https://waitbutwhy.com/2014/11/1000000-grahams-number.html

http://www.alaricstephen.com/main-featured/2016/11/4/knuths-up-arrow-notation-and-grahams-number

https://groups.google.com/forum/#!topic/tinspire/iTK2BulxtCQ

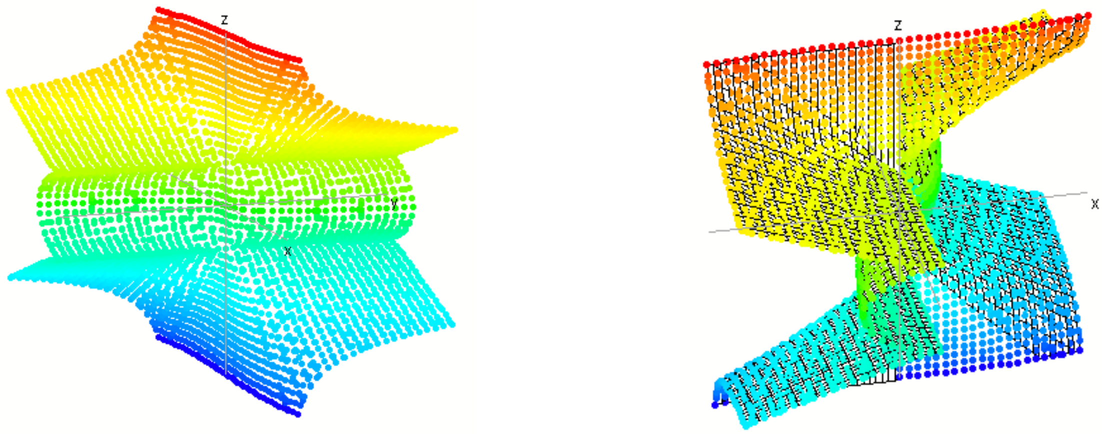
and once more:

https://www.haskell.org/

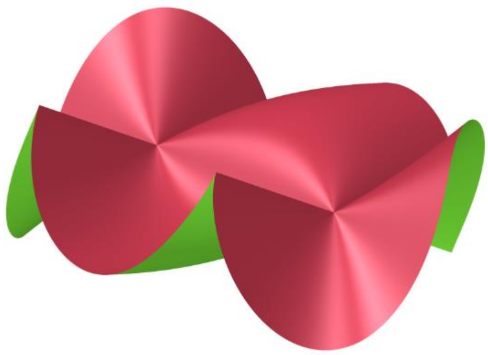
Surfaces from the Newspaper (8)

$$\text{ImplicitPts}\left(x^3 - y\cdot(1 - z^2)^2,\ -4,\ 4,\ 0.2\right)$$

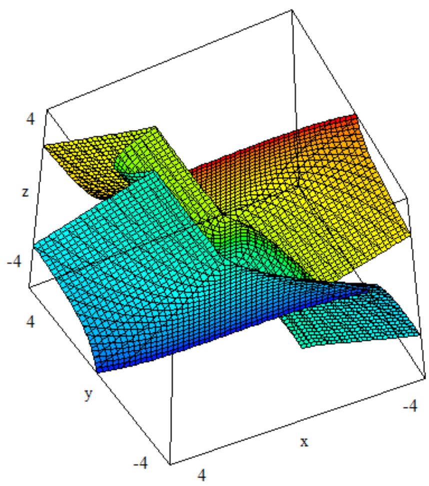
$$\left[\sqrt{1 - \sqrt{\frac{x^3}{y}}},\ -\sqrt{1 - \sqrt{\frac{x^3}{y}}},\ \sqrt{1 + \sqrt{\frac{x^3}{y}}},\ -\sqrt{1 + \sqrt{\frac{x^3}{y}}}\right]$$



DERIVE Plots (superimposed explicit form – in 4 parts)

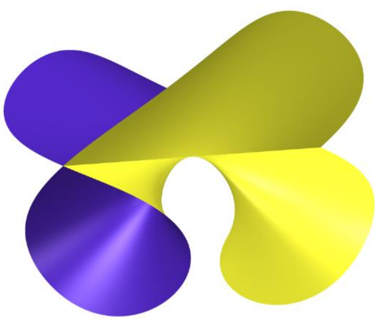


Surfer

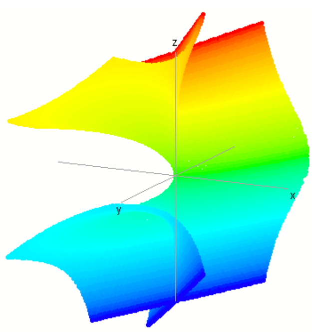


DP Graph

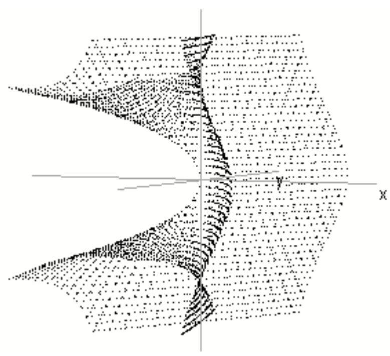
$$\text{ImplicitDots}\left(x^3 + x^2\cdot z^2 - y^2,\ -3,\ 3,\ 0.15\right)$$



Surfer



DP Graph

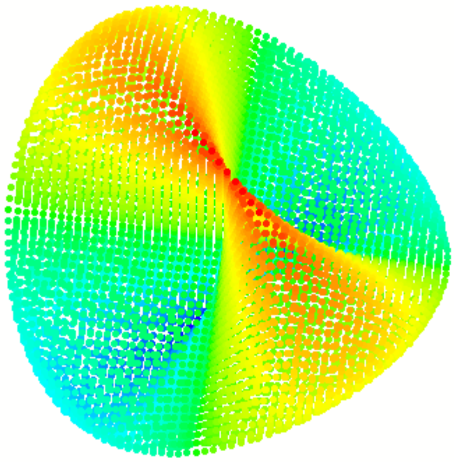


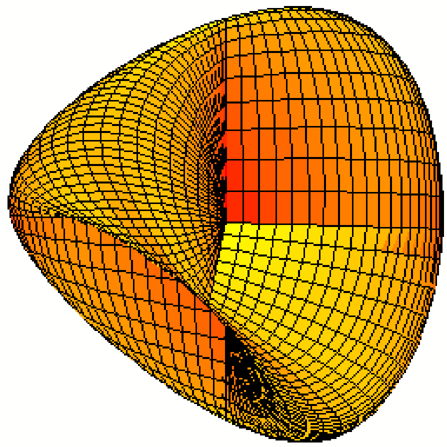
DERIVE

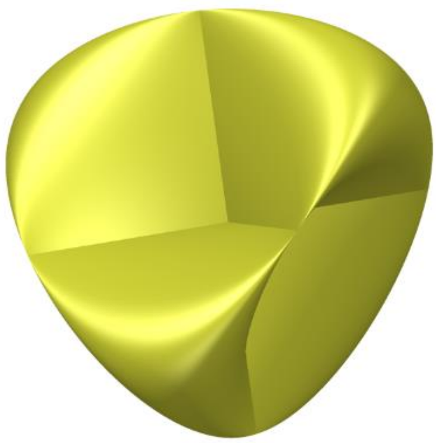
Steiner's Roman Surface. It was discovered by Jacob Steiner when he visited Rome in 1844.

$$\mathrm{ImplicitPts}(x^2\cdot y^2 + x^2\cdot z^2 + y^2\cdot z^2 - x\cdot y\cdot z,\ -1,\ 1,\ 0.02)$$

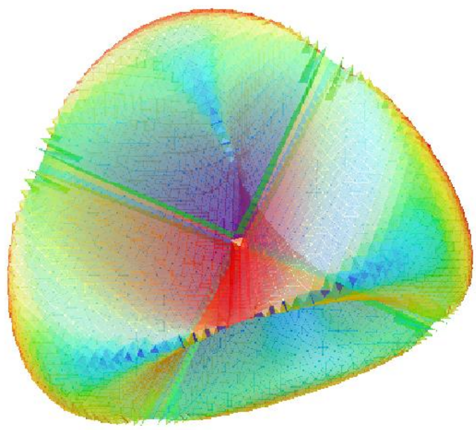
$$\left[\mathrm{SIN}(2\cdot s)\cdot \mathrm{COS}(t)^2,\ \mathrm{SIN}(s)\cdot \mathrm{SIN}(2\cdot t),\ \mathrm{COS}(s)\cdot \mathrm{SIN}(2\cdot t)\right]$$







Surfer (www.imaginary.org)



DPGraph (http://www.dpgraph.com/)

This is a remarkable surface – so you can find numerous websites:

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

https://en.wikipedia.org/wiki/Roman_surface

https://www.mathcurve.com/surfaces/romaine/romaine.shtml

https://mathworld.wolfram.com/RomanSurface.html

http://paulbourke.net/geometry/steiner/

https://www.geogebra.org/m/QRqzzDGN

http://old.nationalcurvebank.org/romansurfaces/romansurfaces.htm

Penney-Ante for TI-Nspire and DERIVE

Don Phillips and Josef Böhm

Don's mail came in on the last day of 2019, December 31. See what did follow:

Josef,

I don't know if you ever came across this probability problem before, but it might give your readers some food for thought. I will give you the strategy if you cannot figure it out.

Best regards,

Don

Penney Ante

I was a long−term substitute in an AP statistics class when a student told me about this game of chance. Two people each choose a sequence of three coin tosses, e.g., HTH or TTH, etc. They would then toss a coin until one of the sequences came up first. He then told me that if the second person chose a sequence after the first person chose one, there was a winning strategy for the second person. The odds of winning could be increased. I said no way! The chance of winning for each was 50%. The student persisted, so I started to do some research.

I found out the game was first presented by Walter Penney in the *Journal of Recreational Mathematics* in 1969 and that Martin Gardner described it in his *Mathematical Games* column in the October 1974 issue of Scientific American. So, the upshot is, there is a winning stategy! See if you can discover it.

penneyante$(\{1,2,1\},\{1,1,2\},100)$

$$\begin{bmatrix} "P1" & "P2" & "ODDS" & "Approx" \\ 28 & 72 & \frac{18}{7} & 2.6 \end{bmatrix}$$

Person 2 has the odds of 18/7 to win! That is, out of 25 games, he will win about 18.

If you do more repititions the odds for winning approaches 2 to 1.

penneyante({1,2,1},{1,1,2},10000)

$$\begin{bmatrix} \text{"P1"} & \text{"P2"} & \text{"ODDS"} & \text{"Approx"} \\ 3350 & 6650 & \frac{133}{67} & 2. \end{bmatrix}$$

And, how about this one!

penneyante({1,1,1},{2,1,1},10000)

$$\begin{bmatrix} \text{"P1"} & \text{"P2"} & \text{"ODDS"} & \text{"Approx"} \\ 1274 & 8726 & \frac{4363}{637} & 6.8 \end{bmatrix}$$

This is the simulation program.

© Here is the program I wrote to simulate the coin tosses. If 1 is heads and 2 is tails, I start with a randInt list of 3 elements of 1's and 2's. If there is no match, I keep the right 2 elements of the list and add one more element. This goes on until there is a match and then another iteration is started.

© What is the winning strategy? I leave it for you, gentle reader, to figure out.

penneyante 17/17

```
Define LibPub penneyante(t1,t2,r)=
Func
Local n,l,a,b,m,o,o1
a:=0: b:=0
For n,1,r
  l:=randInt(1,2,3):   m:=0
  While m=0
  If string(l)=string(t1) Then
    a:=a+1:   m:=1
  ElseIf string(l)=string(t2) Then
    b:=b+1:   m:=1
  Else
    l:=augment(right(l,2),{randInt(1,2)})
  EndIf
  EndWhile
EndFor
o:=b/a
o1:=round(approx(o),1)
["P1" "P2" "ODDS" "Approx"
  a    b     o      o1   ]
EndFunc
```

Can you figure out the winning strategy?

Dear Don,

many thanks for this great example for probability theory.

Instead of figuring it out, I – shame on me – “googled” for *Penney Ante* and found a lot of respective websites (strategy included!!).

http://www.math.unl.edu/~sdunbar1/ProbabilityTheory/BackgroundPapers/Penney%20ante/PenneyAnte_CounterintuitiveProbabilities.pdf

https://en.wikipedia.org/wiki/Penney%27s_game

https://penneyante.weebly.com/uploads/5/9/3/5/59353369/penney_ante_problem_for_website.pdf

Maybe that I will try translating your Nspire program to a DERIVE function?
Thanks again and
best regards and wishes
Josef

Dear Don,
I like your Prob problem very much.
Just to demonstrate this – and just for fun – I made a little change:
Instead of entering {1,1,1} and {2,1,1} I enter “HHH” and “THH”.
This is function penney2.
Regards as ever
Josef

penney2 allows to enter the given sequences as strings:

penney2("HTH","HHT",10000) ▸ $\begin{bmatrix} \text{"HTH"} & \text{"HHT"} & \text{"ODDS"} & \text{"Approx"} \\ 3329 & 6671 & \frac{6671}{3329} & 2. \end{bmatrix}$

penney2("HHH","THH",10000) ▸ $\begin{bmatrix} \text{"HHH"} & \text{"THH"} & \text{"ODDS"} & \text{"Approx"} \\ 1225 & 8775 & \frac{351}{49} & 7.2 \end{bmatrix}$

penney2("THH","HHH",10000) ▸ $\begin{bmatrix} \text{"THH"} & \text{"HHH"} & \text{"ODDS"} & \text{"Approx"} \\ 8752 & 1248 & \frac{78}{547} & 0.1 \end{bmatrix}$

penney2("HHH","TTT",10000) ▸ $\begin{bmatrix} \text{"HHH"} & \text{"TTT"} & \text{"ODDS"} & \text{"Approx"} \\ 4974 & 5026 & \frac{2513}{2487} & 1. \end{bmatrix}$

Josef,

I like your improvement! You should use that one if you decide to publish it in the newsletter. And, it's given me another thought. I'm going to add some code so a person can play against the program. I'll let you know when I'm successful.

Well, I hope you're not in a deep freeze like we are on this side of the pond!

I've added penney3 which chooses the correct strategy. As long as you don't look at the code, you have a chance to discover the strategy yourself.

Regards,
Don

$$\begin{bmatrix} 3311 & 6689 & \frac{6689}{3311} & 2. \end{bmatrix}$$

penney3("HTH",10000) $\begin{bmatrix} \text{"HTH"} & \text{"HHT"} & \text{"ODDS"} & \text{"Approx"} \\ 3297 & 6703 & \frac{6703}{3297} & 2. \end{bmatrix}$

penney3("HHH",10000) $\begin{bmatrix} \text{"HHH"} & \text{"THH"} & \text{"ODDS"} & \text{"Approx"} \\ 1189 & 8811 & \frac{8811}{1189} & 7.4 \end{bmatrix}$

penney3("HHT",10000) $\begin{bmatrix} \text{"HHT"} & \text{"THH"} & \text{"ODDS"} & \text{"Approx"} \\ 2568 & 7432 & \frac{929}{321} & 2.9 \end{bmatrix}$

Hi Don,
I added one more function penney4(t1,t2). It demonstrates one single game between two players and shows the series of tosses together with the winner.
Regards
Josef

"Player 2 wins"

penney4("HTH","HHT") $\begin{bmatrix} \text{"TTTHHHT"} \\ \text{"Player 2 wins"} \end{bmatrix}$

penney4("HTH","HHT") $\begin{bmatrix} \text{"HTH"} \\ \text{"Player 1 wins"} \end{bmatrix}$

penney4("HTH","HHT") $\begin{bmatrix} \text{"TTTTTTTHTTHHT"} \\ \text{"Player 2 wins"} \end{bmatrix}$

penney4("HTH","HHT") $\begin{bmatrix} \text{"HTH"} \\ \text{"Player 1 wins"} \end{bmatrix}$

penney4("HTH","HHT") $\begin{bmatrix} \text{"THHHT"} \\ \text{"Player 2 wins"} \end{bmatrix}$

Dear Don,

sorry, to bother you once more:

I changed penney4(t1,t2) in such a way that you can enter toss-sequences of variable length e.g. penney4("HHHHH","TTTTT").

Regards

Josef

```
penney4("HHHHH","TTTTT")
                HHTTT
                HHTTTH
                HHTTTHT
                HHTTTHTT
                HHTTTHTTH
                HHTTTHTTHH
                HHTTTHTTHHH
                HHTTTHTTHHHH
                HHTTTHTTHHHHH
        ["HHTTTHTTHHHHH"  "Player 1 wins"]
```

```
penney4                                    11/22
Define LibPub penney4(t1,t2)=
Func
Local c,l1,l,lc,m,k,i
k:=dim(t1)
c:={"H","T"}
l:=""
For i,1,k
  l:=l&c[randInt(1,2,1)[1]]
EndFor
lc:=l
Lbl next
© next line can be removed
Disp lc
If l=t1 Then
  m:=1:Goto end
```

Josef,

I have changed my original program to handle any number of coin sequences, using some of your code. Thanks for all your help! By the way, the same strategy seems to work for any number of coins.

Don

© Two examples of more coin sequences.

penneyante("TTHHT","THHHT",5000)

$$\begin{bmatrix} \text{"TTHHT"} & \text{"THHHT"} & \text{"ODDS"} & \text{"Approx"} \\ 2562 & 2438 & \frac{1219}{1281} & 1. \end{bmatrix}$$

penneyante("HHHTTT","TTTHHH",5000)

$$\begin{bmatrix} \text{"HHHTTT"} & \text{"TTTHHH"} & \text{"ODDS"} & \text{"Approx"} \\ 2454 & 2546 & \frac{1273}{1227} & 1. \end{bmatrix}$$

penneyante("HHHTTT","THHHTT",5000)

$$\begin{bmatrix} \text{"HHHTTT"} & \text{"THHHTT"} & \text{"ODDS"} & \text{"Approx"} \\ 1807 & 3193 & \frac{3193}{1807} & 1.8 \end{bmatrix}$$

© Two examples of more coin sequences.

penneyante("TTHHT","THHHT",5000)

$$\begin{bmatrix} \text{"TTHHT"} & \text{"THHHT"} & \text{"ODDS"} & \text{"Approx"} \\ 2562 & 2438 & \frac{1219}{1281} & 1. \end{bmatrix}$$

penneyante("HHHTTT","TTTHHH",5000)

$$\begin{bmatrix} \text{"HHHTTT"} & \text{"TTTHHH"} & \text{"ODDS"} & \text{"Approx"} \\ 2454 & 2546 & \frac{1273}{1227} & 1. \end{bmatrix}$$

penneyante("HHHTTT","THHHTT",5000)

$$\begin{bmatrix} \text{"HHHTTT"} & \text{"THHHTT"} & \text{"ODDS"} & \text{"Approx"} \\ 1807 & 3193 & \frac{3193}{1807} & 1.8 \end{bmatrix}$$

Hi Don,

many thanks. I had the intention to suggest this generalization.

I am sure that this will make a fine contribution for the next DNL (including your penney3 and possibly my penney4).

I liked this collaboration very much.

Best regards

Josef

As Don is writing: "*the same strategy seems to work for any number of coins*" I am not sure if this is true. I found only one paper dealing with *n*-tuples of tosses (see the first link among the URLs given below). **Latest news: Please follow my DERVE implementation!**

I wanted to realize the Penney Ante Game with DERIVE, too – and I added two more functions (programs) to indicate the difference between using the winning strategy and using it not. You can follow my functions on the next page. I don't print the programs in order to save space. All functions are contained in `penney_ante.dfw`.

Links:

http://www.math.unl.edu/~sdunbar1/ProbabilityTheory/BackgroundPapers/Penney%20ante/PenneyAnte_CounterintuitiveProbabilities.pdf

https://penneyante.weebly.com/uploads/5/9/3/5/59353369/penney_ante_problem_for_website.pdf

https://plus.maths.org/content/os/issue55/features/nishiyama/index

http://mlg.eng.cam.ac.uk/adrian/Penney.pdf

https://digitalcommons.newhaven.edu/cgi/viewcontent.cgi?referer=https://www.google.com/&httpsredir=1&article=1004&context=chemicalengineering-facpubs

These are my DERIVE functions. They are a little bit different from the Nspire programs from above:

Player 1 enters a sequence of his choice with arbitrary length. The computer- it is Player 2 - does not apply any strategy but behaves like an uninformed opponent and answers with a random sequence of heads and tails.

The output shows the second player's sequence, followed by the history of the tosses.

It can happen that the computer chooses the same sequence as Player 1. Then we will have no winner – and no looser, of course.

```
#2:   no_penney(HHHH)
PLayer 2:TTTH
TTHH
TTHHH
TTHHHH
       [    TTHHHH    ]
#3:    [              ]
       [ Player 1 wins ]
```

```
#4:   no_penney(HHHHT)
PLayer 2:TTTHH
THTTT
THTTTH
THTTTHH
       [    THTTTHH    ]
#5:    [               ]
       [ Player 2 wins ]
#6:   no_penney(HHH)
PLayer 2:HHH
HHT
HHTT
HHTTH
HHTTHH
HHTTHHH
       [  HHTTHHH  ]
#7:    [           ]
       [ no winner ]
```

Same as above, but we simulate n games between Player 1 and Player 2 (again the computer) having no special strategy.

```
no_penney_n(HTH, 10000)
[ HTH    THH    Odds   ]
[ 4978   5022   1.0088 ]
no_penney_n(HHH, 20000)
[ HHH        THT         Odds   ]
[                  4            ]
[ 8347   1.1653·10      1.3960  ]
no_penney_n(HTHTHT, 30000)
[ HTHTHT          HTTTTH         Odds   ]
[           4               4           ]
[ 1.3417·10       1.6583·10      1.2359 ]
```

I (Player 1) choose HTHTHT. My opponent (computer = Player 2) bets on HTTTTH (randomly chosen). The (simulated) odds for Player 2 are 1.24 : 1.

Function `penney(t1,t2)` allows playing against a real opponent: Player1 enters his sequence of tosses (as t1) and then Player 2 can enter his sequence as second argument t2.

Now you can follow what happens and finally you get the winner.

My last function `penney_n(t1,n)` lets you play *n* times against an opponent who knows the right strategy (the computer).

```
penney_n(HTH, 10000)
```

$$\begin{bmatrix} \text{HTH} & \text{HHT} & \text{Odds} \\ 3377 & 6623 & 1.9612 \end{bmatrix}$$

```
penney_n(HHH, 20000)
```

$$\begin{bmatrix} \text{HHH} & \text{THH} & \text{Odds} \\ 2497 & 1.7503\cdot 10^{4} & 7.0096 \end{bmatrix}$$

```
penney_n(HTHTHT, 10000)
```

$$\begin{bmatrix} \text{HTHTHT} & \text{HHTHTH} & \text{Odds} \\ 2237 & 7763 & 3.4702 \end{bmatrix}$$

Did you find out the winning strategy for Player2?

```
#22:  penney(HHH, TTT)
THT
THTH
THTHT
THTHTH
THTHTHH
THTHTHHT
THTHTHHTT
THTHTHHTTH
THTHTHHTTHT
THTHTHHTTHTT
THTHTHHTTHTTH
THTHTHHTTHTTHH
THTHTHHTTHTTHHH
```

$$\#23: \begin{bmatrix} \text{THTHTHHTTHTTHHH} \\ \text{Player 1 wins} \end{bmatrix}$$

```
#24:  penney(HHHT, TTTH)
TTTT
TTTTT
TTTTTH
```

$$\#25: \begin{bmatrix} \text{TTTTTH} \\ \text{Player 2 wins} \end{bmatrix}$$

You can find the proof following the links to some websites which are given above. All proofs are concerning the case of choosing three tosses. I repeat the trick: Player 2 should take the second toss of player 1, reverse it (T ↔ H or H ↔ T) and set this one in front of Player 1's sequence and delete the last one. H T H → H H T.

I am not sure if this recipe will hold for longer sequences. Simulation indicates that it is improving the odds for Player 2 significantly, but I don't know if there is a better strategy depending on the length of the sequence.

Would be great to receive any answers or comments.

Thanks to Don for providing this game with a surprising result and the wonderful communication.

Josef **(See the following appendix!)**

More simulations – and an important resource:

It is a nice coincidence that I – just after having finished the article above for this newsletter – found among my so (too) many books and papers the proceedings of a *Lehrerfortbildungstag* (teacher trainings day) held in 1998 (!!!) at the Vienna University. Hans Humenberger – *he is now full professor for Mathematics with Special Consideration to the Didactics of Mathematics* – gave a talk: *Ein Paradoxon bei Münzwurfserien und bedingte Erwartungswerte* (A paradox at sequences of coin tosses and conditional expected values).

http://www.oemg.ac.at/DK/Didaktikhefte/1998%20Band%2029/Humenberger1998.pdf

Here Humenberger demonstrates how to calculate the probabilities for the appearance of the various combinations of Heads and Tails not only for three coins but also for two and for more than three.

Similar like in the paper given in the first URL on page 20 he treats the expected values for the waiting times (number of tosses) until the requested pattern will appear.

Sequence	HHH	HHT	HTH	THH	HTT	THT	TTH	TTT
Expectation	14	8	10	8	8	10	8	14

I will not demonstrate how to find the values (applying conditional expectations) but will refer to a computer simulation:

```
       wait(t, m, c := "HT", nc, k, n, l, lc, dummy) :=
         Prog
           dummy := RANDOM(0)
           k := DIM(t)
           m := k
           l := CODES_TO_NAME(VECTOR(NAME_TO_CODES(c↓(RANDOM(2) + 1)), i, k)'↓1)
           lc := l
           DISPLAY(lc)
#35:       Loop
             If l = t
                RETURN DIM(lc)
             nc := c↓(RANDOM(2) + 1)
             lc := APPEND(lc, nc)
             DISPLAY(lc)
             l := APPEND(DELETE(l, 1), nc)
             m :+ 1
```

```
#36:   wait(TTT)

HHT

HHTH

HHTHT

HHTHTH

HHTHTHT

HHTHTHTT

HHTHTHTTT

#37:   9
```

Nine tosses to have TTT the first time

If you remove the two `DISPLAY` commands you will receive only the number of tosses:

```
wait(THTHT) = 41

wait(THTHT) = 48

wait(THTHT) = 102

wait(THTHT) = 6
```

Now I'd like to investigate the expected values performing *n* random experiments:

```
wait_n(TTT, 100)

12.43

wait_n(TTT, 1000)

13.982

wait_n(TTT, 1000)

14.13

[wait_n(HHH, 5000), wait_n(HHT, 5000), wait_n(HTH, 5000), wait_n(THH, 5000)]

[13.795, 7.9328, 9.9148, 8.0532]
```

We can have all expected values in one list applying my favorite command: VECTOR.

```
#53:  combs := [HHH, HHT, HTH, THH, HTT, THT, TTH, TTT]

#54:  VECTOR(wait_n(t, 5000), t, combs)

#55:  [13.838, 8.0348, 9.887, 7.9842, 7.9468, 9.849, 7.9904, 13.922]
```

Please compare with the table from above!

I want to simulate a large number of games with the players each of them keeping their first choice.

I start with Player 1 takes HHH and Player 2 answers with HHT – equal chance on the long hand.

$$\text{comp\_n(HHH, HHT, 10000)} = \begin{bmatrix} \text{HHH} & \text{HHT} & \text{WinProb for Player 2} & \text{Odds for Player 2} \\ 4993 & 5007 & 0.5007 & 1.0028 \end{bmatrix}$$

$$\text{comp\_n(HHH, HTH, 10000)} = \begin{bmatrix} \text{HHH} & \text{HTH} & \text{WinProb for Player 2} & \text{Odds for Player 2} \\ 4037 & 5963 & 0.5963 & 1.477 \end{bmatrix}$$

$$\text{comp\_n(HHH, THH, 10000)} = \begin{bmatrix} \text{HHH} & \text{THH} & \text{WinProb for Player 2} & \text{Odds for Player 2} \\ 1251 & 8749 & 0.8749 & 6.9936 \end{bmatrix}$$

$$\text{comp\_n(HHH, HTT, 10000)} = \begin{bmatrix} \text{HHH} & \text{HTT} & \text{WinProb for Player 2} & \text{Odds for Player 2} \\ 3912 & 6088 & 0.6088 & 1.5562 \end{bmatrix}$$

$$\text{comp\_n(HHH, THT, 10000)} = \begin{bmatrix} \text{HHH} & \text{THT} & \text{WinProb for Player 2} & \text{Odds for Player 2} \\ 4155 & 5845 & 0.5845 & 1.4067 \end{bmatrix}$$

$$\text{comp\_n(HHH, TTH, 10000)} = \begin{bmatrix} \text{HHH} & \text{TTH} & \text{WinProb for Player 2} & \text{Odds for Player 2} \\ 3027 & 6973 & 0.6973 & 2.3036 \end{bmatrix}$$

$$\text{comp\_n(HHH, TTT, 10000)} = \begin{bmatrix} \text{HHH} & \text{TTT} & \text{WinProb for Player 2} & \text{Odds for Player 2} \\ 4999 & 5001 & 0.5001 & 1.0004 \end{bmatrix}$$

The best answer of Player 2 is THH – which is applying the right strategy!!

Humenberger provides a table showing all probabilities for Player 1 winning against Player 2 for all possible combinations. (I reordered the first row of table to show the prob of Player 2 to win.)

	Player 2							
Player 1	HHH	HHT	HTH	THH	HTT	THT	TTH	TTT
HHH	-	1/2	3/5	7/8	3/5	7/12	7/10	1/2
Odds		1	3/2	7	3/2	7/5	7/3	1

The relationship between probability p and odds o is easy: $o = \frac{p}{1-p}$. You are invited to compare the exact values (given by Humenberger) with the results of the simulation of 10000 games.

Now it is no problem to proceed with longer patterns. I can come back to my question on page 23. ("*the same strategy seems to work for any number of coins*" I am not sure if this is true.)

One paper among the references gives the answer YES, it is true- but look at the third and the fourth result given below. Player 1 chooses HHTH. According the Penney Ante strategy Player 2 answers with THHT and he will have a prob to win of 7/12 (simulated 0.5886) but if he tries TTHH then the probability increases to 9/14 (simulated 0.6442). So, the strategy is good, but in this case, it is not the best one.

$$\text{comp\_n(HHHH, THHH, 10000)} = \begin{bmatrix} \text{HHHH} & \text{THHH} & \text{WinProb for Player 2} & \text{Odds for Player 2} \\ 618 & 9382 & 0.9382 & 15.181 \end{bmatrix}$$

$$\text{comp\_n(HHHH, THHH, 10000)} = \begin{bmatrix} \text{HHHH} & \text{THHH} & \text{WinProb for Player 2} & \text{Odds for Player 2} \\ 628 & 9372 & 0.9372 & 14.923 \end{bmatrix}$$

$$\text{comp\_n(HHTH, THHT, 10000)} = \begin{bmatrix} \text{HHTH} & \text{THHT} & \text{WinProb for Player 2} & \text{Odds for Player 2} \\ 4114 & 5886 & 0.5886 & 1.4307 \end{bmatrix}$$

$$\text{comp\_n(HHTH, TTHH, 10000)} = \begin{bmatrix} \text{HHTH} & \text{TTHH} & \text{WinProb for Player 2} & \text{Odds for Player 2} \\ 3558 & 6442 & 0.6442 & 1.8105 \end{bmatrix}$$

$$\text{comp\_n(HHTH, TTHH, 10000)} = \begin{bmatrix} \text{HHTH} & \text{TTHH} & \text{WinProb for Player 2} & \text{Odds for Player 2} \\ 3555 & 6445 & 0.6445 & 1.8129 \end{bmatrix}$$

$$\text{comp\_n(TTHH, HHTH, 10000)} = \begin{bmatrix} \text{TTHH} & \text{HHTH} & \text{WinProb for Player 2} & \text{Odds for Player 2} \\ 6393 & 3607 & 0.3607 & 0.56421 \end{bmatrix}$$

Humenberger's paper closes with a table for all possible games between 4-tosses-games and a very rich list of references (59). It is interesting that he does not mention the name "Penney Ante" in his article although his paper – cited by Don Phillips in his first email – can be found among the references as #44.

The next – and last – page dealing with Penney Ante is the realization of my additional functions with TI-Nspire.

```
HHTTTTTHHHTTTHHT
HHTTTTTHHHTTTHHTH
                15
waitt("HTH")
HTT
HTTT
HTTTH
HTTTHH
HTTTHHT
HTTTHHTT
HTTTHHTTT
HTTTHHTTTH
HTTTHHTTTHH
HTTTHHTTTHHT
HTTTHHTTTHHTH
                13
waitt("TTH")
HHH
```

```
waitt                                13/15
Define LibPub waitt(t)=
Func
Local k,c,i,l,lc,nc
k:=dim(t):c:={"H","T"}
l:=" "
For i,1,k
  l:=l&c[randInt(1,2,1)[1]]
EndFor
lc:=l
Disp lc
While l≠t
  nc:=c[randInt(1,2,1)[1]]
  lc:=lc&nc
Disp lc
  l:=right(l,k−1)&nc
EndWhile
dim(lc)
EndFunc
```

Function name "wait" is not permitted, because "wait" is an implemented Nspire-function.

	A n	B	C	D	E	F	G	H	I	J
=										
1	10000	HHH	HHT	HTH	THH	HTT	THT	TTH	TTT	
2	_	14.06	7.9453	9.9842	7.9926	8.0211	10.0445	7.9338	13.9434	
3	1000	HHHH	HHHT	HHTH	HTHH	THHH	HHTT	HTHT	HTTH	
4		30.609	16.45	18.155	17.406	15.575	15.842	20.213	17.831	

B4 =waitt_n(b3,a3)

```
This is penney from DERIVE. (Two players enter their sequence.)
penney_game("TTTH","HHTT") ▸ ["HHHTHTTHHTT"
                              "Player 2 wins"]
waitt("TTT") ▸ 14
waitt("THTHT") ▸ 49
waitt_n("TTT",10000) ▸ 13.8632
waitt_n("THTHT",5000) ▸ 41.8796

comp_n("HHTH","TTHH",5000) ▸ ["HHTH"  "TTHH"  "WinProb for Pl 2"  "Odds for Pl 2"
                               1764    3236    0.6472              1.83447]
```

I cannot use VECTOR to calculate the simulations of the expected values of waiting times, but I create the table in the Lists & Spreadsheet Application. Just enter the numbers of simulations in cells A1 and A3. All functions are contained in penney_ante.tns and penney_ante.dfw.

Direction Fields, Phase Planes and Nullclines

Josef Böhm, Würmla

I am member of a small group working through and discussing Steven Strogatz's book *Nonlinear Dynamics and Chaos*. There are many exercises and we try to solve some selected ones. David, one of the group members sent a mail asking for support:

… and (2) I recall that a few (or many) years there was a discussion of using DERIVE to plot "phase planes" for ODEs!"

Can you transmit it to me (via an email or text attachment)? I want to use it for plotting some of Strogatz's problems. I assume the one program has specific domains which can be poorly defined. I would appreciate it if you would look around and see what you have and what may be easy to use.

Let me start with an example from a textbook (because then I can check if I am right or not):
(Nice coincidence: from *Differential Equations, C.H. Edwards & David E. Penney*, Penney-Ante is from *Walter Penney*!)

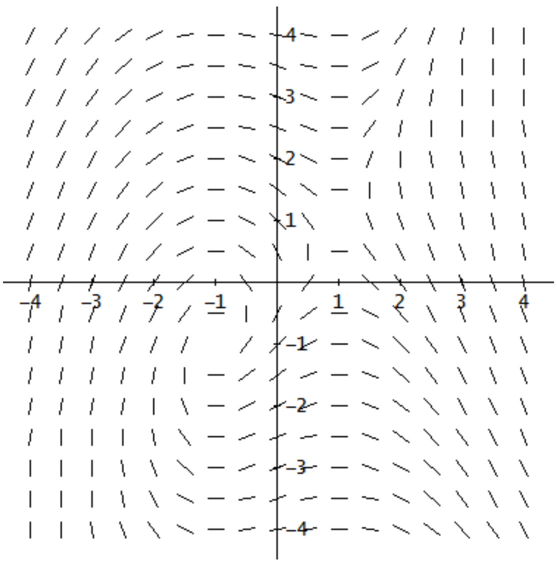
Given is the system $\begin{array}{l} x' = x - y \\ y' = 1 - x^2 \end{array}$.

We plot the direction field using DERIVE's built in function `DIRECTION_FIELD`:

#1: $\text{DIRECTION\_FIELD}\left(\dfrac{1 - x^2}{x - y}, x, -4, 4, 16, y, -4, 4, 16\right)$

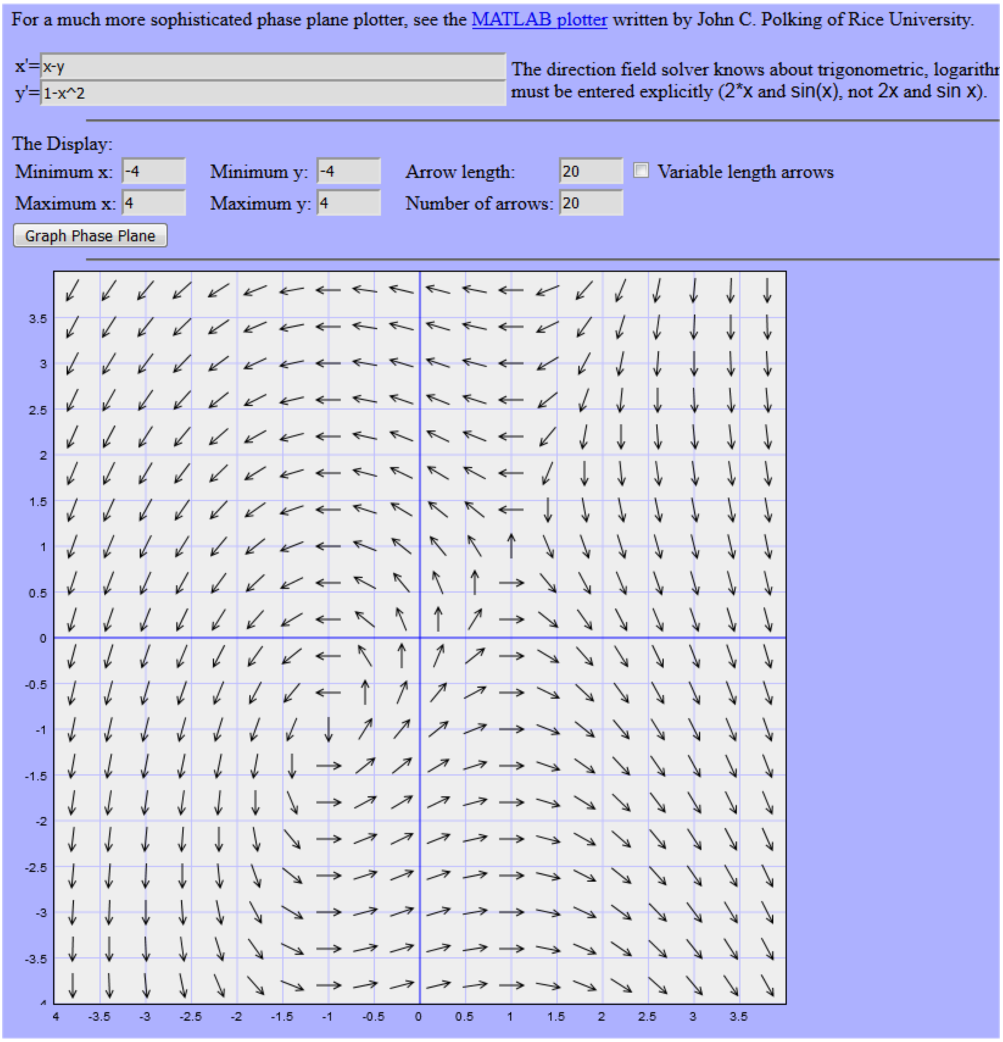
(the slope is $\dfrac{dy}{dx} = \dfrac{y'}{x'}$)

Don't evaluate expression #1, but open the 2D-plot window (set Options > Display > Points > Small and Connected and Options > Approximate before Plotting, and plot. You should find the following graph:



The website given below allows to plot the direction field interactively:

https://aeb019.hosted.uark.edu/pplane.html



Now, we proceed accomplishing the phase portrait (direction field + trajectories = solution curves) with the solution functions using the built in Runge-Kutta method:

$$\mathrm{RK}\left(\left[x - y,\ 1 - x^2\right],\ [t,\ x,\ y],\ [0,\ -4,\ -4],\ 0.1,\ 20\right)$$

$$\begin{bmatrix} 0 & -4 & -4 \\ 0.1 & -3.922935156 & -5.479650234 \\ 0.2 & -3.687382967 & -6.838140350 \\ 0.3 & -3.295224611 & -7.967179885 \\ 0.4 & -2.758336585 & -8.792571965 \\ 0.5 & -2.095172969 & -9.289818246 \\ 0.6 & -1.325838354 & -9.490076438 \\ 0.7 & -0.4664763402 & -9.477799034 \\ 0.8 & 0.4764633298 & -9.385223377 \end{bmatrix}$$

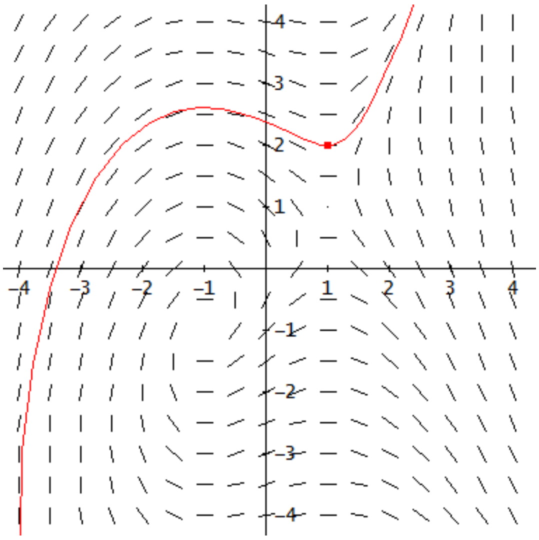
The result is a 3-columns matrix with *t*-values from 0 to 2 (step 0.1) and the respective function values of the *x*(*t*)- and *y*(*t*)-solution curve with initial point (-4,-4). Changing the step from 0.1 to -0.1 will give the values in the reverse direction.

All what remains to do, is to extract columns [1,2] and [1,3] for plotting the solution curves and **columns [2,3] for the phase portrait**:

```
(RK([x - y, 1 - x^2], [t, x, y], [0, 1, 2], 0.1, 20)) COL [2, 3]

(RK([x - y, 1 - x^2], [t, x, y], [0, 1, 2], -0.1, 20)) COL [2, 3]
```

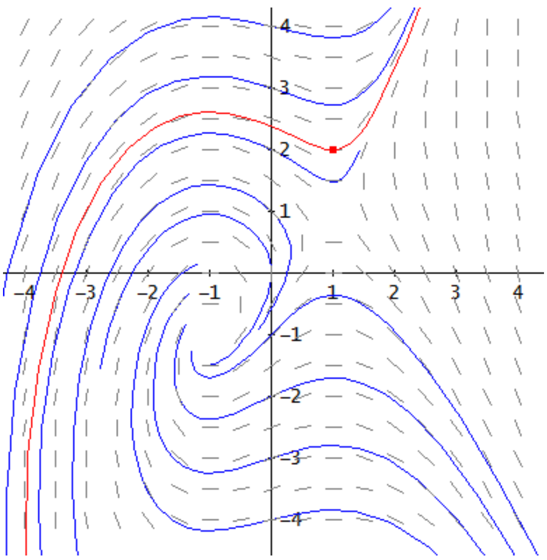
Switch to the 2D-plot window and plot (without evaluating in the Algebra window):



This is the phase diagram of the system with initial point (1,2).

The following VECTOR-construct allows to plot a family of phase diagrams in one single step:

```
VECTOR(VECTOR((RK([x - y, 1 - x^2], [t, x, y], [0, 0, k], 0.1·t_, 20)) COL [2, 3], k, -4, 4), t_, [-1, 1])
```



Plots of all curves with initial points on the *y*-axis (from -4 to 4 step 1) with t-steps in both directions (the outer VECTOR-command). Looks quite nice.

With 0.5-steps for *k* you will receive a denser net of curves:

```
VECTOR(VECTOR((RK([x - y, 1 - x^2], [t, x, y], [0, 0, k], 0.1·t_, 20)) COL [2, 3], k,
  -4, 4, 0.5), t_, [-1, 1])
VECTOR(VECTOR((RK([x - y, 1 - x^2], [t, x, y], [0, k, 0], 0.1·t_, 20)) COL [2, 3], k, 0,
  4, 0.5), t_, [-1, 1])
```

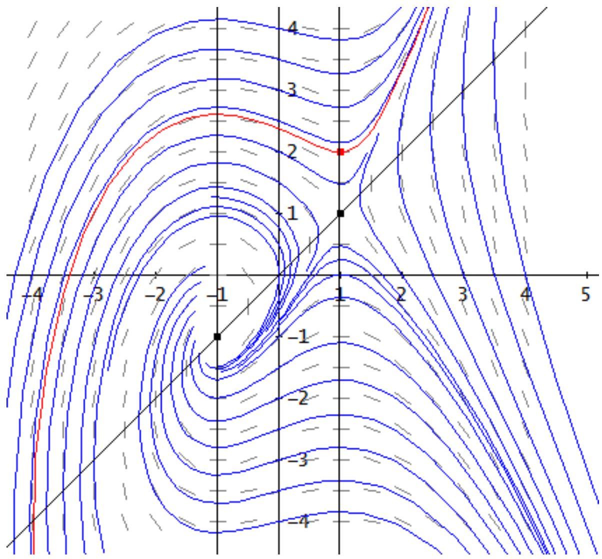
We can add the *fixed points* and finally, we plot the *nullclines* (all in black):

The nullclines are the loci of the points in the direction field with horizontal (*y*-nullcline) and vertical (*x*-nullcline) slope.

$$\text{SOLUTIONS}([x - y = 0,\ 1 - x^2 = 0],\ [x,\ y]) = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}$$

$$x - y = 0$$

$$1 - x^2 = 0$$

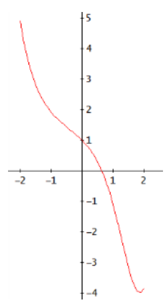


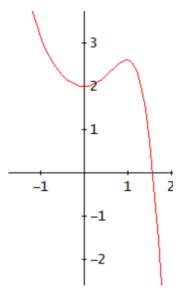
A good explication can be found at:

https://mcb.berkeley.edu/courses/mcb137/exercises/Nullclines.pdf

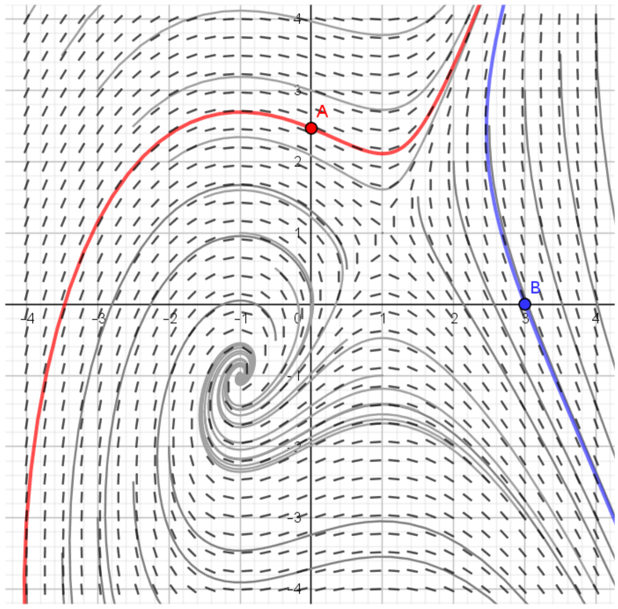
Finally, I'd like to plot the solution curves with initial values (1,2) for $x(t)$ and $y(t)$. Using the VECTOR-construction from above you could plot a family of solution curves in one single step.

```
(RK([x - y, 1 - x^2], [t, x, y], [0, 1, 2], 0.1, 20)) COL [1, 2]
(RK([x - y, 1 - x^2], [t, x, y], [0, 1, 2], -0.1, 20)) COL [1, 2]
(RK([x - y, 1 - x^2], [t, x, y], [0, 1, 2], 0.1, 20)) COL [1, 3]
(RK([x - y, 1 - x^2], [t, x, y], [0, 1, 2], -0.1, 20)) COL [1, 3]
```





This is a GeoGebra plot of the phase plane:

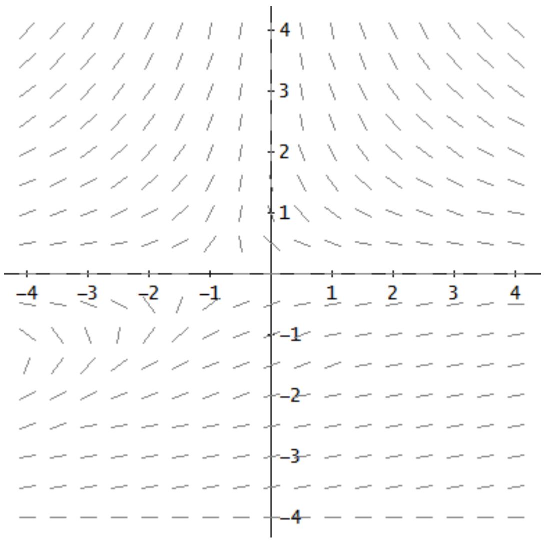


I will turn to Strogatz Example 6.1.1

Given is the system $\begin{array}{l} \dot{x} = x + e^{-y} \\ \dot{y} = -y \end{array}$.

The plot of the direction field is obtained like above:

$$\text{DIRECTION\_FIELD}\left(-\frac{y}{x + e^{-y}},\ x,\ -4,\ 4,\ 16,\ y,\ -4,\ 4,\ 16\right)$$



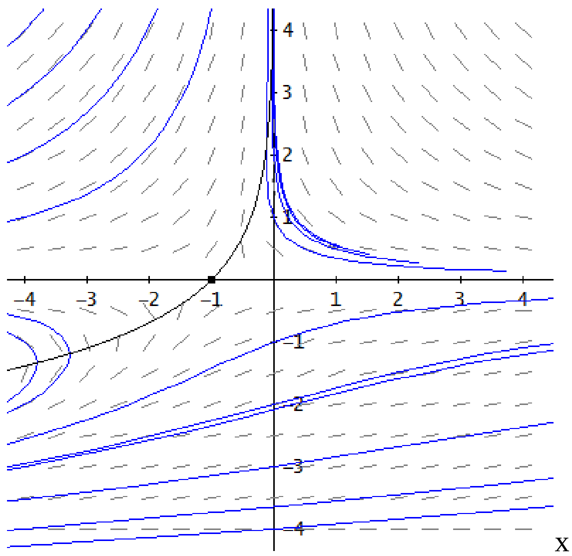
This gives a family of solution curves (blue):

```
VECTOR(VECTOR((RK([x + e^-y, -y], [t, x, y], [0, 0, k], 0.1·t_, 20)) COL [2, 3], k, -4, 4), t_, [-1, 1])
VECTOR(VECTOR((RK([x + e^-y, -y], [t, x, y], [0, -4, k], 0.1·t_, 20)) COL [2, 3], k, -4, 4), t_, [-1, 1])
```

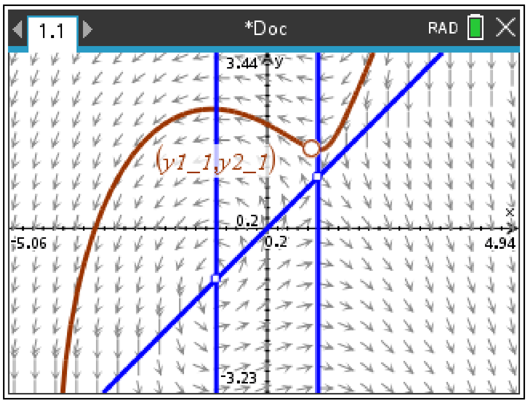
I add the fixed points and the nullclines (black):

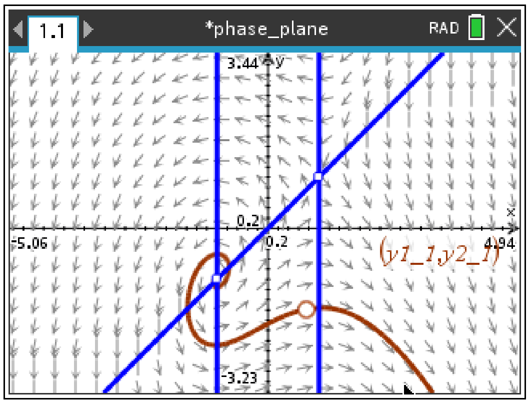
We don't need a CAS to find out that ($x + e^{-y} = 0$, $-y = 0$) have the solution $y = 0$ and $x = -1$. So, the fixed point is at (-1,0).

The nullclines are the curves $x + e^{-y} = 0$ (= $y = -\ln(-x)$) and $y = 0$.

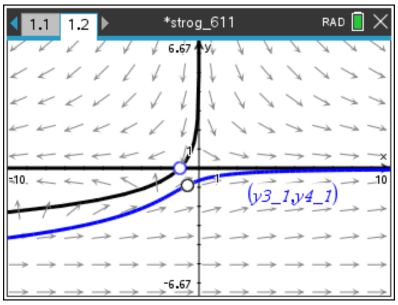


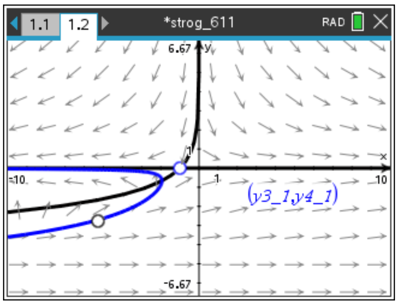
Two Nspire screenshots for the first example (textbook):

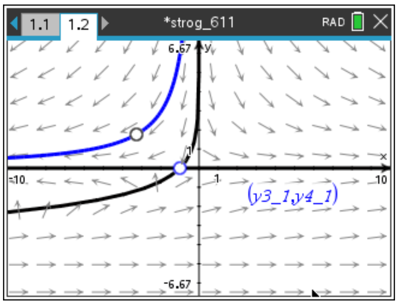




Followed by three screens for the second system (Strogatz 6.1.1):

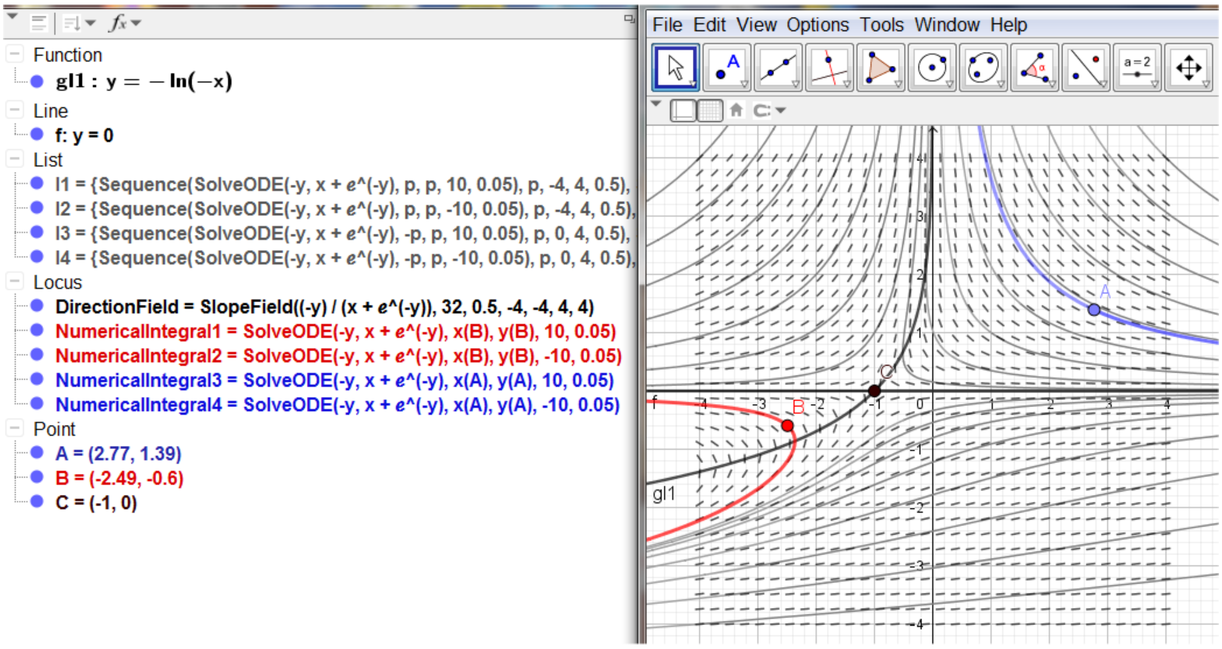






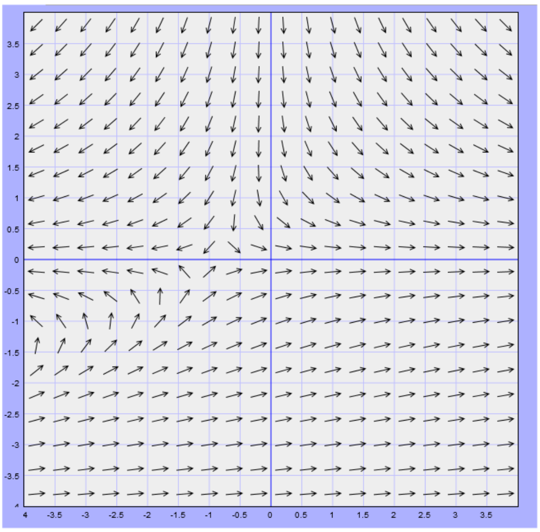
I was not able to plot more than one solution curve with TI-Nspire. Is there anybody who knows how to do this? Please let me know.

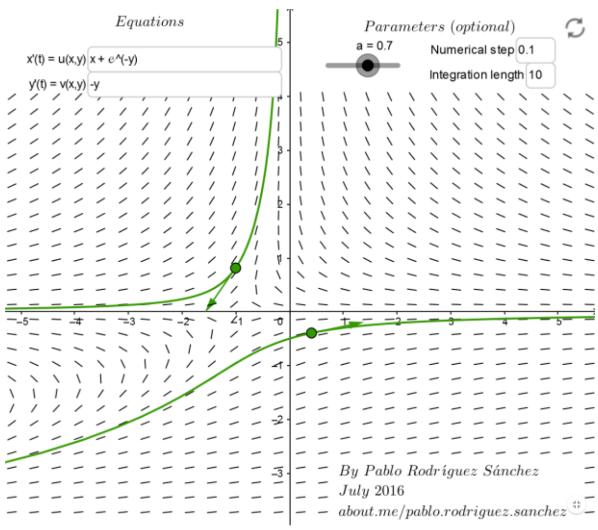
I used GeoGebra to plot the direction field, a family of solution curves, the nullclines and two points which can be dragged through the plane together with the respective trajectories:



A **phase portrait** is a geometric representation of the trajectories of a dynamical system in the **phase plane**. Each set of initial conditions is represented by a different curve, or point. **Phase portraits are** an invaluable tool in studying dynamical systems. ... An attractor is a stable point which is also called 'sink'. (Wikipedia)

Two links to interactive applets:





https://aeb019.hosted.uark.edu/pplane.html https://www.geogebra.org/m/utcMvuUy

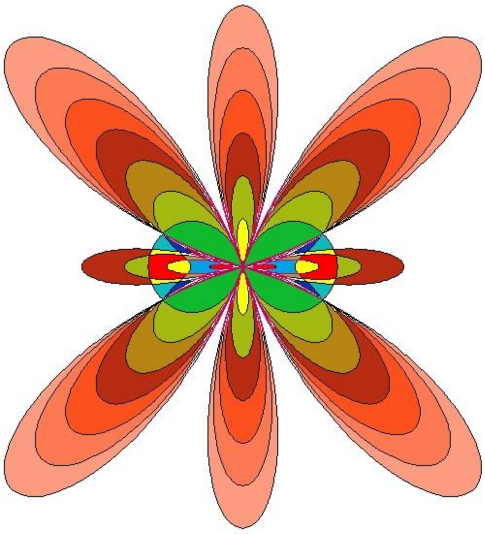
and https://www.geogebra.org/m/s8zdVwt7

Spring Time – Flower Time – Butterflies Awake

When I came across the pretty picture given on page 1 (nationalcurvebank website) I became curious to search for more information about the Butterfly Curve. After Wikipedia and some other resources, I found Temple H. Fay's one-page article "*The Butterfly Curve*". He discovered this transcendental curve when investigating *petal curves* (not to be confused with *pedal curves*).

https://www.jstor.org/stable/2325155?read-now=1&seq=1

```
VECTOR(EXP(COS(2·t)) - a·COS(4·t), a, 6.5, 0.5, -1)
```

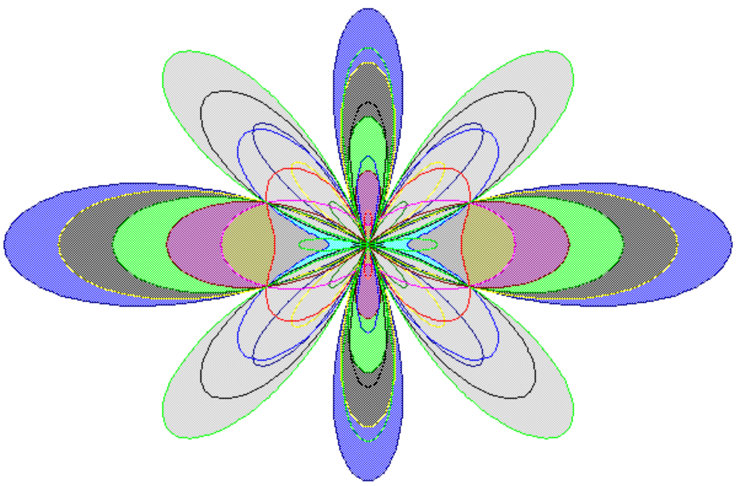


A family of curves colored using a special DERIVE feature (converting the plot to a bitmap image). How to do this is described by Tania Koller in DNL#63 from October 2006).

But we can shade regions in various colors defining inequalities as shown below. Unfortunately, the colors provided by Derive for shading regions are not really bright.

```
VECTOR(r ≤ EXP(COS(2·t)) - a·COS(4·t), a, -4, 4, 1)

VECTOR(EXP(COS(2·t)) - a·COS(4·t), a, -4, 4, 1)
```

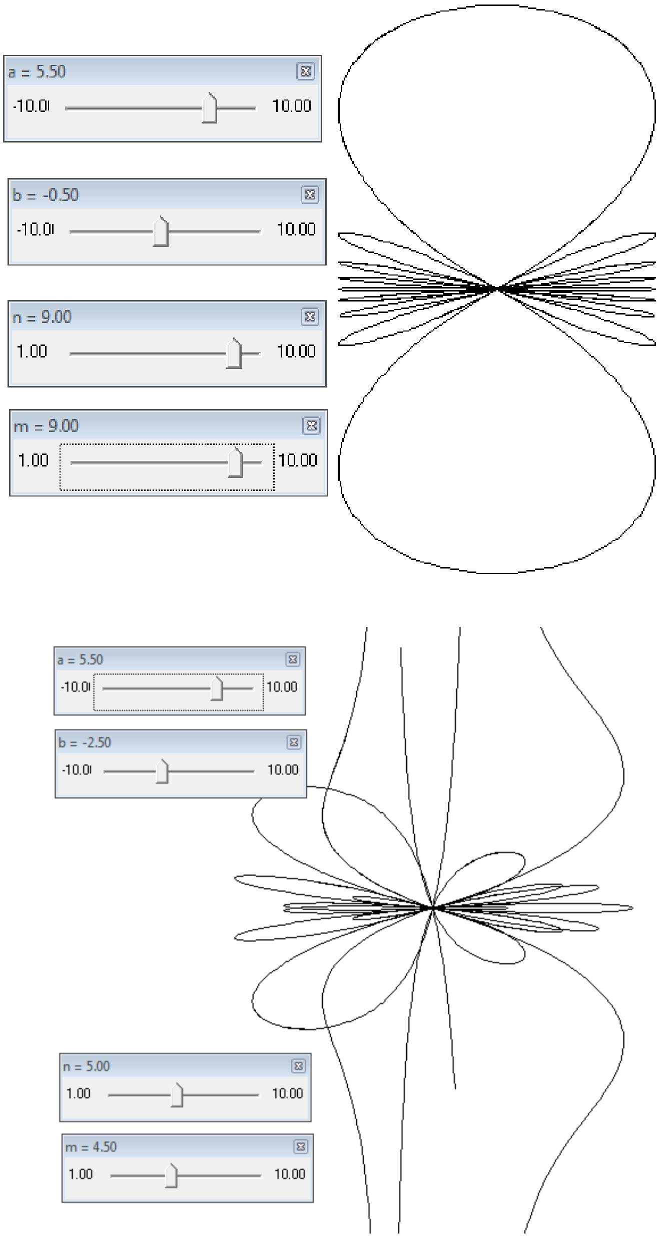


Temple H. Fay is now Professor Extraordinaire at Tshwane University of Technology. I had the pleasure and honor to meet him at the TIME 2009 Conference held in South Africa. By the way, my best regards to Steve Joubert (also from Tshwane University).

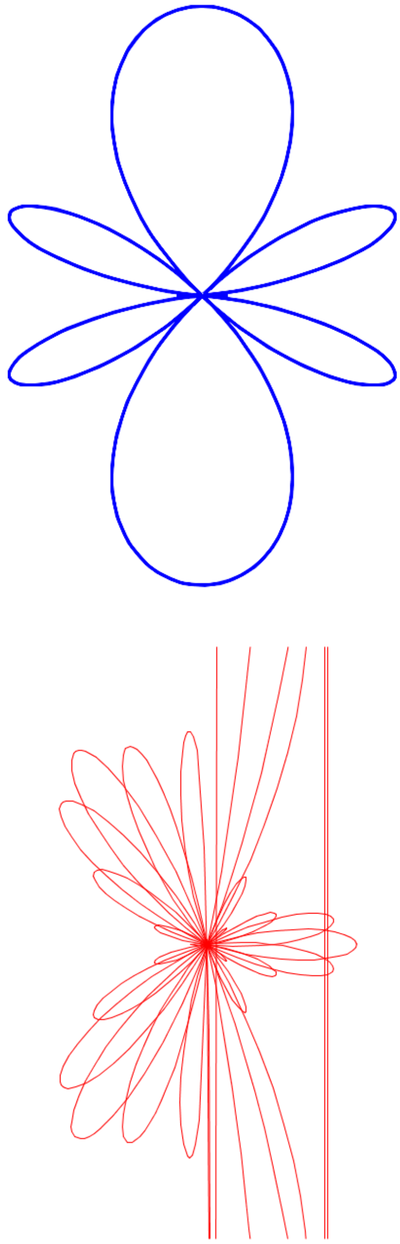
Temple investigated petal curves of general form $\rho = \dfrac{a\cos(n\theta) + b\cos(m\theta)}{\cos(\theta)}$. He recommends to try $a < b$, $a > b$, $a = b$, n is even, n is odd, …

Temple proposes to ask students predicting the number of petals with both m and n odd; what happens if (at least) one of them is even. The sliders make it easy to apply rational numbers for the parameters. Range for t must be adapted.

Derive (and TI-Nspire as well) offer sliders to experiment in all directions:



DERIVE



TI-Nspire

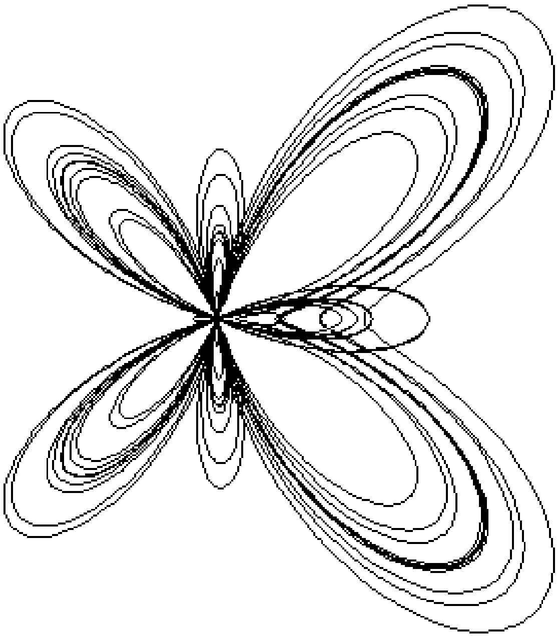
At the end of his paper Temple Fay writes about his discovering of the "*most interesting and beautiful of all the curves*".

$$\rho = e^{\cos(2\phi)} - 1.5\cos(4\phi) + \sin^2\left(\frac{\phi}{12}\right)$$

$$0 \le \phi \le 24\pi$$

This is the original Temple Fay butterfly:

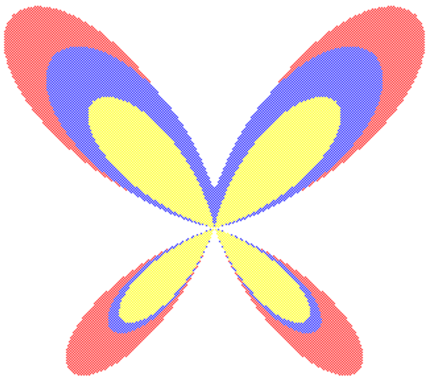
A small change in one argument gives the butterfly a 90° turn and now we can create butterflies as we like:

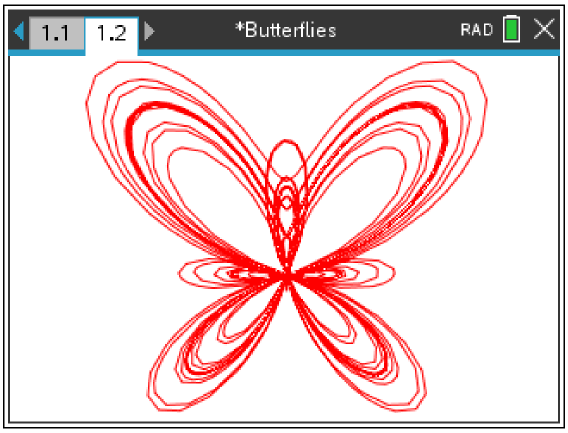


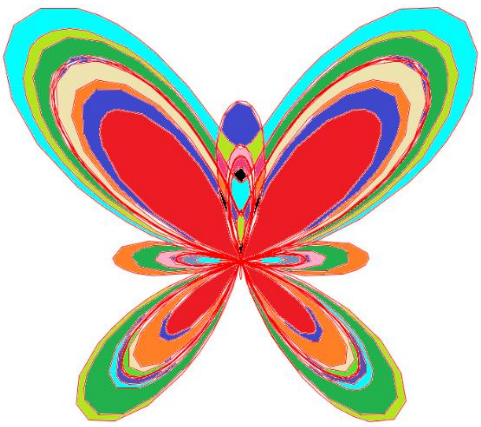
$$r < \mathrm{EXP}(\mathrm{SIN}(t)) - 2\cdot\mathrm{COS}(4\cdot t) + \mathrm{SIN}\left(\frac{2\cdot t - \pi}{24}\right)^5$$

$$r < \mathrm{EXP}(\mathrm{SIN}(t)) - 3\cdot\mathrm{COS}(4\cdot t) + \mathrm{SIN}\left(\frac{2\cdot t - \pi}{24}\right)^5$$

$$r < 0.5\cdot\mathrm{EXP}(\mathrm{SIN}(t)) - 2\cdot\mathrm{COS}(4\cdot t) + \mathrm{SIN}\left(\frac{2\cdot t - \pi}{24}\right)^5$$



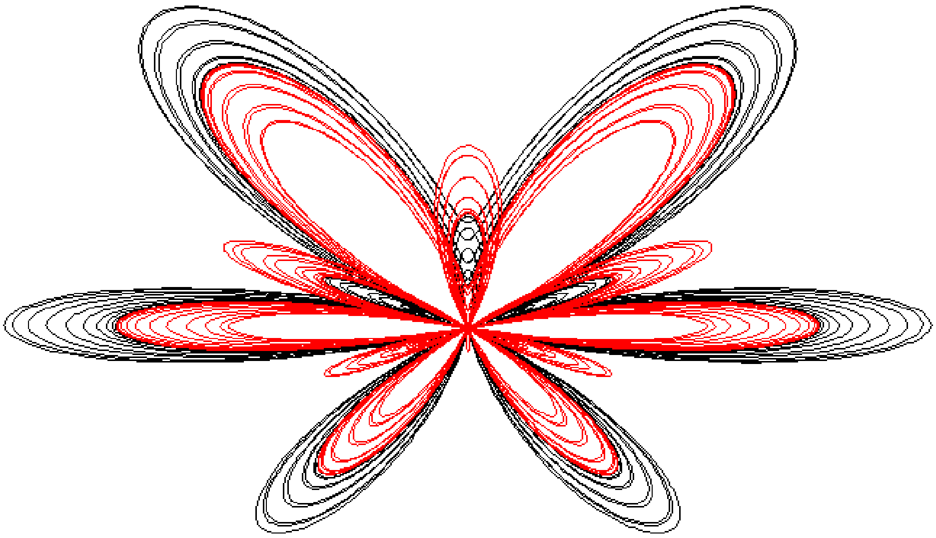




I colored the butterfly using a painting program

Another butterfly form is given by *Clifford Pickover*: The MαTHβOOK:

$$\mathrm{EXP}\left(\cos\left(t-\frac{\pi}{2}\right)\right)-2.1\cdot\cos\left(6\cdot\left(t-\frac{\pi}{2}\right)\right)-\mathrm{SIN}\left(\frac{t-\frac{\pi}{2}}{30}\right)^7$$



Young Hee Geum's paper on Butterfly Curves can befound at:

https://www.researchgate.net/publication/232899918_On_the_analysis_and_construction_of_the_butterfly_curve_using_Mathematica_R/link/594cf2e9a6fdcc79e18cc97f/download

Watch animated "Butterflies":

https://www.geogebra.org/m/CjPFXGYj

https://www.youtube.com/watch?v=MCQljZM-jF0

These butterflies are from our garden (summer butterflies):