

THE BULLETIN OF THE



USER GROUP

+ TI 92

C o n t e n t s :

- | | |
|----|--|
| 1 | Letter of the Editor |
| 2 | Editorial - Preview |
| | Terence & Josefine |
| 3 | Reflections 1 & 2 |
| | Albert Rich |
| 5 | A Brief History of the muMath / DERIVE CASs |
| | José M. M. Cardia Lopes & Gilberto Pinto |
| 6 | Some Classroom Experiments with <i>DERIVE</i> |
| | in a Chemical Engineering Undergraduate Course |
| | Steven Schonefeld |
| 13 | Plotting 3-Dimensional Curves with SPACE_TUBE |
| | Eugenio Roanes Lozano |
| 19 | Propositional Multivalues Logics with the |
| | TI Symbolic Calculators |
| | Walter Schiller |
| 30 | Programming with DERIVE – A Compiler |
| | Josef Böhm |
| 44 | <i>DERIVE</i> as Problem Generator |
| | Thomas & Josef |
| 47 | Trochoids on the TI-92/TINspire |
| | Josef Böhm |
| 48 | The RUN-Test |
| 54 | <i>DERIVE</i> and TI-92 User Forum |

- [1] **Using the TI-89 in Physics**, A textbook with examples for students, George Adie, bk-teachware, no SL-14, ISBN 3-901769-31-5 (+ diskette).
- [2] **Experiments in Motion Using the CBR and TI-83+**, A manual of Activities and their Analysis, Heinz-Dieter Hinkelmann, bk-teachware, no SL-13, ISBN 3-901769-29-3.
- [3] **Exploring Integration with the TI-89/92/92+**, From Counting Raindrops to the Fundamental Theorem, J Böhm & W Pröpper, bk-teachware, no SL-12, ISBN 3-901769-28-5 (+ diskette).
- [4] **Exam Questions and Basic Skills in Technology-Supported Mathematics Education**, Proceedings of the 6th ACDCA Summer Academy in Portoroz, Slovenia, Jul 2-5, 2000, bk-teachware, no SL-15, ISBN 3-901769-33-1.
- [5] **Abituraufgaben lösen mit TI-89/92/92+**, Praktische Anleitungen und Lösungen aus Analysis u. Analytischer Geometrie, G Raup & W Scheu, bk-teachware, Nr SR-18, ISBN 3-901769-30-7.

All titles can be ordered at <http://www.bk-teachware.com> **No more valid in 2018**

Important note on TI-92 programs: Unfortunately changes in the TI-92+ Operating System's source code (2.05) can cause problems running programs delivered on the diskettes which accompany some bk-teachware books. The same can happen as a consequence of localizations (German or other menus) for the PLUSes. In that case please send an e-mail and I'll send an update.

☞ New DERIVE discussion group installed ☞

David Halprin, a very enthusiastic DERIVE User and DUG member from Australia established a DERIVE discussion group **eDUG**. David and I would like to invite you all to join eDUG by clicking on

<http://www.egroups.com/group/edug>.

To post a message: the groups e-mail address is: eDUG@egroups.com.

Many thanks to David for his efforts to improve communication within the *DERIVE*-family.

ICTMT5 Klagenfurt, August 6 - 10, 2001(*)

In the frame of this Conference Bernhard Kutzler, Vlasta Kokol-Voljc and Josef Böhm will together chair a Special Working group

DERIVE, TI-89/92 and other CAS

Education has become one of the fastest growing application areas for computers in general and computer algebra in particular. Computer algebra tools such as TI-89/92, Derive, TI Interactive, Mathematica, Maple, Axiom, Reduce, Macsyma, or MuPAD make powerful teaching tools in mathematics, physics, chemistry, biology, economy, at all grade levels (schools and universities).

The goal of this session is to exchange ideas and experiences from various CAS, to hear about classroom experiments, and to discuss all issues related to the use of computer algebra tools in classroom (such as assessment, change of curricula, new support material, ...).

If you have anything of the above which you would like to share with colleagues in the inspiring atmosphere of a conference in one of Austria's most famous holiday regions then please make a submission using the submission and registration forms presented on the Conference's web site.

Deadline is 31 January 2001

b.kutzler@eunet.at or nojo.boehm@pgv.at

(*) Conference web site: <http://www.uni-klu.ac.at/ictmt5/>

We hope to meet many of you in Klagenfurt.

thanks for your cooperation.

Wir ersuchen unsere Mitglieder aus Österreich und Deutschland, den beigelegten Zahlschein zu benutzen. Herzlichen Dank.

Find all the DERIVE and TI-files on the following web sites

Dear DUG members,

End of 2000 is approaching and we are completing the first DNL-decade. DNL#40 is ready and the last work for me is writing the *Letter of the Editor*. I have to start with an apologize. In the last DNL some problems with my (PC's) memory caused an uncomplete print of Johann Wiesen-bauer's Titbits#18. The printer refused to print the mathematical symbols and the formulae as well. As they were not very large I didn't realize that. I beg your pardon. On the diskette accompanying this DNL you can find Johann's file in his original version.

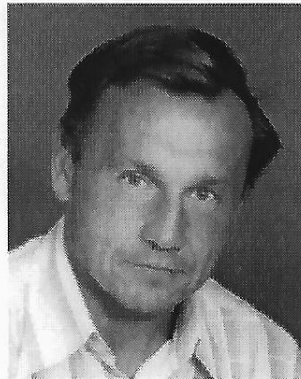
This DNL contains among other exciting contributions a really extraordinary one. One month ago Mr Schiller sent a heavy letter and wrote that he had produced a *DERIVE* compiler which enables to produce *DERIVE* programs for *DERIVE5* using an editor and several user friendly instructions instead of the very tight *DERIVE* code.

I tried his tool and was more than surprised. I called Mr Schiller and made some proposals for possible extensions and two weeks later the improved version was back in Würmla.

You can find some examples how to apply TOM.EXE in this *DNL* and on the diskette

as well. (TOM is from to mth) Unfortunately Mr Schiller is not reachable by Internet, so if you have some questions then please send an email to my address or contact Mr Schiller directly.

It is always a pleasure to have memories on meetings in the DNL. So we are very grateful for Josefine Sjöstrand and Terence Etchells' *Reflections* on the Liverpool Conference.



By the way, please take into account the announcement of the *ICTMT5* 2001 in Klagenfurt. I would be very happy to meet many of you in Austria in one of our loveliest regions.

Concerning the training program for *DfW5* on page 39: I try to produce a very similar program for *DERIVE4* and *DfD*, too. Please look for it on the diskette.

There are a lot of new materials on our ACDCA / T³ homepage (see address below).

My wife Noor and I wish you all a

**Merry Christmas and a
Happy New Year 2001**

and we hope to meet you next year again in the *DERIVE* and TI-92 User Group.

Josef

A handwritten signature in dark ink, appearing to be 'Josef'.

Don't forget to renew your DUG - membership. Please use the included form to send or fax your data. You can also use email. Please avoid cheques if possible. This is the most expensive way to settle the dues for both partners. Many thanks for your cooperation.

Wir ersuchen unsere Mitglieder aus Österreich und Deutschland, den beigelegten Zahlschein zu benutzen. Herzlichen Dank.

Find all DERIVE and TI files on

<http://www.austromath.at/dug>

<http://www.acdca.ac.at>

The *DERIVE-NEWSLETTER* is the Bulletin of the *DERIVE & TI-92 User Group*. It is published at least four times a year with a contents of 44 pages minimum. The goals of the *DNL* are to enable the exchange of experiences made with *DERIVE* and the *TI-92/89* as well as to create a group to discuss the possibilities of new methodical and didactical manners in teaching mathematics.

As many of the *DERIVE* Users are also using the *TI-92/89* the *DNL* tries to combine the applications of these modern technologies.

Editor: Mag. Josef Böhm
A-3042 Würmla
D'Lust 1
Austria
Phone/FAX: 43-(0)660 3136365
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 & TI-92 Newsletter* will be.

Next issue: March 2001
Deadline 15 February 2001

Preview: Contributions for the next issues

Inverse Functions, Simultaneous Equations, Speck, NZL
A Utility file for complex dynamic systems, Lechner, AUT
Some examples how to work with DfW5, Lechner, AUT
Examples for Statistics, Roeloffs, NL
Quaternion Algebra, Sirota, RUS
Sand Dunes, Halprin, AUS
Type checking, Finite continued fractions, Welke, GER
Kaprekar's "Self numbers", Schorn, GER
Flatterbandkurven, Rolfs, GER
Comparing statistics tools: a pie chart with DERIVE, a stem & leaf diagram on the TI, Böhm, AUT
Errors occurring presenting graphs on the TI-92 compared with DERIVE and TI-Interactive, Himmelbauer, AUT
Some special Integrals for DERIVE, Magiera, POL
VECTOR is my favourite DERIVE-function, Böhm, AUT
and
Setif, FRA; Vermeylen, BEL; Leinbach, USA; Aue, GER; Koller, AUT,

Impressum:

Medieninhaber: DERIVE User Group, A-3042 Würmla, D'Lust 1, AUSTRIA
Richtung: Fachzeitschrift
Herausgeber: Mag. Josef Böhm
Herstellung: Selbstverlag

**Reflections of an Organiser of the 4th International Derive-TI89/92
Conference, Liverpool John Moores University, July 2000.**

There was an eerie feel to the University when the last of the delegates had left the conference. Walking around the building, taking down the signs and notices, it felt like being on a set of a deserted town in a spaghetti western. I was half expecting tumbleweed to rush past me as the wind swirled around. After the rush of activity throughout the previous days the silence was deafening and my emotions of relief and exhaustion were quickly overcome with those of emptiness and sadness. All those old and new friends had gone. The party was over.

Five days earlier, it was all hands to the pumps. I entered the conference office Tuesday morning to find Wade and Jane Ellis, who had just flown in from the USA, stuffing information into the conference bags. Ian Malabar was on the phone finalising the arrangements for the conference trips. Paul Cartwright and his team of technicians were setting up the last of the computers, projectors and TI view screens in all the lecture rooms. Pat and Carl Leinbach were at Lime Street Station meeting conference delegates and Dave Pountney was printing the final version of the delegates list. There were a few other niggling jobs that had to be done, but we were ready. The conference rooms were set, the catering organised, the delegate information collated, the trips were booked. The roller coaster had reached the top, it was ready to drop, I held my breath

The registration of delegates was hectic, every few seconds familiar faces would arrive, lots of hugs and handshaking. No time to catch up on the last two years, catch you later, next delegate "Hiyaaa". Soon it was time for the conference opening and the first keynote lecture. Time seemed to speed up as I began frantically running around, checking that every speaker was happy with the technical equipment, this manifested itself in anxiously peering through door windows, all seemed well. As the hum on the corridors subsided I managed to sneak in the back of Pat and Carl Leinbach's talk on "Estimating time since death" only to be confronted with a picture of a dead body, Jeez I had to get out of there!

In the evening we had a brewery trip and a Beatles magical mystery tour, all seemed to be going well. Thursday we had trips to Alton Towers, Chester and Liverpool with a quiz in the conference bar in the evening. Friday morning started with registration again, this time for the teachers day. The conference committee had invited local school teachers to attend the conference for the day to experience computer algebra at an international level. Boy, what a busy day! The Friday evening was, by many peoples accounts, the highlight of the conference, with the conference banquet and the rock band "The lounge lizards" (who was that little guy on keyboards and acoustic guitar?). The evening ended with dancing, with the conference team really letting their hair down and Bernhard Kutzler gracing us with his quite magnificent dancing. Saturday morning came with some rather sore heads! Conference ended at noon and people drifted away. We took down the signs; cleared the equipment and went to Dr Duncan's pub for (as Karen Stoutemyer calls them) a repair beer.

Without doubt the conference had been a success, made so in no small part by my great friend Carl Leinbach who was on sabbatical with us for the year. Carl was the real workhorse of the conference and who I owe a great debt of gratitude, for his experience and shear hard work during the build up towards the conference.

Would I do it again? We'll I surely enjoyed it, it was hard work but very rewarding. "Yes, but would you do it again?" . Erm!!! Not for at least ten years and only if I had Carl and Dave Pountney as my partners in crime.

Good luck to the conference organisers of the next conference in Vienna 2002.
Cheers

Terence

Lovely Liverpool

Liverpool was an excellent place for the 4th Derive/TI-89-92 conference. After the first time I went there as a fifteen-year-old beatle fan, I always wanted to come back, and this conference was a perfect reason for it. Things just couldn't be better: A bunch of intelligent, nice people coming together to exchange ideas and have fun in a city most of all known as the birthplace of the Beatles!

This time it was slightly difficult to get to the conference. My parents and I had to wait for eight hours at the airport before we could fly to London, since the British Airways flight we were going on was cancelled due to problems with the fuel system. As soon as we landed at London Gatwick, we realised that we had missed the last train to Liverpool. Vexation! We desperately wanted to come to Liverpool, and the very thought of having to stay one night at an airport hotel was slightly depressing.

Even though we hardly believed it, we found ourselves on a northbound train the day after. We were travelling through a beautiful landscape, and the farther north we came, the nicer became the people's accents. After nearly four hours, we were in Liverpool! We were totally happy for a few minutes, before we realised that we didn't know where to find the hotel. A second after that, we realised that there were only ten minutes before the registration desk at the conference would close....and even worse, we didn't know where the conference was!

We had obviously forgotten all the information at home! After about ten minutes, we knew that Liverpool John Moore's University wasn't just one building. There were several, spread all over the city. That's pretty bad when you don't know where you're supposed to go. After that we had asked a few people about the way and they all had given us different answers, we found one building that clearly belonged to the university. Perhaps this was the place we were looking for? No, it wasn't. But a friendly lady told us the way to the right place, and soon we were there, in the same room as our friends from all over the world. It was wonderful to see everyone again, and we had soon forgotten all the trouble we had had while we were travelling.

I have loads of lovely memories from these four days in Liverpool. From early morning to late at night, there was always something going on. On the first evening, mum, dad and I had decided to go on the Magical Mystery Tour to see all the places in the city that had something to do with the Beatles. While we were sitting there on the bus, I came to think of the fact that it was exactly three years since I was on my first Magical Mystery Tour. That added some extra magic to an already excellent trip.

I also remember the lovely day in Chester, with all its Roman remains and medieval houses and the boat trip on the river that allowed us to catch a glimpse of the magnificent home of the Duke of Westminster's. Later that night, many of us enjoyed a difficult quiz, which was also an excellent opportunity to get to know people better. I had a great time with both old (yep, Uncle Terence admitted he was already 40☺) and new friends, and I can still recall the feeling of total happiness that came over me that night. It was going to remain with me for a long time after I left Liverpool.

The last evening in Liverpool was a splendid one. We had a lovely dinner and conversations that were perhaps even nicer than the excellent food. After that, it was time for the Lounge Lizards to enter the stage. Terence Etchells, Phillip Yorke and friends had a great band and they had picked some great songs that they performed. I helped them a little with the piano playing on "Get Back" and additional vocals on "Can't buy me love". I also sang Queen's "You take my breath away". After a while, people began to dance, especially Bernard and Andrea Kutzler. They were both brilliant dancers and it was truly enjoyable to see them dance together.

Since all these events took place in Liverpool, birthplace of The Beatles, I would like to give these four lads the last words of this text:

Though I know I'll never lose affection

*For people and things that went before
I know I'll often stop and think about them
In my life
I'll love you more.*

See you in Vienna!
Josefine Sjöstrand

A Brief History of the *muMATH/DERIVE*® CASs

Albert Rich, Soft Warehouse, Honolulu, Hawaii

On 1 January 1979, a partnership named The Soft Warehouse was founded by Albert D. Rich and David R. Stoutemyer. At the time computer algebra systems (CASs) were only available on large mainframe computers, usually at academic institutions. Our goal was to make computer algebra widely available to the masses on small computers (note that this was well before the term "Personal Computer" had been invented).

muMATH-79 was released in 1979 and ran on 8080 and Z80 computers with as little as 48K bytes of memory running CP/M, and on Radio Shack TRS-80 computers running TRS-DOS.

muMATH-80 was released in 1980 and ran on the above computers as well as the 6502 based Apple II computers.

muMATH-83 was released in 1983 and ran on the above computers as well as the 8088 based IBM PC and XT computers with as little as 300K bytes of memory.

On 5 February 1985 the company was incorporated under the name Soft Warehouse, Inc.

DERIVE was released in October 1988, had an easy to use menu-oriented CAS interface, 2D and 3D graphics, and ran on PC compatible computers running MS-DOS with a minimum of 512K bytes of memory.

DERIVE for Windows was released in October 1996, had a GUI Windows interface, a 32-bit math engine kernel, and ran on PC compatible computers running MS Windows and NT.

The "mu" in **muMATH** is the Latin name for the Greek letter mu, which is used to represent micro in the Metric system of units. Since our math program ran on the micro-processors used in small computers, the name **muMATH** seemed like a natural.

muMATH was written in a surface language for LISP that we named **muSIMP**. **muSIMP** stands for micro Symbolic IMplementation language. While semantically equivalent to LISP, **muSIMP** provides a more conventional syntax than LISP (e.g. infix notation for math operators instead of LISP's Cambridge prefix notation, etc.). **muSIMP** starts out as **muLISP**, and then the first thing that is loaded is a parser (written in **muLISP**) that replaces the LISP parser with the more sophisticated **muSIMP** parser.

For example, in LISP (or **muLISP**), you would write $(+ 2 3)$ whereas in **muSIMP** you would write $2+3$. As another example, in **muLISP** you would write

$$((\text{ZEROP } x) y)$$

whereas in **muSIMP** it would be

When $x=0$, y Exit,

Rather than just refining and improving **muMATH**, we decided that an entire re-write was needed. **DERIVE** is the result. Instead of being written in **muSIMP**, **DERIVE** is written directly in LISP, specifically **muLISP**. More importantly, in **DERIVE** expressions are represented in an implicit form that makes for much more compact storage and efficient algorithms.

We agonized over the name for the successor to **muMATH** for a long time. We wanted a name that suggested the dynamic, creative process of doing math on a computer. So we finally converged on the verb "**DERIVE**", rather than a static noun beginning with "M".

SOME CLASSROOM EXPERIMENTS WITH DERIVE IN A CHEMICAL ENGINEERING UNDERGRADUATE COURSE.

José M.M. Cardia Lopes, Gilberto Pinto, IPP/ISEP, Oporto, Portugal

e-mail: cardialopes@mail.telepac.pt

DERIVE is a valuable learning tool in teaching not only calculus and algebra but also numerical analysis. With DERIVE we can make evident the differences and the complementarities between the two approaches (calculus and numerical). Let us report some examples.

Example 1 - Townend and Pountney (1989, p.77)

"A horizontal cylinder closed at one end and fitted with a piston, contains an ideal gas, initially at pressure p_1 . The gas undergoes a reversible isothermic expansion, throughout which the outer face of the piston is exposed to atmospheric pressure p_a and is acted upon by a variable external applied force for which the net work done is zero. If the final pressure of the gas is p_2 then, by the first law of thermodynamics, p_2 satisfies the equation

$$\frac{p_1}{p_a} \ln\left(\frac{p_1}{p_2}\right) = \frac{p_1}{p_2} - 1$$

If $p_1 = 124 \text{ KN m}^{-2}$ and $p_a = 103.5 \text{ KN m}^{-2}$ are given, compute p_2 from the equation."

Introducing the equation in canonical form and solving for p_2 we get $p_2 = 124 \text{ KN m}^{-2}$. It is not the solution for our problem: after expansion it is obvious that we will have $p_2 < p_1$.

#1: $p_1 := 124$

#2: $p_a := 103.5$

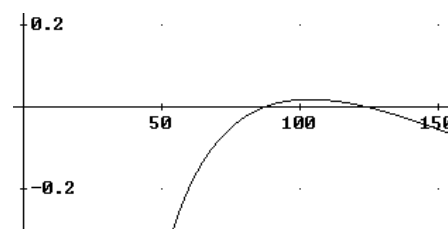
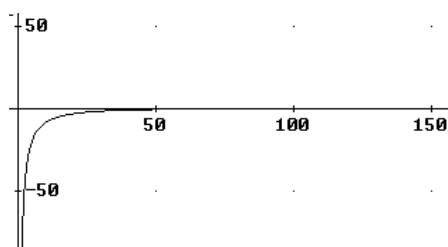
#3: $p_2 \in \text{Real}$

#4: $\frac{p_1}{p_a} \cdot \text{LN}\left(\frac{p_1}{p_2}\right) - \frac{p_1}{p_2} + 1$

#5: $\text{SOLVE}\left(\frac{p_1}{p_a} \cdot \text{LN}\left(\frac{p_1}{p_2}\right) - \frac{p_1}{p_2} + 1, p_2\right)$

#6: $[p_2 = 124, p_2 = \infty, p_2 = -\infty]$

A glance on the graph can help us (we need some trials to get the adequate scales): we have evidence that the problem is almost ill conditioned, with two roots, and then we must solve (numerically) for the lower root. The answer is $p \approx 87.3 \text{ atm}$.



#7: $\text{SOLVE}\left(\frac{p_1}{p_a} \cdot \text{LN}\left(\frac{p_1}{p_2}\right) - \frac{p_1}{p_2} + 1, p_2, 50, 100\right)$

#8: $[p_2 = 87.2802]$

Solving the equation with DERIVE 6: (we proceed with DERIVE 6)

#1: [p1 := 124, pa := 103.5, p2 :=]

#2: $\text{SOLVE}\left(\frac{p1}{pa} \cdot \ln\left(\frac{p1}{p2}\right) - \frac{p1}{p2} + 1, p2\right)$

Result for Simplify #2

#3: $p2 \cdot e^{\frac{207}{(2 \cdot p2)}} = 124 \cdot e^{\frac{207}{248}}$

Result for Approximate #2


#4: p2 = 124

#5: $\text{NSOLVE}\left(\frac{p1}{pa} \cdot \ln\left(\frac{p1}{p2}\right) - \frac{p1}{p2} + 1, p2\right)$

#6: p2 = 87.28048688

Solving the equation with TI-NspireCAS:

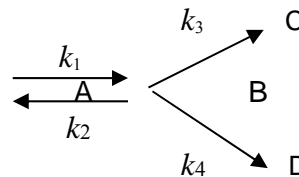
©Example 1

 $\text{solve}\left(\frac{p1}{pa} \cdot \ln\left(\frac{p1}{p2}\right) - \frac{p1}{p2} + 1 = 0, p2\right) | p1=124 \text{ and } pa=103.5$

p2=87.2805 or p2=124.

Example 2:

Consider a reactor where the following chemical reactions take place:



The rate equations are:

$$\frac{dC_A}{dt} = -k_1 C_A + k_2 C_B$$

$$\frac{dC_B}{dt} = k_1 C_A - (k_2 + k_3 + k_4) C_B$$

$$\frac{dC_C}{dt} = k_3 C_B$$

$$\frac{dC_D}{dt} = k_4 C_B$$

C_A , C_B , C_C , C_D are the concentrations of materials A, B, C, D respectively, and k_1 , k_2 , k_3 , k_4 are the rate constants. We know the concentrations at $t=0$ and the values of the rate constants:

$$C_{A0} = 50 \text{ g mol/l}$$

$$C_{B0} = 5 \text{ g mol/l}$$

$$C_{C0} = 0 \text{ g mol/l}$$

$$C_{D0} = 0 \text{ g mol/l}$$

$$k_1 = 2 \text{ hour}^{-1}$$

$$k_2 = 1 \text{ hour}^{-1}$$

$$k_3 = 0.2 \text{ hour}^{-1}$$

$$k_4 = 0.6 \text{ hour}^{-1}$$

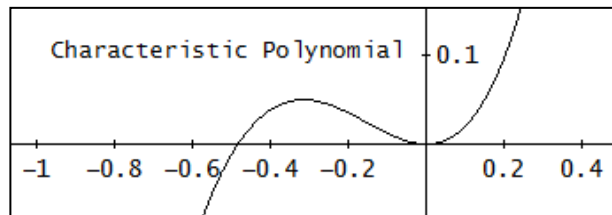
If we want to obtain the product B, what will be the reaction time?

We have a set of linear ordinary differential equations $\dot{Y} = AY$, for which the solution may be given by the matrix equation $Y = X e^{At} X^{-1} Y_0$, (X is a matrix where the columns are the eigenvectors of A , Λ is a diagonal matrix with the eigenvalues of A , Y_0 is the initial conditions vector and Y is a functional vector with the solution).

The students first enter matrix A and calculate the eigenvalues (#1 to #2). We get only three distinct eigenvalues and we must have four: a glance at the characteristic polynomial and its graph and we see that $w=0$ is a double eigenvalue.

Matriz A

$$\#1: A := \begin{bmatrix} -2 & 1 & 0 & 0 \\ 2 & -1.8 & 0 & 0 \\ 0 & 0.2 & 0 & 0 \\ 0 & 0.6 & 0 & 0 \end{bmatrix}$$



$$\#2: \text{EIGENVALUES}(A) = \left[0, \frac{\sqrt{201}}{10} - \frac{19}{10}, -\frac{\sqrt{201}}{10} - \frac{19}{10} \right]$$

$$\#3: \text{EIGENVALUES}(A) = [0, -0.482255, -3.31774]$$

$$\#4: \text{CHARPOLY}(A) = \frac{w^2 \cdot (5 \cdot w^2 + 19 \cdot w + 8)}{5}$$

The eigenvectors are obtained in #5 to #10 (we need different DERIVE commands because the eigenvalue $w=0$ is exact but $w = -3.31774$ and $w = -0.482255$ are approximate eigenvalues).

$$\#5: \text{EXACT_EIGENVECTOR}(A, 0)$$

$$\#6: \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ -1 & 0 \\ 0 & -1 \end{bmatrix}$$

$$\#7: \text{APPROX_EIGENVECTOR}(A, -0.482255)$$

$$\#8: [0.370995, 0.563076, -0.233518, -0.700554]$$

$$\#9: \text{APPROX_EIGENVECTOR}(A, -3.317744687)$$

$$\#10: [0.597661, -0.787566, 0.0474756, 0.142428]$$

Therefore the solution of the differential equations (with arbitrary values to $p1$ and $p2$) is:

$$\#11: X := \begin{bmatrix} 0 & 0 & 0.370995 & 0.597661 \\ 0 & 0 & 0.563076 & -0.787566 \\ -1 & 0 & -0.233518 & 0.0474756 \\ 0 & -1 & -0.700554 & 0.142428 \end{bmatrix}$$

Matriz $\exp(AT)$

$$\#12: L := \begin{bmatrix} \exp(0 \cdot t) & 0 & 0 & 0 \\ 0 & \exp(0 \cdot t) & 0 & 0 \\ 0 & 0 & \exp(-0.482255 \cdot t) & 0 \\ 0 & 0 & 0 & \exp(-3.31774 \cdot t) \end{bmatrix}$$

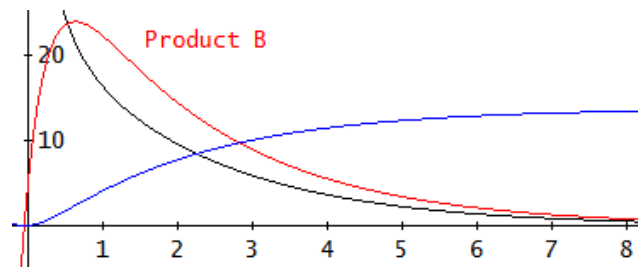
Vector condicoes iniciais

#13: [50, 5, 0, 0]

#14: $X \cdot L \cdot X^{-1} \cdot [50, 5, 0, 0]$

$$\#15: \begin{bmatrix} -0.482255 \cdot t + 25 \cdot e^{-0.482255 \cdot t} - 3.31773 \cdot t + 25 \cdot e^{-3.31773 \cdot t}, & 37.9436 \cdot e^{-0.482255 \cdot t} - 32.9436 \cdot e^{-3.31773 \cdot t}, & -15.7359 \cdot e^{-0.482255 \cdot t} + 1.98589 \cdot e^{-3.31773 \cdot t} + 13.75, & -47.2078 \cdot e^{-0.482255 \cdot t} + 5.95772 \cdot e^{-3.31773 \cdot t} + 41.2500 \end{bmatrix}$$

The graph of #15 shows the four functions $C_A=C_A(t)$, $C_B=C_B(t)$, $C_C=C_C(t)$ and $C_D=C_D(t)$ for $t \geq 0$. We can see that the concentration of product B has a maximum near $t = 0.6$ hour.



In lines #16 and #17 we get this value of t maximizing the function $C_B=C_B(t)$ (the second component of vector #15):

$$\#16: \text{SOLVE} \left(\frac{d}{dt} (37.9436 \cdot e^{-0.482255 \cdot t} - 32.9436 \cdot e^{-3.31773 \cdot t}), t \right)$$

#17: $t = 0.630320 - 2.21592 \cdot i \vee t = 0.630320 + 2.21592 \cdot i \vee t = 0.630320$

The answer is $t = 0.63$ hour (we don't verify the 2nd order conditions because the graph shows that C_B has effectively a maximum).

With DERIVE we can solve the same problem numerically, for example applying the fourth order Runge-Kutta method. We define the vectors r (differential equations in canonical form), v (variables) v_0 (initial values), step size and number of steps and the commands Simplify/Approximate give the numerical solution as a matrix (#19 to #21).

#19: $r := [-2 \cdot a + b, 2 \cdot a - 1.8 \cdot b, 0.2 \cdot b, 0.6 \cdot b]$

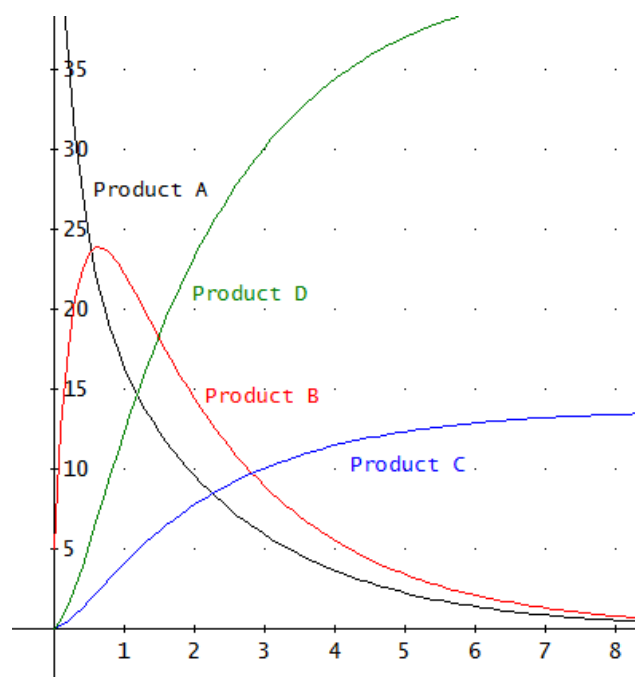
#20: $[v := [t, a, b, c, d], v_0 := [0, 50, 5, 0, 0]]$

#21: $\text{RKso1} := \text{RK}(r, v, v_0, 0.1, 120)$

$$\begin{bmatrix} 0 & 50 & 5 & 0 & 0 \\ 0.1 & 41.7649 & 12.5141 & 0.180205 & 0.540617 \\ 0.2 & 35.5780 & 17.4867 & 0.483815 & 1.45144 \\ 0.3 & 30.8738 & 20.6549 & 0.867784 & 2.60335 \\ 0.4 & 27.2463 & 22.5471 & 1.30160 & 3.90482 \\ 0.5 & 24.4034 & 23.5415 & 1.76375 & 5.29127 \end{bmatrix}$$

.....

From matrix #21 (121 rows) we can extract pairs of columns (EXTRACT_2_COLUMNS or using $\downarrow\downarrow$) to study the graphs of the different functions in the solution $C_A=C_A(t)$, $C_B=C_B(t)$, $C_C=C_C(t)$ and $C_D=C_D(t)$ or to find the trajectories in the phase space.



Frequently in chemical engineering we need to solve stiff problems where numerical methods such as Runge-Kutta are inadequate. Frequently the matrix equation $Y=Xe^{At}X^{-1}Y_0$ allows us to solve these problems without the need of special methods such as the finite differences Gear's method.

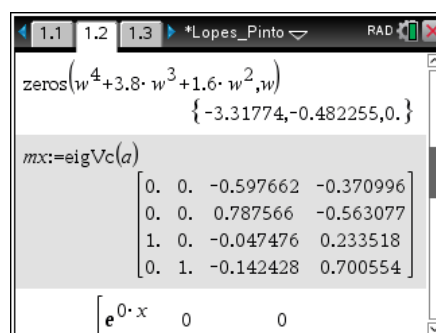
The matrix equation $Y=Xe^{At}X^{-1}Y_0$ is obtained from calculus but usually we need some help from numerical analysis to determine the eigenvalues of matrix A . It is an equation which is not very popular between the old generations of students because it implies a lot of hand calculations. With DERIVE the calculations are immediate.

References:

- Constantinides, A., "Applied Numerical Methods with Personal Computers", NY, McGraw-Hill International Edition, 1989
- Edgar, T.F., Himmelblau, D.M., "Optimization of Chemical Process", NY, McGraw-Hill International Edition, 1989
- Townend, M.S., Pountney, D.C. "Computer-Aided Engineering Mathematics", Chichester, Ellis Horwood, 1989

We can perform the same procedure with TI-NspireCAS – with one advantage: as you can see on the screen shot, we will get the Eigenvectors (matrix X) in one single step.

On the next page I will solve the DE-system using a tool provided by Michel Beaudin's great Nspire-library `ets_specfunc.tns` (see DNL102).



ets_funcspec.tns is a file in my MyLib-folder where all my favourite libraries are stored. You can easily follow the syntax of the function *simultd*():

©Example 2

$$mat := \begin{bmatrix} \frac{d}{dx}(ca(x)) = -2 \cdot ca(x) + cb(x) \\ \frac{d}{dx}(cb(x)) = 2 \cdot ca(x) - (1 + 0.2 + 0.6) \cdot cb(x) \\ \frac{d}{dx}(cc(x)) = 0.2 \cdot cb(x) \\ \frac{d}{dx}(cd(x)) = 0.6 \cdot cb(x) \end{bmatrix}$$

$$ini := \begin{bmatrix} ca(x) & 50 \\ cb(x) & 5 \\ cc(x) & 0 \\ cd(x) & 0 \end{bmatrix}$$

$$ets_specfunc \backslash simultd(mat, ini)$$

$$ca(x) = \frac{25 \cdot \left(e^{\sqrt{67} \cdot \sqrt{3} \cdot x} \right)^{\frac{1}{10}}}{\left(e^x \right)^{\frac{19}{10}}} + \frac{25}{\left(e^x \right)^{\frac{19}{10}} \cdot \left(e^{\sqrt{67} \cdot \sqrt{3} \cdot x} \right)^{\frac{1}{10}}}$$

As the solution functions are pretty bulky in exact mode, I approximate and transfer them into the Graphs-application.

$$cc(x) = \frac{\frac{8}{(e^x)^{\frac{19}{10}}} + \frac{8}{(e^x)^{\frac{19}{10}} \cdot (e^{\sqrt{67} \cdot \sqrt{3} \cdot x})^{\frac{1}{10}}} + \frac{55}{4}}{\frac{19}{(e^x)^{\frac{19}{10}}} + \frac{19}{(e^x)^{\frac{19}{10}} \cdot (e^{\sqrt{67} \cdot \sqrt{3} \cdot x})^{\frac{1}{10}}} + \frac{1}{4}}$$

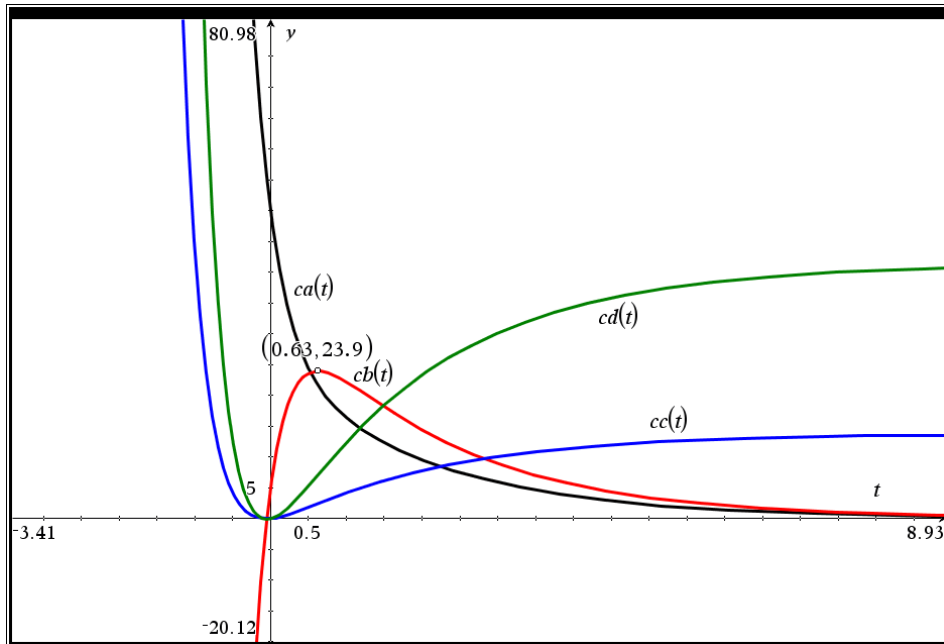
$$cd(x) = \frac{\left(\frac{-15 \cdot \sqrt{67} \cdot \sqrt{3}}{8} - \frac{165}{8} \right) \cdot (e^{\sqrt{67} \cdot \sqrt{3} \cdot x})^{\frac{1}{10}}}{\frac{19}{(e^x)^{\frac{19}{10}}} + \frac{19}{(e^x)^{\frac{19}{10}} \cdot (e^{\sqrt{67} \cdot \sqrt{3} \cdot x})^{\frac{1}{10}}} + \frac{1}{4}} + \frac{\frac{15 \cdot \sqrt{67} \cdot \sqrt{3}}{8} - \frac{165}{8}}{\frac{19}{(e^x)^{\frac{19}{10}}} + \frac{19}{(e^x)^{\frac{19}{10}} \cdot (e^{\sqrt{67} \cdot \sqrt{3} \cdot x})^{\frac{1}{10}}} + \frac{1}{4}} + \frac{165}{4}$$

$$\begin{bmatrix} ca(x) = 25 \cdot (0.617389)^x + 25 \cdot (0.036234)^x \\ cb(x) = 37.9436 \cdot (0.617389)^x - 32.9436 \cdot (0.036234)^x \\ cc(x) = -15.7359 \cdot (0.617389)^x + 1.9859 \cdot (0.036234)^x + 13.75 \\ cd(x) = -47.2077 \cdot (0.617389)^x + 5.95771 \cdot (0.036234)^x + 41.25 \end{bmatrix}$$

$f1(x) := 25 \cdot (0.617389)^x + 25 \cdot (0.036234)^x$ $f2(x) := 37.9436 \cdot (0.617389)^x - 32.9436 \cdot (0.036234)^x$ Done

$f3(x) := -15.7359 \cdot (0.617389)^x + 1.9859 \cdot (0.036234)^x + 13.75$ $f4(x) := -47.2077 \cdot (0.617389)^x + 5.95771 \cdot (0.036234)^x + 41.25$ Done

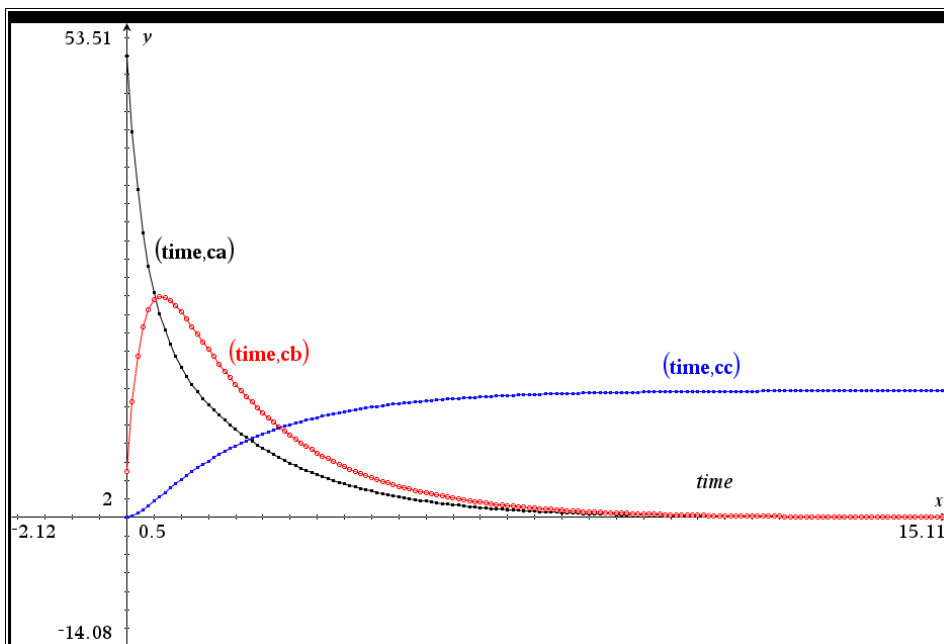
The function graphs with the maximum point of $cb(t)$ – without using Calculus but Nspire's Analyze Graph Tool. Then you can see how to solve the system using the Runge-Kutta numerical method.



```
rk1:=rk23 $\left\{ \begin{array}{l} -2 \cdot ca + cb \\ 2 \cdot ca - 1.8 \cdot cb \\ 0.2 \cdot cb \\ 0.6 \cdot cb \end{array} \right.$ , t, {ca, cb, cc, cd}, {0, 20}, {50, 5, 0, 0}, 0.1
```

	0.	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.
50.	41.7629	35.5744	30.8695	27.2399	24.3944	22.1232	20.2717	18.7388	17.4365	16.3178	
5.	12.517	17.4914	20.6608	22.5557	23.5534	23.9238	23.8647	23.5034	22.9503	22.2619	
0.	0.180037	0.483534	0.867438	1.3011	1.76305	2.23824	2.7159	3.18946	3.65332	4.10506	
0.	0.540112	1.4506	2.60231	3.90329	5.28915	6.71471	8.1477	9.56837	10.96	12.3152	

```
time:=mat▶list(rk1[1]):ca:=mat▶list(rk1[2]):cb:=mat▶list(rk1[3]):cc:=mat▶list(rk1[4])
{0., 0.180037, 0.483534, 0.867438, 1.3011, 1.76305, 2.23824, 2.7159, 3.18946, 3.65332, 4.10506, 4.54158, 4.96306}
```

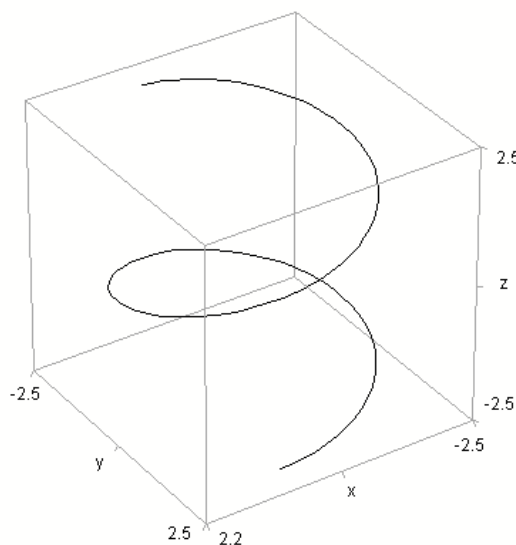
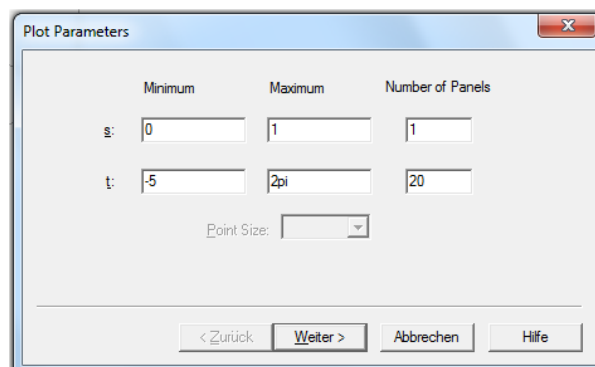


Plotting 3-Dimensional Curves with SPACE_TUBE(,,)

by Steven Schonefeld

Suppose we wish to perform a 3-D plot of the helix #1 for the parameter s ranging from -5 to 5 . This is easy to do using DERIVE 5 or 6. You simply highlight #1, click on the 3-D plot icon, tap function key F4 in order to set the Plot Parameters (only for parameter s), and click on the 3-D plot icon in the graphics window. The settings and the plot are shown below. Since #1 contains only one parameter t , the settings for parameter s make no difference ($0 \leq s \leq 1$ recommended).

$$\#1: \left[2 \cdot \cos(t), 2 \cdot \sin(t), \frac{t}{2} \right]$$



As the reader may know, the helix #1 is part of the cylinder having radius 2 with center along the z -axis. Such a cylinder is given parametrically by #2. It would be nice to illustrate the connection between the helix and the cylinder by plotting both #1 and #2 in the same graphics window. However, when we perform this plot, the helix nearly disappears into the cylinder. This result is less than desirable. One way to make the helix stand out is to change the plot color and slightly shrink the size of the cylinder.

Perhaps a better alternative is to use the function $\text{SPACE_TUBE}(v, s, r, t)$ contained in the DERIVE utility file Graphics.mth (see comment next page). The arguments for this function are: v = a vector representation for the desired curve (such as #1), s = the parameter used in v , r = the radius for the space tube (usually a small, positive number), t = a second parameter. The function $\text{SPACE_TUBE}(\ , \ , \ , \)$ creates a surface composed of circles having radius r which enclose the given curve. It adds thickness to the curve. We illustrate the use of this function below. After loading the file Graphics.mth, we author #4 and simplify to get #5.

The Plot Properties for the cylinder #2 are: s ranges from -2.5 to 2.5 with 20 panels; t ranges from 0 to 2π with 40 panels. For the space tube #5, the Plot Properties are: s from -5 to 5 with 40 panels; t from 0 to 2π with 10 panels. The results of these plots are shown below #5. In this graph it is clear that the helix is part of the cylinder, yet we can clearly see the path of the helix. With DERIVE 5 and 6, we may easily view this figure from any possible angle with a few clicks of our mouse. Notice that we can see the path of the helix both inside and outside the cylinder. On top of all that, we may put the figure in motion (it rotates about the z -axis.) This truly brings 3-Dimensional figures to life!

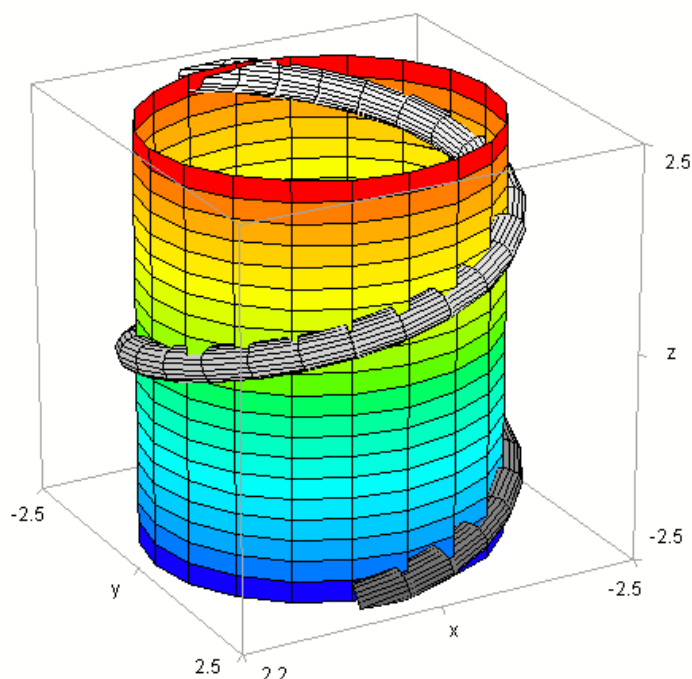
#2: $[2 \cdot \cos(t), 2 \cdot \sin(t), s]$

#3: `LOAD(C:\Program Files (x86)\DfW5\Math\Graphics.mth)`

#4: $\text{SPACE_TUBE}\left(\left[2 \cdot \cos(s), 2 \cdot \sin(s), \frac{s}{2}\right], s, \frac{1}{5}, t\right)$

#5:
$$\left[\cos(s) \cdot \left(2 - \frac{\sin(t)}{5}\right) + \frac{\sqrt{17} \cdot \sin(s) \cdot \cos(t)}{85}, \sin(s) \cdot \left(2 - \frac{\sin(t)}{5}\right) - \frac{\sqrt{17} \cdot \cos(s) \cdot \cos(t)}{85}, \frac{4 \cdot \sqrt{17} \cdot \cos(t)}{85} + \frac{s}{2} \right]$$

Comment: Graphics.mth is not contained in the DERIVE 6 utility files. You can find the function `space_tube()` at the end of this article, Josef.



Next example:

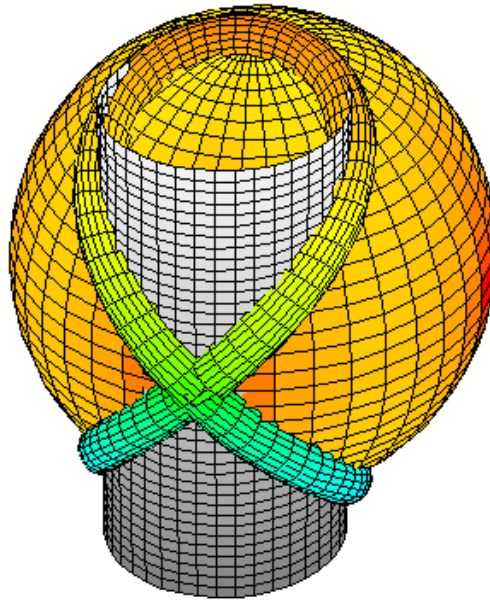
The curve #8 (t from 0 to 4π) is contained in the intersection of the cylinder #6 and the sphere #7. For the cylinder #6: s from -2 to 2 with 40 panels; t from 0 to 2π with 40 panels. For the sphere #7: s from 0 to 2π with 40 panels; t from 0 to π with 40 panels. For the space tube #9: t from 0 to 4π with 100 panels; s from 0 to 2π with 20 panels. The results of this plot are shown below #9. It is not necessary to simplify #9. (Except you want to see the bulky expression.)

$$\#6: [1 + \cos(t), \sin(t), s]$$

$$\#7: [2 \cdot \cos(s) \cdot \sin(t), 2 \cdot \sin(s) \cdot \sin(t), 2 \cdot \cos(t)]$$

$$\#8: \left[1 + \cos(t), \sin(t), 2 \cdot \sin\left(\frac{t}{2}\right) \right]$$

$$\#9: \text{SPACE_TUBE}\left(\left[1 + \cos(t), \sin(t), 2 \cdot \sin\left(\frac{t}{2}\right) \right], t, \frac{1}{5}, s\right)$$



Comment: It might be of interest how to derive equation #8 of the space curve – it is the “Window of Viviani”? It will be presented in an appendix. Josef.

The vector expressions #10 through #16 represent 3-D curves, which are contained in one or more surfaces. The interested reader might like to determine the surface or surfaces for each curve and plot the surface(s) together with the space tube for the curve. For #12, let s range from -2 to 2. For #13, let s range from 0 to 3. For #16, let s range from 0 to 4π . For all others, let s range from 0 to 2π .

$$\#10: \left[\cos(t) + 1, \sin(t), 2 \cdot \sin\left(\frac{t}{2}\right) \right]$$

$$\#11: [2 \cdot \cos(s), 2 \cdot \sin(s), 2 \cdot \cos(s)]$$

$$\#12: [2 \cdot \cos(s), 2 \cdot \sin(s), 4 \cdot \cos(s)^2]$$

$$\#13: [s \cdot \cos(s), s \cdot \sin(s), s]$$

$$\#14: [s \cdot \cos(s), s \cdot \sin(s), s^2]$$

$$\#15: [2 \cdot \cos(s) \cdot \sin(4 \cdot s), 2 \cdot \sin(s) \cdot \sin(4 \cdot s), 2 \cdot \cos(4 \cdot s)]$$

$$\#16: [2 \cdot \cos(3 \cdot s) \cdot \sin(s), 2 \cdot \sin(3 \cdot s) \cdot \sin(s), 2 \cdot \cos(s)]$$

$$\#17: \left[\left(3 + \cos\left(\frac{5 \cdot s}{2}\right) \right) \cdot \cos(s), \left(3 + \cos\left(\frac{5 \cdot s}{2}\right) \right) \cdot \sin(s), \sin\left(\frac{5 \cdot s}{2}\right) \right]$$

Here are some remarks concerning the function `SPACE_TUBE(, ,)`.

(1) This function has been available in earlier versions of DERIVE. However, "plotting" the resulting surface could only be done with the 2-D isometric projections (also contained in Graphics.mth). A colleague of mine, Richard Ruselink, has been using this method of displaying space curves for years. This isometric plotting is very cumbersome.

(2) Suppose we make s the variable of the vector representation for our curve and t the other parameter in `SPACE_TUBE(, ,)` -- as we have done above. We must use a large number of panels for s , to give the tube a nice appearance. As the curve gets longer, the number of panels for s should increase. The parameter t must range over an interval of length 2π to get the complete space tube. Since the circular cross-sections have a small radius, we only need a few panels (above we used 10)

for t . It is interesting to experiment with an even smaller number of panels. If parameter t is plotted using 4 panels, the cross-sections of the space tube are squares. Using only 3 panels for t , these cross-sections are equilateral triangles.

(3) The definition of `SPACE_TUBE(, ,)` is shown as expressions #18, #19, and #20 below. At first glance, it appears that the auxiliary functions `NORMAL_VECTOR(,)` and `BINORMAL(,)` are the usual unit normal and unit binormal vectors, which we explain to our students in calculus class. As Professor Ruselink told me, this is only true if the parameter s is arc length or a constant multiple of arc length. This may not always be the case. The curve #21 is a re-parametrization of #1 obtained by replacing s by s^3 . This should give the same helix as #1. However, as we can see by the graph below #21, the space tube for #21 appears to be pinched near the point where $s = 0$.

The lines #22 -- #25 give a definition of `MY_SPACE_TUBE(, ,)`, which uses the true unit normal and unit binormal vectors for our curve. These two unit vectors are used as a coordinate system to create a circle perpendicular to the curve. That is, for each value of parameter s , we get a normal circle of radius r with center on the curve. In the figure below #25, we show the performance of `MY_SPACE_TUBE(, ,)` in displaying the helix #21. The resulting space tube is clearly better.

Now, for the bad news. It turns out that `MY_SPACE_TUBE(, ,)` may take a lot more computer time than `SPACE_TUBE(, ,)` to simplify. Also, the expression we get from simplifying `MY_SPACE_TUBE(, ,)` may be much more complicated than the expression we get from simplifying `SPACE_TUBE(, ,)`. However, if you are patient, the resulting space tube is a better representation for the curve when you use `MY_SPACE_TUBE(, ,)`.

(4) If the unit normal vector (or the second derivative) for a curve is always the zero vector (for example, if our "curve" is a straight line), then the functions `MY_SPACE_TUBE(, ,)` and `SPACE_TUBE(, ,)` fail to define the desired surface. In this situation, you would need to manually find two perpendicular vectors to replace `NORMAL_VECTOR(,)` and `BINORMAL(,)` in the definition of `SPACE_TUBE(, ,)`.

In case the unit normal is the zero vector at an isolated value of the parameter s , there may be a gap in the space tube corresponding to this value of s . By changing the number of panels for s , we may get the 3-D plot to skip over this troublesome point in the space tube.

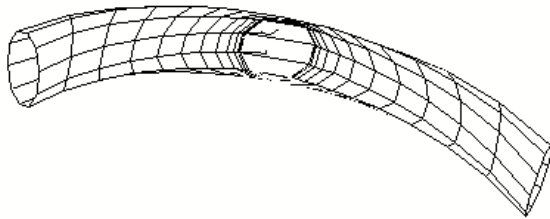
(5) Since the true unit normal vector is used in the definition of $MY_SPACE_TUBE(, , ,)$, the resulting space tube will "twist" as the unit normal winds around the given curve. This can enhance our understanding of the curve. The reader may wish to use $MY_SPACE_TUBE(, , ,)$ to plot the curves #11 -- #17 and compare these results with those obtained using $SPACE_TUBE(, , ,)$.

$$\#18: \text{NORMAL_VECTOR}(v, t) := \text{SIGN} \left(\left(\frac{d}{dt} \right)^2 v \right)$$

$$\#19: \text{BINORMAL}(v, t) := \text{SIGN} \left(\text{CROSS} \left(\frac{d}{dt} v, \left(\frac{d}{dt} \right)^2 v \right) \right)$$

$$\#20: \text{SPACE_TUBE}(v, t, r, \phi) := v + r \cdot (\text{SIN}(\phi) \cdot \text{NORMAL_VECTOR}(v, t) + \text{COS}(\phi) \cdot \text{BINORMAL}(v, t))$$

$$\#21: \text{SPACE_TUBE} \left(\left[2 \cdot \text{COS}(s)^3, 2 \cdot \text{SIN}(s)^3, \frac{s^3}{2} \right], s, \frac{1}{4}, t \right)$$



$-1 \leq t \leq 1$, odd number of panels

$0 \leq s \leq \pi$

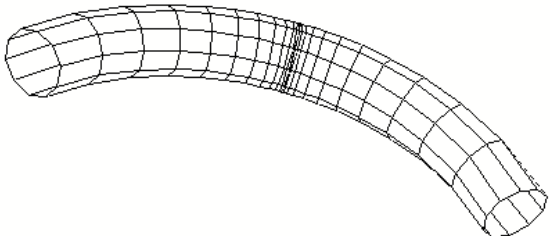
$$\#22: \text{unit_tangent}(v, t) := \text{SIGN} \left(\frac{d}{dt} v \right)$$

$$\#23: \text{unit_normal}(v, t) := \text{SIGN} \left(\frac{d}{dt} \text{unit_tangent}(v, t) \right)$$

$$\#24: \text{unit_binormal}(v, t) := \text{CROSS}(\text{unit_tangent}(v, t), \text{unit_normal}(v, t))$$

$$\#25: \text{my_space_tube}(v, t, r, \phi) := v + r \cdot (\text{SIN}(\phi) \cdot \text{unit_normal}(v, t) + \text{COS}(\phi) \cdot \text{unit_binormal}(v, t))$$

$$\#26: \text{my_space_tube} \left(\left[2 \cdot \text{COS}(s)^3, 2 \cdot \text{SIN}(s)^3, \frac{s^3}{2} \right], s, \frac{1}{4}, t \right)$$



$-1 \leq t \leq 1$, odd or even number of panels

$0 \leq s \leq \pi$

Appendix 1:

Calculation of the intersection curve of the cylinder and the sphere to obtain the "Window of Viviani".

We take the implicit form of the sphere $x^2 + y^2 + z^2 = 2^2$ and replace x , y and z by the components of the parameter form of the cylinder. Then we solve the resulting equation for s .

```

#27: [1 + COS(t), SIN(t), s]

#28: (1 + COS(t))^2 + SIN(t)^2 + s^2 = 4

#29: 2 * COS(t) + s^2 + 2 = 4

#30: SOLUTIONS(2 * COS(t) + s^2 + 2 = 4, s)

#31: [Trigonometry := Collect, Trigpower := Sines]

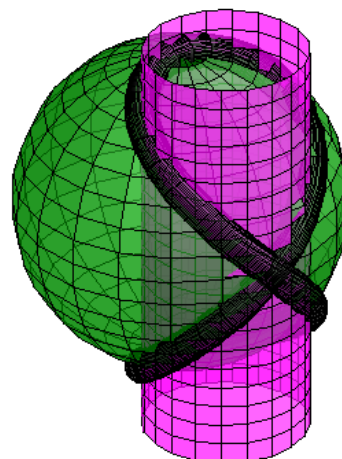
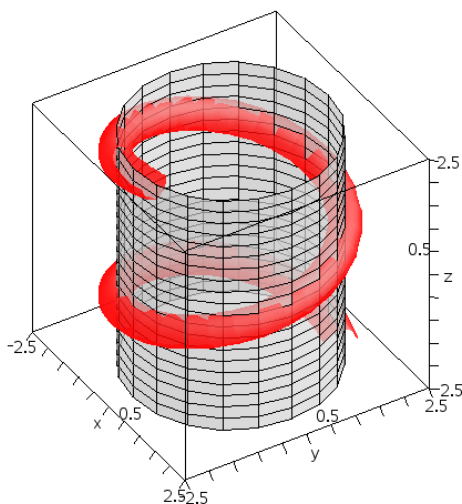
#32: [2 * |SIN(t/2)|, -2 * |SIN(t/2)|]

#33: [1 + COS(t), SIN(t), 2 * SIN(t/2)]

```

Appendix 2: Space tubes with TI-NspireCAS (pretty nice!)

$$\begin{aligned}
 \text{unit_t}(v,t) &:= \frac{\frac{d}{dt}(v)}{\text{norm}\left(\frac{d}{dt}(v)\right)} \quad \text{Done} & \text{unit_n}(v,t) &:= \frac{\frac{d^2}{dt^2}(v)}{\text{norm}\left(\frac{d^2}{dt^2}(v)\right)} \quad \text{Done} \\
 \text{unit_b}(v,t) &:= \text{crossP}(\text{unit_t}(v,t), \text{unit_n}(v,t)) \quad \text{Done} \\
 \text{m_sp_t}(v,r,t,u) &:= \text{mat} \rightarrow \text{list}(v + r \cdot (\sin(u) \cdot \text{unit_n}(v,t) + \cos(u) \cdot \text{unit_b}(v,t))) \quad \text{Done} \\
 \text{hel} &:= \left[2 \cdot \cos(t) \quad 2 \cdot \sin(t) \quad \frac{t}{2} \right] \rightarrow \left[2 \cdot \cos(t) \quad 2 \cdot \sin(t) \quad \frac{t}{2} \right] \\
 \text{hel_t} &:= \text{m_sp_t}(\text{hel}, 3/10, t, u) \\
 &\rightarrow \left\{ \cos(t) \cdot \left(2 - \frac{3 \cdot \sin(u)}{10} \right) + \frac{3 \cdot \sqrt{17} \cdot \sin(t) \cdot \cos(u)}{170}, \sin(t) \cdot \left(2 - \frac{3 \cdot \sin(u)}{10} \right) - \frac{3 \cdot \sqrt{17} \cdot \cos(t) \cdot \cos(u)}{170}, \frac{t}{2} + \frac{6 \cdot \sqrt{17}}{2} \right\} \\
 \text{viv_t} &:= \text{m_sp_t} \left(\left[1 + \cos(t) \quad \sin(t) \quad 2 \cdot \sin\left(\frac{t}{2}\right) \right], \frac{1}{5}, t, u \right) \\
 &\rightarrow \left\{ \frac{\left(2 \cdot \sin(t) \cdot \cos\left(\frac{t}{2}\right) - \cos(t) \cdot \sin\left(\frac{t}{2}\right) \right) \cdot \cos(u)}{5 \cdot \sqrt{\left(\sin\left(\frac{t}{2}\right)\right)^2 + 4}} - \frac{2 \cdot \cos(t) \cdot \sin(u)}{5 \cdot \sqrt{\left(\sin\left(\frac{t}{2}\right)\right)^2 + 4}} + \cos(t) + 1, \frac{\sin(t) \cdot \left(5 \cdot \sqrt{\left(\sin\left(\frac{t}{2}\right)\right)^2 + 4} - 2 \right)}{5 \cdot \sqrt{\left(\sin\left(\frac{t}{2}\right)\right)^2 + 4}} \right\}
 \end{aligned}$$



Propositional Multivalued Logics with the TI-92 and TI-89 Symbolic Calculators

Eugenio Roanes-Lozano*

Dept. Algebra, Univ. Complutense de Madrid
eroanes@eucmos.sim.ucm.es

Abstract: This little package deals with Kleene's style (min/max) p -valued logic with modal connectives (for any given prime number p). It can build the truth table corresponding to two given propositions, check if a given proposition is a tautology and check if the second of two given propositions is a tautological consequence of the first one.

Introduction

A very interesting approach to multivalued modal logics using Gröbner bases exists [7, 6, 3]. We have developed models (implemented in Maple and CoCoA) to study inference in propositional algebras and consistency of Knowledge Based Systems (KBSs) based on Boolean and multivalued modal logics [8, 9, 10, 11], as well as applications [12, 13], that follow that line. But this approach, although very powerful, involves the calculation of Gröbner bases [4] of polynomial ideals over fields of finite characteristic, what is not intuitive.

It takes a long time to produce truth tables by hand, but many intuitive ideas come from working with them. Moreover, an elementary approach to multivalued logics, based on the use of truth tables, can be easily implemented in a Computer Algebra System (CAS). Therefore to have a mechanical truth table builder and tautologies and tautological checker could be very convenient.

We presented at the 1st International Derive Conference (Plymouth, 1994) an approach to the Boolean case in DERIVE 2 [5]. But, unfortunately for us, the new release, DERIVE 3, presented at the same conference, included a built-in function for the same purpose. An obvious extension is to construct truth tables in multivalued logics. We have done so in Maple [14], Macsyma (this implementation is included in version 2.3) and DERIVE [15] (this implementation is included in DERIVE 5). In this paper the implementation is ported to the language of the symbolic calculators TI-92 and TI-89 [1, 2].

Using a CAS or a symbolic calculator has the advantage to use the built-in matrix operations and exact arithmetic provided (for using rational numbers to represent truth values when the number of truth values is greater than two).

1 Classic Propositional Bivalued Logic

1.0.1 Definition.- *The classic propositional bivalued logic is $(\mathcal{C}, \vee, \wedge, \neg)$, where \mathcal{C} is the set of propositions; \vee, \wedge are the (basic) logical binary connectives "and" and "or" and \neg is the logical unary connective "negation".*

*Partially supported by project DGES PB96-0098-C04-03 (Spain). This paper is based on a demonstration of the capabilities of the TI-92 & 89 performed at the "Fifth International Conference AISC'2000" (Artificial Intelligence and Symbolic Computation), Madrid (Spain), July 17th-19th 2000.

In a constructive way, if the propositional variables are X_1, X_2, \dots, X_m then \mathcal{C} is the set of well-constructed formulae using \vee, \wedge, \neg and X_1, X_2, \dots, X_m .

Logical binary connectives can be given as a truth table (i.e., as a mapping $\{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\}$)

P	Q	$P \vee Q$	$P \wedge Q$	$P \rightarrow Q$	$P \leftrightarrow Q$
0	0	0	0	1	1
0	1	1	0	1	0
1	0	1	0	0	0
1	1	1	1	1	1

(where P, Q are any propositions, 0 represents *False* and 1 represents *True*).

The mapping can also be given directly in a functional way; e.g. P and Q can be valued in $\{0, 1\}$ and

$$\begin{aligned} P \vee Q &= \max(P, Q) \\ P \wedge Q &= \min(P, Q) \end{aligned}$$

or in polynomial form

$$\begin{aligned} P \vee Q &= P + Q - P \cdot Q \\ P \wedge Q &= P \cdot Q \end{aligned}$$

The unary connective $\neg P$ can be given as a truth table

P	$\neg P$
1	0
0	1

(for any proposition P) or in a functional or polynomial way; e.g. P can be valued in $\{0, 1\}$ and

$$\neg P = 1 - P.$$

1.0.2 Remark.- Observe that, although connectives are used to build \mathcal{C} , they are usually identified with functions $\mathcal{C} \times \mathcal{C} \rightarrow \mathcal{C}$ (if binary) or $\mathcal{C} \rightarrow \mathcal{C}$ (if unary).

1.0.3 Definition.- Once \neg has been introduced, \rightarrow and \leftrightarrow can also be defined as follows

$$\begin{aligned} P \rightarrow Q &\text{ iff } \neg P \vee Q \\ P \leftrightarrow Q &\text{ iff } (P \rightarrow Q) \wedge (Q \rightarrow P) \end{aligned}$$

1.0.4 Remark.- Observe that, in the introduction above, the sentence “are valued in $\{0, 1\}$ ” is used, what is a bit vague. Moreover, the introduction above used an abuse in the notation as connectives were applied both to propositions and truth values. Formal definitions that distinguish between a function \tilde{F} (applied to numbers) and the logical connective F (applied to propositions), that uses valuations for propositional variables that are extended to valuations of propositions, can be given [11], but will be skipped here for the sake of brevity.

1.0.5 Definition.- Q is a tautological consequence of P , denoted $P \models Q$, iff whenever P is True, then Q is also True.

1.0.6 Example.- Prove that $P \wedge \neg Q \models P \vee \neg Q$

P	Q	$\neg Q$	$P \wedge \neg Q$	$P \vee \neg Q$
0	0	1	0	1
1	0	1	1	1
0	1	0	0	0
1	1	0	0	1

1.0.7 Definition.- P is a tautology, denoted $\models P$, iff P is always True.

1.0.8 Example.- Prove that $\models ((P \wedge \neg Q) \rightarrow (P \vee \neg Q))$

P	Q	$\neg Q$	$P \wedge \neg Q$	$\neg(P \wedge \neg Q)$	$P \vee \neg Q$	$(P \wedge \neg Q) \rightarrow (P \vee \neg Q)$
0	0	1	0	1	1	1
1	0	1	1	0	1	1
0	1	0	0	1	0	1
1	1	0	0	1	1	1

1.0.9 Theorem.- It can be proven that, in classic bivalued logic, for any propositions P and Q ,

$$\models (P \rightarrow Q) \text{ iff } P \models Q .$$

2 Multivalued Logics

An introduction to multivalued logics can be found in [16].

2.1 Intuitive Presentation of Kleene's Three-Valued Logics

In this logic the three truth values considered are: "True", "False" and "Undecided", that we shall represent by 1, 0 and $\frac{1}{2}$ (respectively). With this numeric representation, the greater the value the greater the certainty, what is very intuitive.

Apart from negation, two more unary connectives (modal connectives) are introduced: "necessary" (\Box) and "possible" (\Diamond):

P	$\neg P$	$\Box P$	$\Diamond P$
0	1	0	0
$\frac{1}{2}$	$\frac{1}{2}$	0	1
1	0	1	1

Binary connectives disjunction and conjunction correspond to "maximum" and "minimum" functions (respectively). Polynomial forms can be given, but they depend on the number of truth values of the logic. Conditional and biconditional are defined like in classic bivalued logic

$$P \rightarrow Q \text{ iff } \neg P \vee Q$$

$$P \leftrightarrow Q \text{ iff } (P \rightarrow Q) \wedge (Q \rightarrow P) .$$

Therefore, the corresponding truth tables are

P	Q	$P \vee Q$	$P \wedge Q$	$P \rightarrow Q$	$P \leftrightarrow Q$
0	0	0	0	1	1
$\frac{1}{2}$	0	$\frac{1}{2}$	0	$\frac{1}{2}$	$\frac{1}{2}$
1	0	1	0	0	0
0	$\frac{1}{2}$	$\frac{1}{2}$	0	1	$\frac{1}{2}$
$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
1	$\frac{1}{2}$	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
0	1	1	0	1	0
$\frac{1}{2}$	1	1	$\frac{1}{2}$	1	$\frac{1}{2}$
1	1	1	1	1	1

2.2 Intuitive Presentation of Lukasiewicz's Three-Valued Logic

In this logic three truth values considered are: "True", "False" and "Indeterminate" (that will be represented by 1, 0 and $\frac{1}{2}$, respectively). Kleene's connectives differs from Lukasiewicz's only in the conditional and biconditional (compare the fifth and sixth columns above with the third and fourth columns below)

P	Q	$P \rightarrow_{Lu} Q$	$P \leftrightarrow_{Lu} Q$
0	0	1	1
$\frac{1}{2}$	0	$\frac{1}{2}$	$\frac{1}{2}$
1	0	0	0
0	$\frac{1}{2}$	1	$\frac{1}{2}$
$\frac{1}{2}$	$\frac{1}{2}$	1	1
1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
0	1	1	0
$\frac{1}{2}$	1	1	$\frac{1}{2}$
1	1	1	1

A polynomial form of Lukasiewicz's $P \rightarrow_{Lu} Q$ is

$$4P^2Q^2 - 4P^2Q - 4PQ^2 + 5PQ - P + 1 .$$

From the truth table it is clear that

$$P \leftrightarrow_{Lu} Q = (P \rightarrow_{Lu} Q) \wedge (Q \rightarrow_{Lu} P) .$$

2.3 Multivalued Propositional Logics (Kleene's-style)

Let p be a prime number.

2.3.1 Definition.- A p -valued propositional logic is $(\mathcal{C}, F_1, \dots, F_n)$, where \mathcal{C} is the set of propositions and F_1, \dots, F_n are the logical connectives (unary or binary).

If the propositional variables are X_1, X_2, \dots, X_m , then \mathcal{C} is the set of well-constructed formulas using F_1, \dots, F_n and X_1, X_2, \dots, X_m .

We shall consider the truth values to be $\{0, \frac{1}{p-1}, \frac{2}{p-1}, \dots, \frac{p-2}{p-1}, 1\}$ for 0 representing False, 1 representing True and intermediate values representing intermediate degrees of certainty.

2.3.2 Definition.- According to the numerical representation of the truth values, we shall define

$$\begin{aligned} P \vee Q &= \max(P, Q) \\ P \wedge Q &= \min(P, Q) \\ \neg P &= 1 - P . \end{aligned}$$

2.3.3 Remark.- \neg tries to translate the idea of reversing the "probability" of something to happen.

2.3.4 Definition.- Two modal connectives, necessary (\Box) and possible (\Diamond) are defined. $\Box P$ is valued 0, unless P is True, when it is valued as 1. Meanwhile $\Diamond P$ is valued 1, unless P is False, when it is valued as 0. Therefore, they can be expressed in a functional way as follows

$$\begin{aligned} \Box P &= \text{floor}(P) \\ \Diamond P &= 1 - \text{floor}(1 - P) . \end{aligned}$$

2.3.5 Definition.- “To be a tautological consequence” and “to be a tautology” are defined (in a Kleene’s-style logic) exactly as in Boolean logic (see definitions 1.0.5 and 1.0.7).

2.3.6 Theorem.- In the general case of p -valued logics ($p > 2$), it can only be proven that, for any propositions P and Q ,

$$\models (P \rightarrow Q) \Rightarrow P \models Q .$$

Adding certain conditions it can be proven that [11],

$$\models (\Box P \rightarrow Q) \Leftrightarrow P \models Q .$$

Therefore, the behaviour is no longer intuitive.

2.3.7 Example.- Let $p = 3$ and let P be a general proposition. Trivially $P \models P$ but $\models (P \rightarrow P)$ doesn’t hold ($P \vee \neg P$ is not always true)

P	$\neg P$	$P \vee \neg P$
0	1	1
$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
1	0	1

2.3.8 Remark.- Lukasiewicz’s logic is not intuitive either. For instance $\models ((P \vee \neg P) \leftrightarrow_{Lu} t)$ doesn’t hold.

3 Implementation in the TI-92 & TI-89 Symbolic Calculators

The symbolic TIs include a CAS (in the “Home” application), that is similar to Derive for DOS.

3.1 Constructing the Logic Operators

The global variable w is used to store the number of truth values, e.g.:

```
3 STO w
```

This is the first value the user has to fix.

Connectives can be defined as functions. They begin with an “M” (standing for “multivalued”), not to interfere with the built-in Boolean connectives.

```
Define MNEG(a_) = 1-a_
Define MPOS(a_) = 1-FLOOR(1-a_)
Define MNEC(a_) = FLOOR(a_)
Define MOR(a_,b_) = MAX(a_,b_)
Define MAND(a_,b_) = MIN(a_,b_)
```

And conditional and biconditional (Kleene’s-style) can be defined using the previous connectives

```
Define MIMP(a_,b_) = MOR( MNEG(a_) , b_ )
Define MIFF(a_,b_) = MAND( MIMP(a_,b_) , MIMP(b_,a_) )
```

Tautology and contradiction are denoted t and c (respectively) and are assigned to constants 1 and 0 (respectively).

```
1 STO t
0 STO c
```


3.2 Constructing Truth Tables

The local variable m will be used within the procedures to specify the number of propositional variables the proposition depends on. With this notation, a truth table will have w^m rows. The first m columns in the truth table (corresponding to the propositional variables) can be organized following a fixed pattern.

3.2.1 Example.- If $w = 3$ and $m = 2$ the 8 rows can be ordered the following way:

0	0	...
$\frac{1}{2}$	0	...
1	0	...
0	$\frac{1}{2}$...
$\frac{1}{2}$	$\frac{1}{2}$...
1	$\frac{1}{2}$...
0	1	...
$\frac{1}{2}$	1	...
1	1	...

In general, a truth table corresponding to propositions including m propositional variables, in a w -valued logic, will have w^m rows. The values given to the first propositional variables (first column) can be loops of

$$0, \frac{1}{w}, \frac{2}{w}, \dots, \frac{w-1}{w}, 1$$

those given to the second propositional variables (second column) can be loops of

$$0, 0, \underbrace{\dots}_w, 0, \frac{1}{w}, \frac{1}{w}, \underbrace{\dots}_w, \frac{1}{w}, \frac{2}{w}, \frac{2}{w}, \underbrace{\dots}_w, \frac{2}{w}, \dots, \frac{w-1}{w}, \frac{w-1}{w}, \underbrace{\dots}_w, \frac{w-1}{w}, 1, 1, \underbrace{\dots}_w, 1$$

...

those given to the i -th propositional variable (i -th column) can be loops of

$$0, 0, \underbrace{\dots}_{w^i}, 0, \frac{1}{w}, \frac{1}{w}, \underbrace{\dots}_{w^i}, \frac{1}{w}, \frac{2}{w}, \frac{2}{w}, \underbrace{\dots}_{w^i}, \frac{2}{w}, \dots, \frac{w-1}{w}, \frac{w-1}{w}, \underbrace{\dots}_{w^i}, \frac{w-1}{w}, 1, 1, \underbrace{\dots}_{w^i}, 1$$

...

Therefore, if j_- is the number of the row, the corresponding value in the j -th row of the different propositional variables can be calculated in the language of these TIs, as follows

```
(1/(w-1))*MOD(FLOOR(j_-/w^0),w) STO p
(1/(w-1))*MOD(FLOOR(j_-/w^1),w) STO q
(1/(w-1))*MOD(FLOOR(j_-/w^2),w) STO r
(1/(w-1))*MOD(FLOOR(j_-/w^3),w) STO s
(1/(w-1))*MOD(FLOOR(j_-/w^4),w) STO u
(1/(w-1))*MOD(FLOOR(j_-/w^5),w) STO v
```

...

Let us denote by vp the list of propositional variables that can be used

```
{p,q,r,s,u,v} STO vp
```

(this list could obviously be enlarged if necessary).

The main procedure to construct truth tables is denoted $TT(m, a, b_-)$, where

0	0	0	0	0
1/2	0	0	0	0
1	0	0	0	0
0	1/2	0	0	0
1/2	1/2	0	1/2	1/2

Figure 1: Truth table of Example 4

- i) m is the number of different propositional variables that appear in a and b (that must be the first m in vp).
- ii) a and b are the formulae which truth table has to be calculated.

It fills the rest of the truth table with the values the propositions have when the propositional variables that appear in it are given the values in the first columns of the same row. It can be implemented as follows

```
Define TT(m_,a_,b_) =
seq(augment(seq(vp[i_],i_,1,m_),{a_,b_}),j_,0,w^m-1)
```

The code is rather cryptic. As a hint note that

```
augment(seq(vp[i_],i_,1,m_),{a_,b_})
```

constructs the (row) vector of the m first propositional variables in vp followed by a and b . Then a matrix is constructed by the “seq” outside by stacking this row vectors upon each others when giving to j the values 0 to $w^m - 1$. As the elements in vp are functions in the variable j , and a and b are functions in the variables in list vp , this is a numerical matrix.

3.2.2 Example.- Check the distributivity of \wedge w.r.t. \vee in Kleene’s 3-valued logic using truth tables. The user has to type

```
3 STO w
```

afterwards the package has to be loaded and the user has to type

```
TT(3, MAND(P,MOR(Q,R)), MOR(MAND(P,Q),MAND(P,R)) )
```

and the fourth and fifth columns obtained are equal (Figure 1). Observe that, from now onwards, the number of truth values is 3. All the examples in this paper take just a few seconds in these TIs.

3.3 Checking Tautologies

ISTAUT(m , a) is similar to TT(m , a , b), but instead of constructing the truth table corresponding to propositions a and b , it calculates the product of all the truth values of the column corresponding to proposition a . Then it checks whether this product is 1 or not, answering 1 (YES) or 0 (NO), respectively.

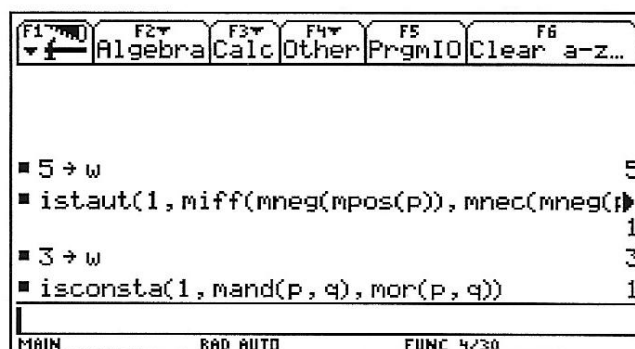


Figure 2: Examples 5 and 6 checked with the TI-92

Define $\text{istaut}(m_a) = \text{when}(\text{product}(\text{seq}(a_j, 0, w^{m-1})) = 1, 1, 0)$

This function – and the next two ones as well were defined originally as If-constructions of some lines. I replaced them in agreement with Eugenio by the shorter when-construction, Josef

3.3.1 Example.- Check that, in Kleene's 5-valued logic, "not possible" is equivalent to "necessarily not". If the user starts in a clean session, he has to type

5 STO w

then the package has to be loaded and the user has to type

and the answer is 1, i.e., "True" (Figure 2). Observe that if we are following the session of Example 3.2.2, it isn't enough to type

5 STO w

to switch to the 5-valued logic; immediately after assigning to w the new value (5), the definitions of p, q, r, s, u, v and vp have to be run again (once) in order to update their values.

3.4 Checking Tautological Consequences

$\text{ISCONSTA}(m_a, b_)$ checks whether $b_$ is a tautological consequence of $a_$ or not. It is similar to $\text{ISTAUT}(m_b_)$, but each new truth value of $b_$ is multiplied iff the truth value corresponding to $a_$ is 1. It uses the auxiliary procedure $\text{AUXT}(a_b_)$.

Define $\text{auxt}(a_b) = \text{when}(a_ = 1, b_ , 1)$

Define `isconsta(m_,a_,b_)=when(product(seq(auxt(a_,b_),j_,0,
m_-1))=1,1,0)`

3.4.1 Example.- Check that, in Kleene's 3-valued logic, $P \vee Q$ is a tautological consequence of $P \wedge Q$. When ready to work with 3-valued logic (see Example 3.3.1), the user has to type

`ISCONSTA(2, MAND(P,Q) , MOR(P,Q))`

and the answer is 1, i.e., "True" (Figure 2).

3.5 Examples

With this implementation it is possible to easily prove (using truth tables) laborious results like

- Classic bivalued logic, 3-valued and 5-valued logics are lattices, but only the first one is a Boolean algebra.
- In classic bivalued logic $(P \wedge Q) \models (P \vee Q)$ and $\models ((P \wedge Q) \rightarrow (P \vee Q))$, but in Kleene's p-valued logic ($p > 2$)..only.the.first holds.

4 Conclusions

This implementation probably makes the TI 92 and 89 the only calculators capable to deal with multivalued logics. We consider this implementation a very useful and interesting enhancement of their possibilities.

References

- [1] **Anonymous:** *TI-92 Guidebook*. Texas Instruments Inc., 1995.
- [2] **Anonymous:** *TI-89 Guidebook*. Texas Instruments Inc., 1998.
- [3] **J. Chazarain, A. Riscos, J.A. Alonso, E. Briales:** *Multivalued Logic and Gröbner Bases with Applications to Modal Logic*. Journal of Symbolic Computation 11, 1991 (pages 181-194).
- [4] **D. Cox, J. Little, D. O'Shea:** *Ideals, Varieties, and Algorithms*. Springer-Verlag, 1992.
- [5] **M. Garbayo, E. Roanes M., E. Roanes L.:** *Automating Logic with Derive* The International Derive Journal, Vol. 1, No. 3, 1994 (pages 73-80).
- [6] **J. Hsiang:** *Refutational Theorem Proving using Term-Rewriting Systems*. Artificial Intelligence, vol. 25, 1985 (pages 255-300).

- [7] **D. Kapur, P. Narendran:** *An Equational Approach to Theorem Proving in First-Order Predicate Calculus*. 84CRD296 General Electric Corporate Research and Development Report, Schenectady, NY, March 1984, rev. Dec. 1984. Also in: *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI-85)*, vol. 2 (pages 1146-1153).
- [8] **L. M. Laita, L. de Ledesma, E. Roanes L., E. Roanes M.:** *An Interpretation of the Propositional Boolean Algebra as a k-algebra. Effective Calculus*. In: **J. Campbell, J. Calmet** (editors): *Proceedings of the Second International Workshop/Conference on Artificial Intelligence and Symbolic Mathematical Computing (AISMC-2)*. LNCS 958. Springer-Verlag, 1995 (pages 255-263).
- [9] **E. Roanes L., L. M. Laita, E. Roanes M.:** *Maple V in A.I.: The Boolean Algebra Associated to a KBS*. CAN Nieuwsbrief num. 14, April 1995 (pages 65-70).
- [10] **E. Roanes L., L. M. Laita, E. Roanes M.:** *An Inference Engine for Propositional Two-valued Logic Based on the Radical Membership Problem*. In: **J. Campbell, J. Calmet, J. Pfalzgraf** (editors): *Proceedings of the Third International Workshop/Conference on Artificial Intelligence and Symbolic Mathematical Computing (AISMC-3)*. LNCS 1138. Springer-Verlag, 1996 (pages 71-86).
- [11] **E. Roanes L., L.M. Laita, E. Roanes M.:** *A Polynomial Model for multivalued Logics with a Touch of Algebraic Geometry and Computer Algebra*. In: **E. Roanes L., Stanly Steinberg, Hoon Hong** (editors): *Non Standard Applications of Computer Algebra* (Special Volume), Mathematics and Computers in Simulation, vol. 45 (1) 1998 (pages 83-99).
- [12] **L.M. Laita, E. Roanes-L., V. Maojo:** (1998) *Inference and Verification in Medical Appropriateness Criteria*. In: **J. Calmet, J. Plaza** (editors): *Artificial Intelligence and Symbolic Computation, Procs. AISC'98*. LNAI 1476. Springer-Verlag, 1998 (pages 183-194).
- [13] **E. Roanes-L., L.M. Laita, E. Roanes-M.:** *An Application of an AI Methodology to Railway Interlocking Systems using Computer Algebra*. In: **A. Pasqual del Pobil, J. Mira, M. Ali** (editors): *Tasks and Methods in Applied Artificial Intelligence, Procs. IEA-98-AIE* (Vol. II). LNAI 1416. Springer, 1998 (pages 687-696).
- [14] **E. Roanes L.:** *Introducing Propositional multivalued Logics with the Help of a CAS*. In: **R.P. Gilbert et al.** (editors): *Recent Developments in Complex Analysis and Computer Algebra (Proceedings of ISAAC 1997)*. Kluwer Academic Pub., 1999 (pages 277-290).
- [15] **E. Roanes L.:** *Constructing Truth-Tables in Propositional multivalued Logics with DERIVE 4*. Electronic Proceedings of the 1999 ACDCA Summer Workshop. Goesing (Austria), 1999.
- [16] **R. Turner:** *Logics for Artificial Intelligence*. Ellis Horwood, 1984.

The next page shows the realisation of Eugenio Roanes' functions on the TI-NspireCAS – made in 2018 which is 18 years after defining the TI-92 code. Josef

As you can see we can transfer the code without any problems from the TI-92 to TI-Nspire.

```

mneg(a_):=1-a_ ▶ Done          mpos(a_):=1-floor(1-a_) ▶ Done
mnec(a_):=floor(a_) ▶ Done      mor(a_,b_):=max(a_,b_)
mand(a_,b_):=min(a_,b_) ▶ Done
mimp(a_,b_):=mor(mneg(a_),b_) ▶ Done  miff(a_,b_):=mand(mimp(a_,b_),mimp(b_,a_)) ▶ Done
t:=1 ▶ 1  c:=0 ▶ 0

p:=1/(w-1) * mod(floor(j_/w^0),w) * mod(floor(j_/3),3)  q:=1/(w-1) * mod(floor(j_/w^1),w) * mod(floor(j_/3),3)
r:=1/(w-1) * mod(floor(j_/w^2),w) * mod(floor(j_/9),3)  s:=1/(w-1) * mod(floor(j_/w^3),w) * mod(floor(j_/27),3)
u:=1/(w-1) * mod(floor(j_/w^4),w) * mod(floor(j_/81),3)  v:=1/(w-1) * mod(floor(j_/w^5),w) * mod(floor(j_/243),3)
vp:={p,q,r,s,u,v}
tt(m_,a_,b_):=seq(augment(seq(vp[i_],i_,1,m_),{a_,b_}),j_,0,w^m-1) ▶ Done

```

I did the first part on a Notes-page. The truth table from fig. 1 is presented on the handheld screen. Calculation is performed in an instant.

I proceed on a calculator page for defining the remaining functions and the closing examples.

1.1	1.2	logic_pac	RAD
w:=3			3
tt(3,mand(p,mor(q,r)),mor(mand(p,q),mand(p,r)))			
	0.	0.	0.
	0.5	0.	0.
	1.	0.	0.
	0.	0.5	0.
	0.5	0.5	0.
	1.	0.5	0.
	0.	1.	0.

istaut(m_,a_):=when(product(seq(a_,j_,0,w^m-1))=1,1,0)	Done
w:=5	5
istaut(1,miff(mneg(mpos(p)),mnec(mneg(p))))	1
auxt(a_,b_):=when(a_=1,b_,1)	Done
isconsta(m_,a_,b_):=when(product(seq(auxt(a_,b_),j_,0,w^m-1))=1,1,0)	Done
w:=3	3
isconsta(1,mand(p,q),mor(p,q))	1

Programming with Derive

Dipl.-Phys. Walter Schiller*

Lagesche Str. 32

33102 Paderborn

Germany

December 4, 2000

The new feature of the Derive version 5 is among other things the possibility to write programs using the "usual" procedural programming style. I will demonstrate, how you can simple write programs in Derive.

In illustration of these possibility we will write the procedure for computing the fast Fourier and inverse Fourier transform of a complex sequence of length n . Following to I give an example for use of the fast Fourier transform.

Because of is it troublesome to edit a bulky procedure in the entry line in Derive, I have written a simple compiler, that translate the source program, edited with a word processing program you are familiar with, into a corresponding loadabel Math-file. My compiler has the name "tom" and I have sent it to Josef Boehm. To run the program, you must call it as following:

```
tom filename
```

where filename is an arbitrary name. He can written with or without suffix. If you have named your source file e.g. `fft.derive` or only `fft`, the result is in any case a loadable Math-file for Derive. With our example the name is `fft.mth`.

1 Program Organisation

This chapter discusses how statements can be organized into blocks to form a Derive program and how control flows within a program from one statement to another.

A Derive program is essentially a function, that means the procedure returns a value. The main procedure must be a function definition. For example:

```
myprog(p1,p2,p3,...,pn) := prog
  instruction_1
  instruction_2
  instruction_3
  .
  .
  .
  instruction_n
end
```

Each instruction is either a Derive expression, an assignment or a keyword instruction. The instructions are separated by semicolons.

How is to edit a Derive program?

*Phone 05251 480488 Germany

1.1 Blanks and Whit Space

Blanks (spaces) may be freely used in a program to improve appearance and layout, and most are ignored. Blanks, however, are usually significant

- within literal strings (see below)
- between two tokens that are not special characters (for example, between two symbols or keywords)
- between the two characters forming a comment delimiter

Note: You cannot take a blank as a multiplication operator!

1.2 Comments

Commentary is included in a Derive program by means of *comments*. Any sequence of characters on one or more lines that are delimited by `"/*"` and `"*/"` is a comment. Comments may be anywhere, and may be of any length. Comments are ignored by the program and do not affect execution of a program.

An example of a comment is:

```
a := /* This sentence could be
      inserted as a comment */ 5
```

The two characters forming a comment delimiter (`"/*"` or `"*/"`) must be adjacent (that is, they may not be separated by blanks or lin-end).

1.3 Implied semicolons and continuations

A semicolon (instruction end) is implied at the end of each line, except if:

1. The line ends in the middle of a comment, in which case the instruction continues at the end of the comment.
2. The last token was a backslash. In this case the backslash is functionally replaced by a null string, and hence acts as a *continuation character*.

For example

```
y := 5\
2 + 7

act as

y := 52 + 7
```

Notes:

Semicolons are added automatically by the compiler after certain instruction keywords when in the correct context. The keywords that may have this effect are **then**, **else** and **otherwise**. These special cases reduce program entry errors significantly.

1.4 The case of names

Variable names can generally be written arbitrarily. Keyword names and names of builtin functions are recognized correspondingly. All other names keep their notation.

2 Keyword Instructions

A Derive program is combined of assignment statements, expressions and keyword instructions.

A keyword instruction is one or more clauses, the first of which starts with a keyword that identifies the instruction. Some keyword instructions (**do**, **if**, **loop** or **select**) can include nested instructions.

Certain other keywords, known as *sub-keywords*, may be known within the clauses of individual instructions — for example, the symbols **to** and **while** in the **loop** instruction.

Blanks adjacent to keywords have no effect other than that of separating the keyword from the subsequent token. For example, this applies to the blanks next to the sub-keyword **while** in

```
loop while x = y
```

Here at least one blank was required to separate the symbols forming keywords and the variable name, x.

2.1 DO Instruction

```
DO;  
  instructionlist  
END;
```

The **do** instruction is used to group instructions together for execution; these are executed once. The most common use of **do** is simply for treating a number of instructions as a group.

Example:

```
/* The three instructions between DO and END will  
   all be executed, if x has the value 10 */  
if x = 10 then do  
  a := 2  
  b := int(a*cos(x),x)  
  v := append(v,[b])  
end
```

The instructions in the *instructionlist* may be any assignments, expressions or keyword instruction, including any of more complex constructions such as **loop**, **if** and **select**.

2.2 GLOBAL Instruction

```
GLOBAL identifier [,identifier] ...
```

The following rules are used to determine whether a variable is local or global: if variables are undeclared, by default, each variable to which an assignment is made, or which appears as the controlling variable in a **LOOP** instruction, is local. All others are global. That is, if you will assign a value to a global variable, you must declare such a variable with the global instruction.

2.3 IF Instruction

```

IF expression [;]
  THEN [;] instruction
  [ELSE [;] instruction]
  [OTHER [;] instruction]

```

The **if** construct is used to conditionally execute an instruction or group of instructions.

The expression is a condition or a logical construct **true** or **false**. The expression is evaluated and must result in either true or false. If the result was false and an **else** was given then the instruction after **else** is executed.

Example:

```

if c < d then
  xj := yes
else
  xn := no

```

This if statement can also be written as follows:

```

if c < d then xj := yes else xn := no

```

Another notation is:

```

if c < d
then
  xj := yes
else
  xn := no

```

The **else** binds to the nearest **then** at the same level. This means that any **if** that is used as the instruction following the **then** in an **if** construct that has an **else** clause, must itself have an **else** clause.

Example:

```

if a1 = b1
then if a2 = b2
  then if a3 = b3
    then result3 := 3
    else result3 := -3
  else result2 := -2
else result1 := -1

```

You can write this whole **if**-statement on one line, but this is not a good idea!

To include more than one instruction following **then** or **else**, use the grouping instruction **do**.

Example:

```

if a < b then do
  j1 := 1
  j2 := 2
end
else do
  n1 := -1
  n2 := -2
  n3 := -3
end

```

You are not limited to two choices. You can use the **select** instruction to have a Derive program select one of any number of branches.

2.4 SELECT Instruction

```
SELECT;
  whenlist
OTHERWISE [;] instruction
END;
```

where *whenlist* is one or more *whenconstructs* as following:

```
WHEN condition THEN instruction
```

For example:

```
select
  when condition1 then instruction1
  when condition2 then instruction2
  when condition3 then instruction3
  ...
  otherwise do
    instruction
    instruction
    instruction
    ...
  end
end
```

select conditionally executes one of several alternative instructions.

Each expression after **when** is evaluated in turn and must result in true or false. If the result is true, the construction following the **then** (which may be a single instruction or a **do** group) is executed and control then passes to the **end**. If the result is false, control passes to the next **when** clause.

If none of the **when** condition is true, control passes to the instruction after **otherwise**.

otherwise is essentially the **select** equivalent of **else**. If there is any possibility that all the **when** conditions could be false, there must be an **otherwise** clause.

2.5 LOOP Instruction

```
LOOP [repetitor] [conditional];
  instructionlist
END;
```

where *repetitor* is one of:

```
reference := expri [TO expri] [BY expri] [FOR expri]
FOR expri
FOREVER
```

and *conditional* is either of:

```
WHILE expri
UNTIL expri
```

The **loop** instruction is used to group instructions together and execute them repetitively.

Syntax notes:

- The **to**, **by**, and **for** phrases in the first form of *repetitor* may be in any order, if used.
- The *exprw* (**expression for while**) or *expru* (**expression for until**) (if present) can be an expression that evaluates to true or false.
- The *exprb* (**expression for by**) option defaults to 1, if relevant.

Computer excel at repetitive tasks. An essential part of any computer language is a *loop* instruction, which is a way to make a program repeat a list of instructions:

- A specific number of times
- As long as some condition is true
- Until some condition is satisfied
- Forever (until the user wants to stop).

2.5.1 Simple Repetitive Loops

A simple repetitive loop is a repetitive loop in which the repetitive phrase is an expression that evaluates to a count of the iterations. To repeat a loop a number of times, use:

```

loop for exprr
    instruction1
    instruction2
    instruction3
    ...
end

```

To make a program easier for people to read, you should indent the instructions between the **loop** and the **end** three spaces to right.

Example:

```

v := []
loop for 5
    v := append(v, ["Hello"])
end

```

The variable *v* has after that the value [Hello,Hello,Hello,Hello,Hello]

2.5.2 Controlled Repetitive Loops

The controlled form specifies a **control variable**, *reference*, which is assigned an initial value (the result of *expri*) before the first execution of the instruction list. The variable is then stepped (by adding the result of *exprb*, at the bottom of the loop) each time the group of instruction is executed. The group is executed repeatedly while the end condition (determined by the result of *exprt*) is not met. If *exprb* is positive or 0, the loop is terminated when *reference* is greater than *exprt*. If negative, the loop is terminated, when *reference* is less than *exprt*.

The *expri*, *exprt*, and *exprb* are evaluated once only, before the loop begins. The default value for *exprb* is 1.

Examples:

```

1.      ...
   v := []
   loop i := 3 to -2 by -1
     v := append(v,[i])
   end

      ...
   <v> = [3,2,1,0,-1,-2]

2.      ...
   v := []
   x := 0.3
   loop y := x to x+4 by 0.7
     v := append(v,[y])
   end

      ...
   <v> = [0.3,1,1.7,2.4,3.1,3.8]

```

Note: one receives this result, if you run the program with the **Approximate** button.

The execution of a controlled loop can be bounded further by a **for** phrase. In this case, you must specify *exprf*, and it must evaluate to a **nonnegative** number!

Example:

```

v := []
loop y := 0.3 to 4.3 by 0.7 for pi
  v := append(v,[y])
end

```

The result is $v = [\frac{3}{10}, 1, \frac{17}{10}]$ in exact mode.

2.5.3 Conditional Loops

A conditional expression is tested to determine how many times the loop is processed. Conditional loops continue running as long as some condition is satisfied. The three main ways to do this, depending on when the test takes place are:

- **loop forever** (with **break**)
- **loop while** (a condition is true)
- **loop until** (a condition is true)

FOREVER with the BREAK Instruktion The simplest way to create a conditional loop is to use the **loop** instruction **forever** and **break**. The **break** instruction causes processing to continue with the construction following the **end** keyword.

Example:

```

i := 0
v := []
loop forever
  i := i+1
  v := append(v,["Hello"])
  if i = 3
    then break
  end
end

```

The result is $v := [\text{Hello}, \text{Hello}, \text{Hello}]$

LOOP WHILE Instruction To create a loop that repeats the list of instructions as long as a given condition is true, use the **loop while** instruction. For example:

```

loop while exprw
  instruction1
  instruction2
  instruction3
  ...
end

```

where *exprw* is an expression that, when evaluated, must give a result of true or false.

The condition is tested at the top of the loop, before the instruction list is processed. This means that if the given condition is false at the start, the list of instructions are not be processed at all.

Example:

```

i := 0; v := []
loop while i < 6
  i := i + 1
  v := append(v, [i])
end

```

The result is *v*:=*[1,2,3,4,5,6]*

LOOP UNTIL Instruction To repeat one or more instructions *until* a given condition is true, you can create a loop with the test at the bottom. For example:

```

loop until expru
  instruction1
  instruction2
  instruction3
  ...
end

```

where *expru* is an expression, when evaluated, must give a result of true or false.

Putting the test at the bottom of the loop means that the enclosed instruction list is always processed *at least once*, even if the condition is false at the start.

Example:

```

i := 0
v := []
loop until i > 6
  i := i + 1
  v := append(v, [i])
end

```

The result is *v*:=*[1,2,3,4,5,6,7]*

2.5.4 Put all together

A conditional phrase, which may cause termination of the loop, can follow any of the form of *repetitor*. If you specify **while** or **until**, *exprw* or *expru*, respectively, is evaluated each time around the loop using the latest values of all variables (and must evaluate to either true or false), and the loop is terminated, if *exprw* evaluates to false or *expru* evaluates to true.

For a **while** loop, the condition is evaluated at the top of the group of instructions. For an **until** loop, the condition is evaluated at the bottom – before the control variable has been stepped.

Example:

```
v := []
loop i := 1 to 10 by 2 until i > 6 for 3
  v := append(v, ["Hello"])
  if i = 2 then break
end
The result is v:=[Hello,Hello,Hello]
```

Here is a whole program as an example for the tests with loop instructions.

```
F(x) := prog
  w1 := []; i:=0
  loop while i<6
    i :=+ 1
    w1 := append(w1,[i])
  end

  u1 := []; i := 0
  loop until i > 6
    i :=+ 1
    u1 := append(u1,[i])
  end

  v := []
  loop i := 1 to 10 by 2 until i > 6 for 3
    v := append(v,["Hello"])
    if i = 3 then break
  end

  u := []
  loop i := 3 to -2 by -1
    u := append(u,[i])
  end

  w := []
  x := 0.3
  loop y := x to x+4 by 0.7 for pi
    w := append(w,[y])
  end
  return([w1,u1,v,u,w])
end /* program end */
```

With the **File > Math File** command load this program in an algebra window, double click on the highlighted instructions and press ENTER. If you just now run this defined program, then the result is:

$[1, 2, 3, 4, 5, 6], [1, 2, 3, 4, 5, 6, 7], [Hello, Hello], [3, 2, 1, 0, -1, -2], [\frac{3}{10}, 1, \frac{17}{10}]$

2.6 BREAK Instruction

BREAK;

break causes an immediate exit from one repetitive LOOP instruction.

Processing of the group of instructions is terminated, and control is passed to the instruction following the **end** clause, just as though the **end** clause has been encountered and the termination condition has been met normally. However, on exit, the control variable (if any) will contain the value it had when the **break** instruction was processed.

Note:

break terminates only the innermost active repetitive loop.

3 Compiling Multiple Programs and Instructions in a Single Step

The Derive compiler allows to compile more than one program in a single step. You can also compile single Derive commands and/or programs mixed in one file. The only things, that you can write in a file for a single call of the compiler are programs and single Derive commands. Compound statements such as IF- or LOOP-instructions must be written as programs.

Example:

```
a := [x,y,z]

b := sin(x^2-z)+tan(2*x)

f1(x,y) := prog
...
...
end

f2(p1,p2):=prog
...
...
end

auf1 := f1((b,b1-#pi/2)
```

4 The Fast Fourier Transform Program

And now we will write a really program for Derive, the program for computing the fast Fourier and inverse Fourier transform of a complex sequence of length n . The calling sequence should be as following:

```
fft(a,b,m,inv)
```

where the first and the second arguments (a and b) should be vectors of length $n = 2^m$. The third argument (m) should be a non-negative integer. The vector a contains the real part of the complex sequence on input. The vector b (the second argument) contains the imaginary part of the complex sequence on input. The fourth argument (inv) is a boolean value indicating the direction of the transform, if inv is false then the result contains the Fourier transform, if inv is true then the result contains the inverse Fourier transform. The result of a call from the fast Fourier transform is a matrix with two rows. The first row contains the real part of the fast Fourier transform on output and the second row contains the imaginary part. (Look on the examples in the description of this FFT-program).

You must use **File>Load>Math File** and define the loaded function with either double click and ENTER or with the **Declare>Function Definition** command in an algebra window.

4.1 A Nonrecursive FFT-Program

Here is the program. (I have edited this file in a text editor for windows and have named this source program file just **fastf.derive**. The source program contains in a few comment lines examples of calling sequences and a description of the parameters.

```

/*****
/* This routine computes the Fast Fourier Transform of the input data*/
/* and returns the results */
/*
/*      Walter Schiller
/*      33102 Paderborn
/*      Germany
/*      Lagesche Str. 32
/*
/*
/*              start date: October, 2000
/*
/*
/* This routine performs the Fast Fourier Transform by the method
/* of Cooley and Tukey.
/*
/* The vector a contains the real data and vector b contains the
/* imaginary data to be transformed. m is LOG(N,2) and inv is
/* a logical value. inv = false if forward transform and true
/* for the inverse transform.
/* The output data is a matrix. The first row is the real part
/* and the second row is the imaginary part.
/*
/* Examples:
/* 1.      x := [1,2,3,4]
/*          y := [0,0,0,0]
/*          fft(x,y,2,false)= [10 -2 -2 -2]
/*                             [0  2  0 -2]
/*
/*          fft([10,-2,-2,-2],[0,2,0,-2],2,true)= [1 2 3 4]
/*                                                  [0 0 0 0]
/*
/* 2.      fft([7,5,6,9],y,2,false)= [27 1 -1 1]
/*                                   [0  4  0 -4]
/*
/*          fft([27,1,-1,1],[0,4,0,-4],2,true)
/*
/*          result of this inverse transform (4. parameter in call
/*                                   is true):
/*
/*          [7 5 6 9]
/*          [0 0 0 0]
/*
/* 3.      a := [0,0,1,0,0,0,0,0]
/*          b := [0,0,0,0,0,0,0,0]
/*
/*          fft(a,b,3,false)= [1 0 -1 0 1 0 -1 0]

```

#

```

/*          [0 -1  0  1  0 -1  0  1]          */
/*          */
/*      It's clear that the inverse transform with this      */
/*      output datas result in the original datas.          */
/*          */
/*      4.      Another practicability is the following call: */
/*          */
/*      result := fft(a,b,3,false)                        */
/*          */
/*      Inverse transform:                                */
/*      fft(result sub 1, result sub2, 3, true)           */
/*          */
/*      result    corresponds with a                      */
/*          1                                           */
/*      result    corresponds with b                      */
/*          2                                           */
/*          */
/*      My filename of this program is:                   */
/*          */
/*      fastf.derive                                       */
/*          */
/*      With my Compilerprogram 'tom' you can translate this program */
/*      into a lodable .mth Derive program by the following call: */
/*          */
/*      tom fastf.derive                                   */
/*          */
/*      The result is the Derive program                  */
/*          */
/*      fastf.mth                                         */
/*          */
/*****
FFT(a,b,m,inv):=prog

n := 2^m      /* must be a power of 2 */
if not dim(a)=n or not dim(b) = n then
    return("The length of the data is not an exact power of 2")

nd2 := n/2

j := 1
loop i :=1 to n-1
    if i < j
        then do
            tr      := a sub j
            ti      := b sub j
            a sub j := a sub i
            b sub j := b sub i
            a sub i := tr
            b sub i := ti
        end
    k := nd2
loop while k < j
    j :=- k
    k :=/ 2

```



```

    end /* end while */
    j :=+ k
end /* end i */

le := 1
loop l := 1 to m
    le1 := le
    le := le + le
    ur := 1
    ui := 0
    ang := pi/le1
    wr := cos(ang)
    wi := -sin(ang)
    /* here for inverse FFT: */
    if inv then wi := -wi
    loop j := 1 to le1
        loop i := j to n by le
            ip := i + le1
            tr := a sub ip * ur - b sub ip * ui
            ti := b sub ip * ur + a sub ip * ui
            a sub ip := a sub i - tr
            b sub ip := b sub i - ti
            a sub i := a sub i + tr
            b sub i := b sub i + ti
        end /* end i */
        tr := ur * wr - ui * wi
        ui := ui * wr + ur * wi
        ur := tr
    end /* end j */
end /* end l */

if inv
then do
    loop i := 1 to n
        a sub i := a sub( i) / n
        b sub i := b sub i / n
    end
end

return([a,b])
end

```

After that, if you have my compiler, you need only call the compiler with

```
tom fastf.derive
```

and you will have a really Derive Math program with the name **fastf.mth**.

You must use **File > Load > Math File** command or the Derive builtin function **LOAD(filename)** and define it. After that you can use the FFT-program. For example:

```
load("f:\Deriveprogrammierung\fastf.mth")
```

and then **Declare > Function Definition**. Examples of use of the FFT-routine are given in the description of this program, try it!

4.2 A recursive FFT-Program

In a book with the title *"Mathematik lernen mit Maple" Vol. 2, Wilhelm Werner, dpunkt.verlag* ISBN no. *ISBN 3-920993-94-2* i have found a recursive Maple procedure for the fast Fourier transform and for the inverse fast Fourier transform. I have these procedures adapted for Derive. It is very simple with my compiler!

To save space I skip the source code of the forward Fast Fourier Transform and the inverse one together with the respective mth.files as well. You can find all files on the diskette. I recommend to compare the documented .derive files with the tom-created mth-files. Josef.

5 Examples

I have a directory with the name **Deriveprogrammierung** on my partition **F**. All my Derive source files have the extension **.derive**. Among other things have I files for the fast Fourier transform. These are:

- **fastf.derive** (A nonrecursive procedure for forward and inverse Fourier transform). The name of the function is **FFT** and the definition is **FFT(a,b,m,inv)**
- **rfft.derive** (A recursive procedure for the fast Fourier transform). The name of the function is **rfft** and the definition is **rfft(y)**
- **rinvfft.derive** The name of the function is **rinvfft** and the definition is **rinvfft(y)**.

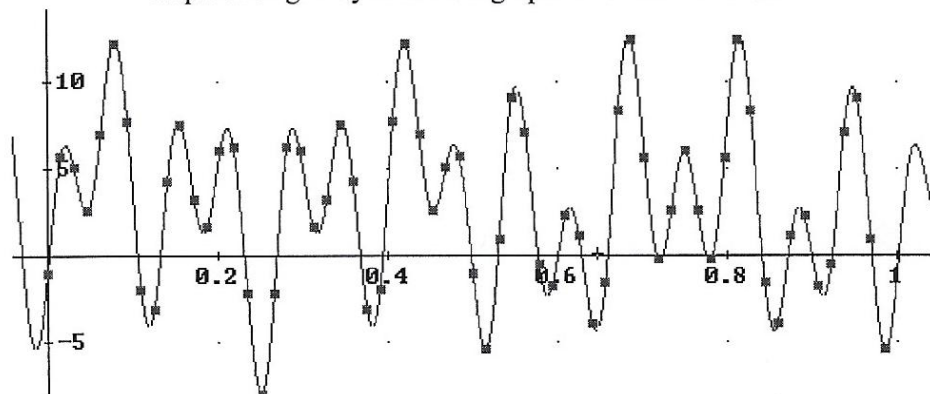
The both recursive procedures are onyl applicable with few datas. Because Derive has not the feature of the table (a array with arbitrary subscripts) therefore I can not speed up the calls of recursive procedures. For that reason take the first program **FFT** if you have many datas!

Data smoothing (filters) plays a central roll in techngical applications, e.g the signal transmitting and image processing. One has typically to do it with large amounts of data there. As example we want to apply the discrete Fourier transform to the data smoothing. To this, at first we create a series of 64 numerical values from an arbitrarily predefined function. This function is

$$f(t) := 2*\sin(3*t) - 4*\cos(8*t) + 5*\sin(15*t) + 3$$

We subject these numerical values to a Fourier transform. And than we smooth the frequency spectrum won so by the fact that we eliminate simply very small frequencies. From the remaind frequencies we create this one for function smoothed so. The appendix is a Derive worksheet, that show all. For a better understanding I have in addition given the two graphs on a extra sheet.

Graph of originally datas and graph of smoothed datas



***DERIVE* as Problem Generator**

Josef Böhm, Würmla, Austria

Despite the fact that my students and I now have powerful tools like the TI-92 or *DERIVE* for manipulating expressions, I wouldn't like to skip practising some basic manipulating skills at all. Several years ago Jan Vermeyleen sent some *DERIVE* (for DOS) files to create randomly generated exercises for factorizing, expanding etc. I remembered his ideas and realized them as you can see with *DfW5*. And I added an error analysis. (Now in 2018, I present the *DERIVE6* version, Josef.)

expand_ut.mth must be loaded as utility file

Expanding Binomials and other Expressions

sq = presents a binomial to be squared;
cu = presents a binomial to be cubed;
tr = presents a trinomial to be squared;
pr = presents a product of two binomials;
di = presents a product of sum and difference.

qu = gives a randomly chosen task of the above.

sqe(n)=, cue(n)=, tre(n)=, pre(n)=, die(n)= und **que(n)=** produce lists of n tasks of the respective problem group.

res = displays the result of the problem(s) set.

ch(my_answer) = gives an analyse of - only single - tasks.

Start with simplifying **random(0)** in order to initialize the random number generator!

#1: RANDOM(0)

Please follow a possible exercising session:

#1: RANDOM(0) = 2856264212

#2: LOAD(M:\DOKUS\DNLS\Dn100\mth40\expand_ut.mth)

#3: $sq = (8 \cdot o - 7 \cdot v)^2$

#4: $ch(64 \cdot o^2 - 56 \cdot o \cdot v + 49 \cdot v^2) = \text{check double product!}$

#5: $ch(64 \cdot o^2 - 112 \cdot o \cdot v + 49 \cdot v^2) = \text{correct!}$

#6: $cu = -(f + 6 \cdot u)^3$

#7: $ch(-f^3 - 18 \cdot f^2 \cdot u - 36 \cdot f \cdot u^2 + 226 \cdot u^3) = \text{check cubes!}$

#8: $res = -f^3 - 18 \cdot f^2 \cdot u - 108 \cdot f \cdot u^2 - 216 \cdot u^3$

#9: $pr = (c + 5 \cdot r) \cdot (6 \cdot c - r)$

#10: $ch(6 \cdot c^2 - 29 \cdot c \cdot r - 5 \cdot r^2) = \text{wrong variable or unidentified error}$

#11: $ch(6 \cdot c^2 - 29 \cdot c \cdot r - 5 \cdot r^2) = \text{check mixed product!}$

#12: $ch(6 \cdot c^2 + 29 \cdot c \cdot r - 5 \cdot r^2) = \text{correct!}$

I'll show only a part of the code. You will find the complete file on the diskette. It was nice and helpful to use Mr Schiller's TOM.EXE to work in an fullscreen editor (MS Word or Wordpad or the DOS editor) to write the *DERIVE*-functions with the many IFs and parentheses.

You can see completely how to create a binomial² together with its evaluation. The other problem types are produced in a very similar way. Don't forget, this is not the final *DERIVE* code, but has to be converted by TOM.EXE into a correct *DERIVE* code. TOM collects all variables in the parameter list, etc. Compare this section with the respective functions in **exprut.mth**!

Please notify the **'-operator**, which prevents autosimplification. If you are interested in such trainings-tools then send an e-mail or call me. I have also a ready made *DfW5*-file for practising factorising. If you prefer working with the TI-92/89, I can offer a *bk-teachware* booklet with these tools plus some more trainers (for calculus, linear functions).

```

l1:=[a,c,e,g,i,k,m,o,q,s,u,w,y]
l2:=[b,d,f,h,j,l,n,p,r,t,v,x,z]
task :=; type:=; group:=; k1:=; k2:=; k3:=; k4:=; v1:=; v2:=; v3:=

sq_(p1,p2):=prog
  global task,type,k1,k2,v1,v2
  v1 := l1 sub (random(13)+1)
  v2 := l2 sub (random(13)+1)
  k1 := random(10)+1
  k2 := random(10)+1
  p1 := k1*v1
  p2 := k2*v2
  type := 1
  task:= ['(p1+p2)^2','(p1-p2)^2','(-p1-p2)^2] sub (random(3)+1)
end

sq:=sq_(p1,p2)

tsq(ans):=prog
  if ans - task = 0
  then "correct!"
  else "wrong!"
  other if k1^2 /= subst(ans,[v1,v2],[1,0]) or \
          k2^2 /= subst(ans,[v1,v2],[0,1])
  then "check squares!"
  else if subst(ans-(k1*v1)^2-(k2*v2)^2, \
                [v1,v2],[1,1]) /= \
          subst(task-(k1*v1)^2-(k2*v2)^2, \
                [v1,v2],[1,1])
  then "check double product!"
  else "wrong!"
  other "wrong variable or unidentified error"
end

qu := [sq,cu,tr,pr,di] sub (random(5)+1)

sqe(n):=prog;global group,task ;task:=[vector(sq,k,1,n)]`;end

ch(ans) := [tsq(ans),tcu(ans),ttr(ans),tpr(ans),tdi(ans)] sub (type)

res:= expand(task)

```

[1] Mathe-Trainer I (for TI-89/92/92+), bk-teachware SR-15, ISBN 3-901769-24-2
(The booklet is in German, I am working on an English version. The programs are available in English together with the booklet on request.)

I added a TI-Nspire-version (expanding.tns, deutsche Version: terme.tns).

Working with binomials and other expressions

sq() presents a binomial to be squared, **sqq()** is a bit harder

cu() presents a binomial to be cubed, **cuu()** is a bit harder

tr() presents a trinomial to be squared, **trr()** ...

pr() presents a product of two binomials, **prr()** ...

di() presents a product of sum and difference, **dii()** ...

qu() and **quu()** give randomly chosen problems

sqe(n), **cue(n)**, **tre(n)**, **pre(n)**, **die(n)** and **que(n)** create a sample of n respective tasks.

res() will offer the result(s) of the given tasks.

Single problems can be analysed with **ch(my_answer)**.

Don't forget to initialize the random number generator by executing **randseed any integer**.

A sample session on two calculator pages. One program is displayed on the next page.

RandSeed 18112018	Done	cu()	
sq()			$-(2 \cdot s - 3 \cdot b)^3$
	$(5 \cdot d - g)^2$		Done
	Done	ch $(-8 \cdot s^3 + 36 \cdot s^2 \cdot b - 27 \cdot s \cdot n^2 + 27 \cdot b^3)$	
ch $(25 \cdot d^2 - 5 \cdot d \cdot g + g^2)$			wrong variable or unidentified error!
	check double product!		Done
	Done	ch $(-8 \cdot s^3 + 36 \cdot s^2 \cdot b - 27 \cdot s \cdot b^2 + 27 \cdot b^3)$	
ch $(25 \cdot d^2 - 10 \cdot d \cdot g + g^2)$			check mixed products!
	check squares or missing variable!		Done
	Done	res()	$-8 \cdot s^3 + 36 \cdot b \cdot s^2 - 54 \cdot b^2 \cdot s + 27 \cdot b^3$
ch $(25 \cdot d^2 - 10 \cdot d \cdot g + g^2)$		cuu()	
	correct!		$-(4 \cdot e^2 - 5 \cdot n^3)^3$
	Done		Done

$-(4 \cdot e^2 - 5 \cdot n^3)^3$

Done

res() $-64 \cdot e^6 + 240 \cdot e^4 \cdot n^3 - 300 \cdot e^2 \cdot n^6 + 125 \cdot n^9$

que(5)

$$\begin{bmatrix} (8 \cdot g - 9 \cdot p) \cdot (8 \cdot g + 9 \cdot p) \\ (2 \cdot c - p) \cdot (3 \cdot c - 10 \cdot p) \\ (s - 3 \cdot z) \cdot (s + 3 \cdot z) \\ (3 \cdot s - b - 4 \cdot k)^2 \\ (w - x + 2 \cdot g)^2 \end{bmatrix}$$

Done

res()

$$\begin{bmatrix} 64 \cdot g^2 - 81 \cdot p^2 \\ 6 \cdot c^2 - 23 \cdot c \cdot p + 10 \cdot p^2 \\ s^2 - 9 \cdot z^2 \\ 9 \cdot s^2 + (-6 \cdot b - 24 \cdot k) \cdot s + b^2 + 8 \cdot b \cdot k + 16 \cdot k^2 \\ w^2 + w \cdot (4 \cdot g - 2 \cdot x) + x^2 - 4 \cdot g \cdot x + 4 \cdot g^2 \end{bmatrix}$$

"sq" stored successfully

Define sq()=

Prgm

Local k1_,k2_

k1_:=rd(10);k2_:=rd(10)

k1:=(-1)rd(2). $\frac{k1_}{gcd(k1_,k2_)}$

k2:= $\frac{k2_}{gcd(k1_,k2_)}$

v1:=st(1,13);v1_:=v1

v2:=st(2,13);v2_:=v2

© v1_ and v2_ are needed for sqq()

typ:=1

task:=(k1·v1_+k2·v2_)²

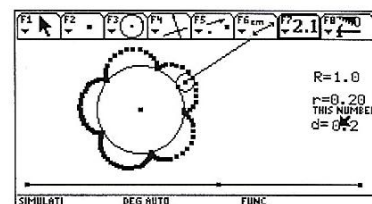
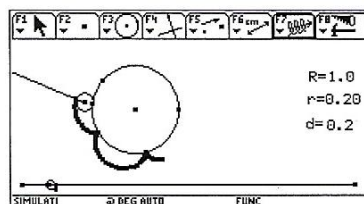
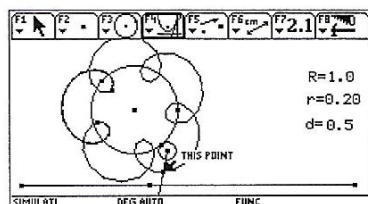
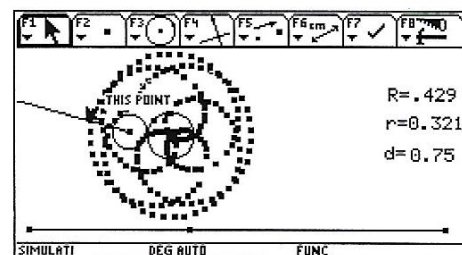
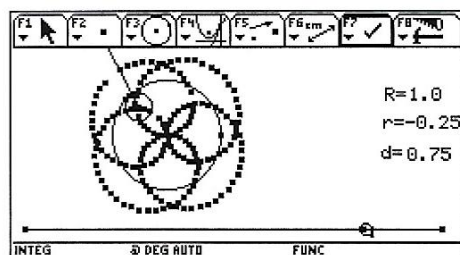
Disp task

EndPrgm

Lists of problems (like sqe(n), cue(n), ...) cannot be checked and analysed.

Trochoids on the TI-92

A short comment in Thomas Weth's contribution in DNL#39, page 32, inspired me to produce a CABRI-tool for creating various trochoids on the TI-92. Thomas Himmelbauer gave very valuable support and I tried to make the tool as user friendly as possible. One can change the values of R (radius of the fixed circle), r (radius of the rolling circle) and d (distance of the moving point from the rolling circle's center). Thomas and Josef



Moving the point on the segment lying on the bottom of the screen and tracing the point lying on the ray starting from the moving circle's center gives the trochoid. Using the locus-tool leads immediately to the epitrochoid or hypotrochoid.

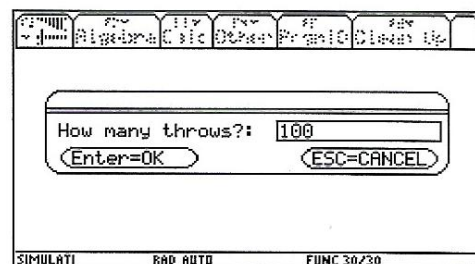
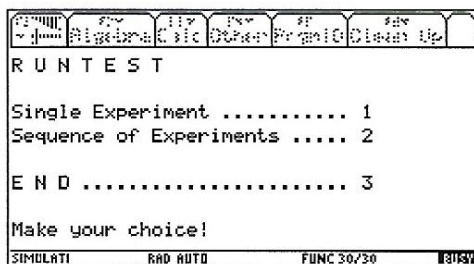
The Runtest - A simulation

A Run is a sequence of identical consecutive outcomes tossing a coin or producing a random binary sequence like 00001001101000111. This sequence contains 8 runs.

Such randomly created sequences provoke a lot of interesting explorations. The TI-92-program `runs ()` gives a simulation of a sequence of coin tosses. The average number of Heads or the average number of runs in one experiment or in a series of experiments are only two of several interesting outcomes. (`runtest ()` on the diskette is the German version. The program should work on the TI-89, too. It could happen that some window setting must be adapted.)

It is funny to ask students to toss a coin 50 times and note Heads and Tails. Some of them should produce this sequence without using a coin, just writing down a random sequence of H and T. If they don't know the point you can be sure to recognize in most cases who of them produced a real random sequence using the coin and who did not, because the "hand made" random sequence mostly shows too many runs. The expected value of runs after n tosses is $\frac{n+1}{2}$.

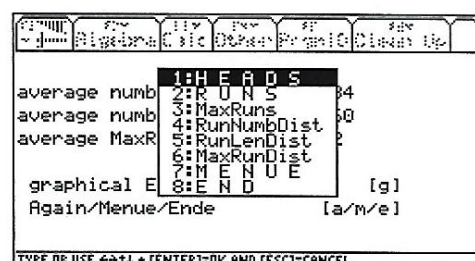
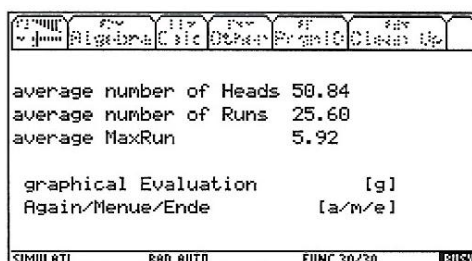
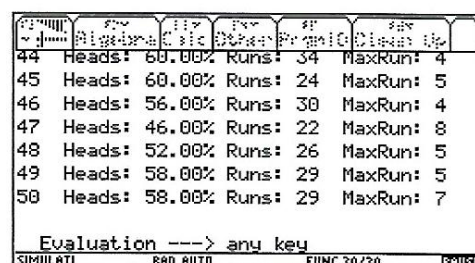
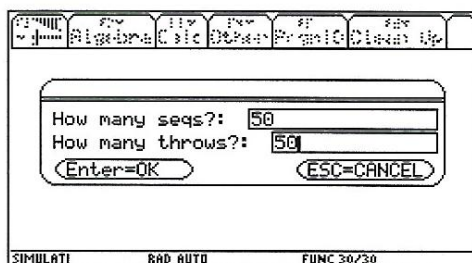
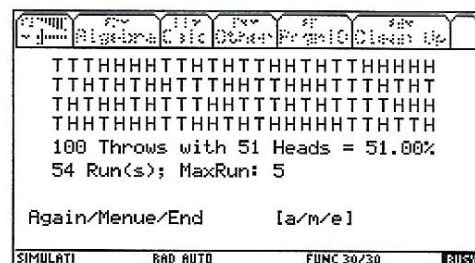
```
runs()
```

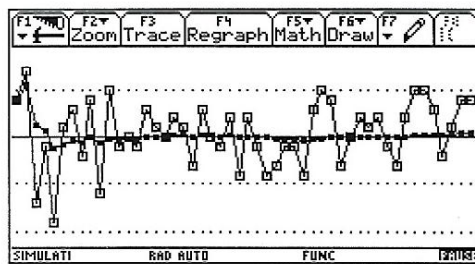


One experiment with 100 coin tosses:

Add and evaluate the outcomes of the whole class!

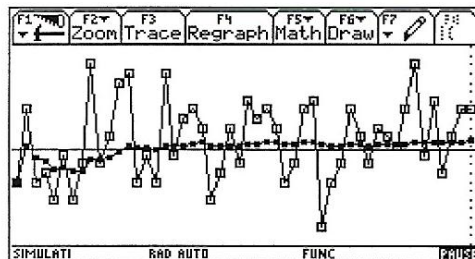
Now we produce a sequence of 50 times 50 tosses and evaluate the results in various manners graphically:





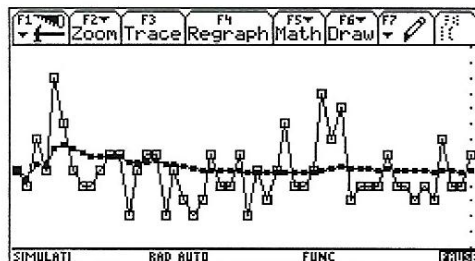
1: HEADS

Heavy oscillating between the experiments but showing the convergence of the cumulated frequencies towards 0.5.



2: RUNS

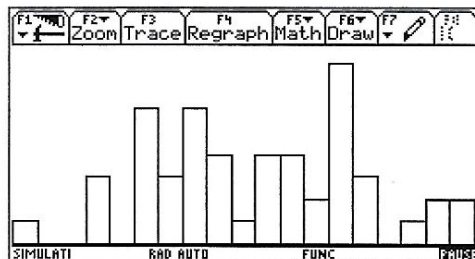
The average number of Runs tends towards 25.5.



3: maxRuns

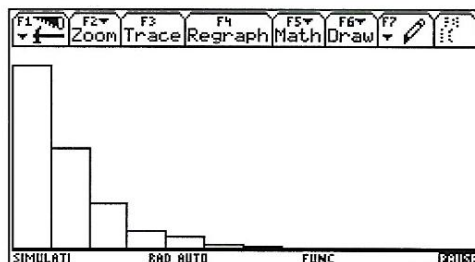
Shows the maximum Run of each try. We can see that obviously the average MaxRun converges.

But what is the limit?



4: RunNumbDist

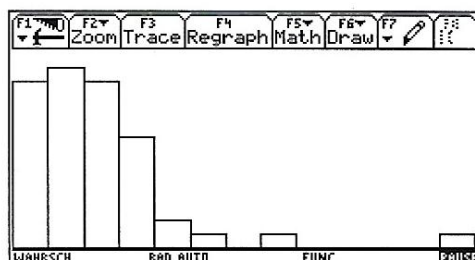
This is the frequency of the numbers of runs in one experiment. (I'll try to add labels!)



5: RunLenDist

shows the frequencies of the collected runlengths of all experiments. (Runs with length 1, length 2, ...)

And finally

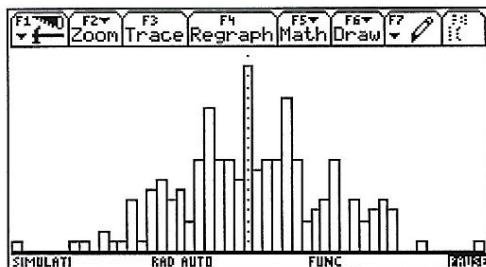


6: MaxRunDist

offers the frequency of the maximum runs.

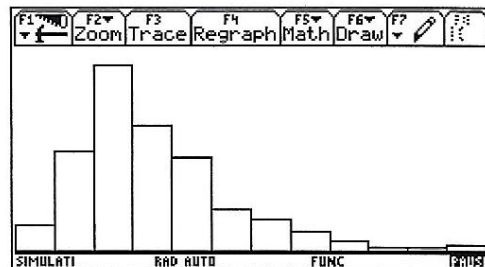
Taking 200 experiments with 200 tosses each we receive histograms which remind us on other distributions. But that takes some time, of course.

RunNumbDist



and

MaxRunDist



[1] Wilfried Rohm, Statistik mit Zufallszahlen aus AMMU 4, May 1994

[2] Arthur Engel, Wahrscheinlichkeitsrechnung und Statistik 1, Klett Studienbücher

This was my simulation on the TI-92. Nowadays in 2018 we would like to perform this nice – maybe introductory – example for probability theory with TI-Nspire.

You are invited to do a project in your classroom using the program *runtest*.

```
runtest(500)

Throws: T T T T T T H T T H H T H T T T T H T H T T H H H H H H T T H H T H T T H H T T T H
T H H T H T T H H T T T T T T T H T H T H H H T H H T H T H H H H T T H T H T H H H H T
T H H T H T H H T T H H H H T H T T T T T T H T T T T H H T T H T T H T T H T T H T T H
T H T H H H T T T H H H H H T H H H T H H H H H T T H H T T H H H H T T T H T T T
T T H T H H H H T T T H H H T T T T H H T H T T T T H T H H H H T T T H T H T H H T T H
H T T T T H T T H T H H H H H H T H H T H T H H H H H T H T H H H H T H T T T T T T H
H H T T H T H H T H T H T T H T H T H H H H H H H H H T T H T H T H T T H H T T H T H H
T T T T H T H T T T T H H T H H H H T H H T T T H H H T T T T H T T T T H T H T T H
H T H H H T H T T H H T T H H H H T T H H H T T T T H H T H T T H H H H H T T T H H T
T T H H H T T H T T H H T H T H T H H T H H T T H T H H H T H H H H H H H H T H T H T
H H T T H T T H H T T H T H H T H H H T T T H H T T T H T H H H T
```

Number of Heads: 258 = 51.6%

Number of Runs: 253

Maximum Runlength: 10

Distribution of Runs:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
130	67	24	16	7	3	3	2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Done

Lets assume that 35 students simulate 100 tosses each (take care that they use different *randseeds*). Each of them has executed *runtest*(100).

I recommend to collect the data from their screens and then perform the graphical evaluation using the statistic tools. So you can practise presentation of data. In this way we can turn a deficiency of the Nspire into a benefit. This

What's the deficiency? TI-Nspire does not allow programming the plots from within the program – which can be done with TI-92 and Voyage 200 as presented above.

The data are collected in lists for further use in the Lists & Spreadsheet- and Data & Statistics- Application of the TI-Nspire.

These might be the collected data of 35 students:

```
perchl:= {50.,59.,50.,50.,49.,48.,49.,55.,49.,57.,50.,52.,56.,47.,35.,50.,55.,46.,51.,56.,49.,45.,47.,53.,42.,57.}
{50.00,59.00,50.00,50.00,49.00,48.00,49.00,55.00,49.00,57.00,50.00,52.00,56.00,47.00,35.00,50.00,55.00,46.00,51.00,56.00,49.00,45.00,47.00,53.00,42.00,57.00}

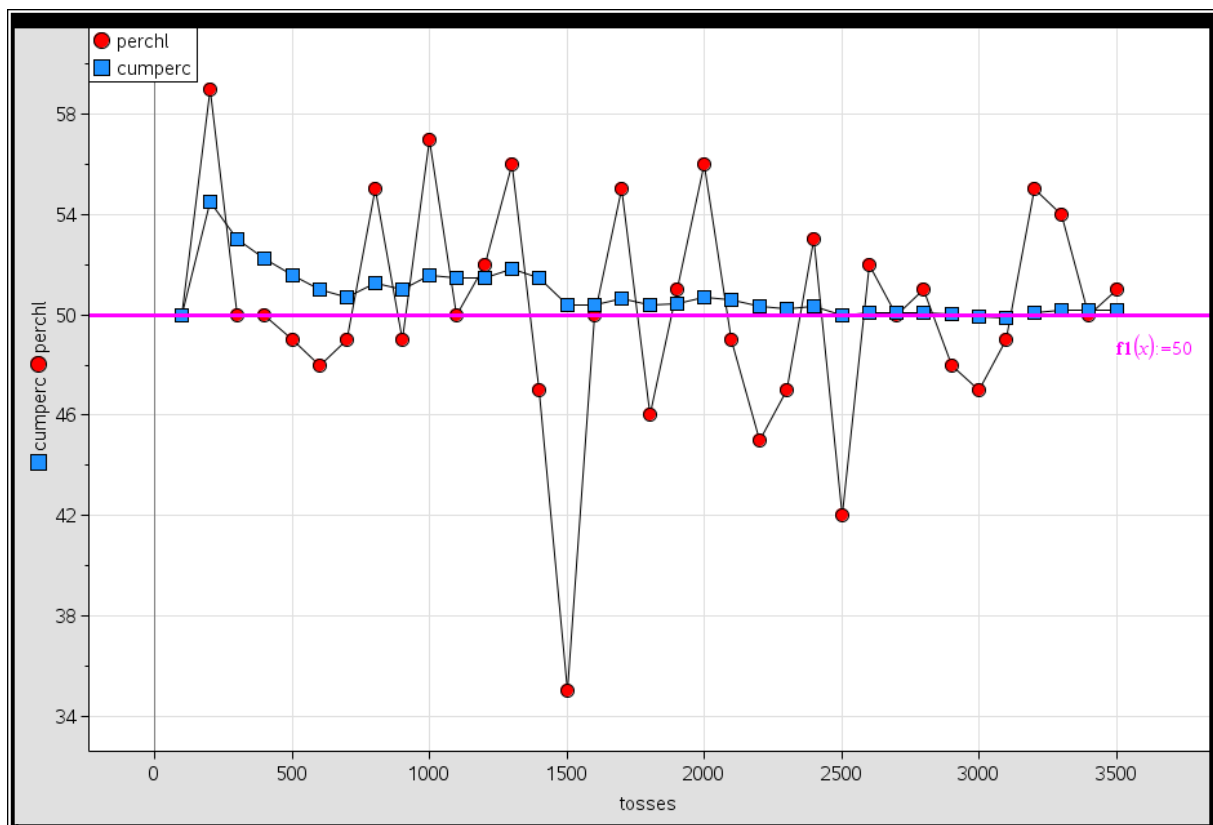
nrurl:= {48,61,50,47,51,51,44,52,51,57,54,45,46,47,53,51,51,57,51,49,49,59,43,46,51,60,55,51,54,49,49,60,42}
{48,61,50,47,51,51,44,52,51,57,54,45,46,47,53,51,51,57,51,49,49,59,43,46,51,60,55,51,54,49,49,60,42,53,42}

mrurl:= {8,6,8,7,6,8,7,7,5,8,5,9,7,5,7,5,5,7,6,7,5,5,6,6,6,5,5,11,6,10,6,6,8,9,8}
{8,6,8,7,6,8,7,7,5,8,5,9,7,5,7,5,5,7,6,7,5,5,6,6,6,5,5,11,6,10,6,6,8,9,8}

runld:= {881,469,214,120,51,23,9,8,2,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
{881,469,214,120,51,23,9,8,2,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}

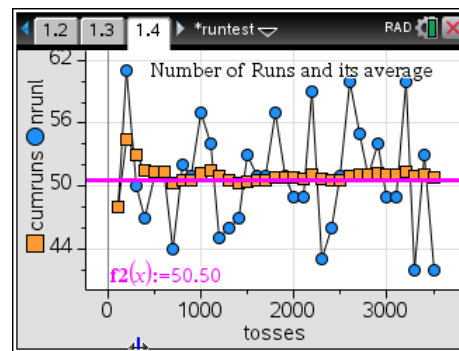
tosses:=seq(100-k,k,1,35)
{100,200,300,400,500,600,700,800,900,1000,1100,1200,1300,1400,1500,1600,1700,1800,1900,2000,2100,2200,2300,2400,2500,2600,2700,2800,2900,3000,3100,3200,3300,3400,3500}
```

First of all we can demonstrate that the number of HEADS tend to 50% of the tosses:



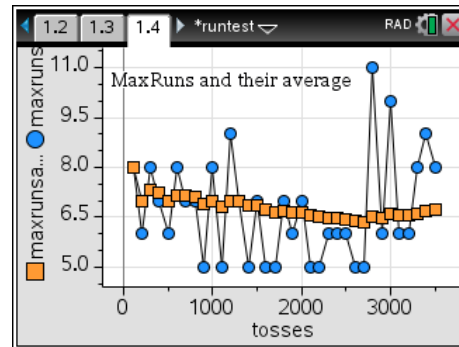
For saving space are the next screen shots from the handheld.

This is Option 2 from above, the number of RUNS together with its convergence towards 50.50.



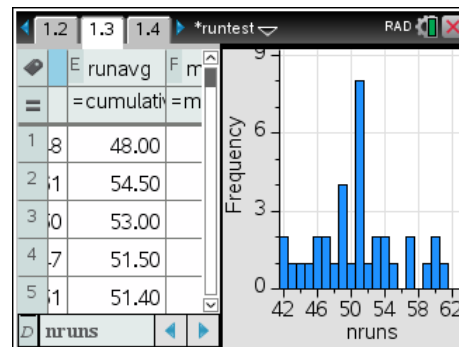
Option 3 from above shows the Maximum Runs and their mean (???).

Any idea?



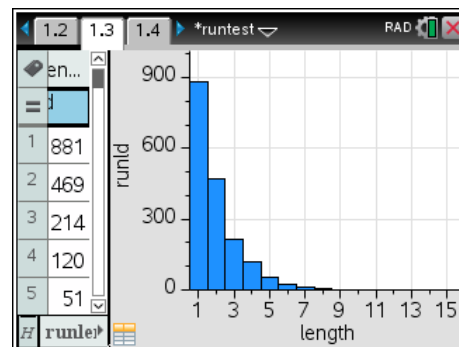
Option 4 presents the distribution of the number of the RUNS.

This will become clearer choosing more tosses and experiments (see next page).



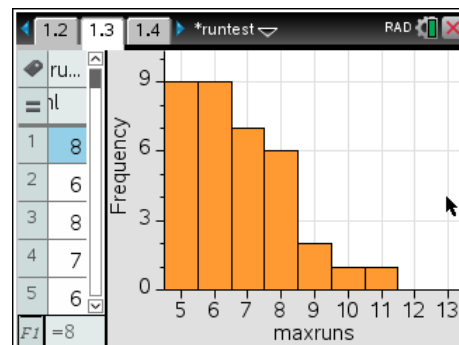
Option 5 presents the distribution of the lengths of all RUNS for all series of tosses.

Any idea for a describing function?



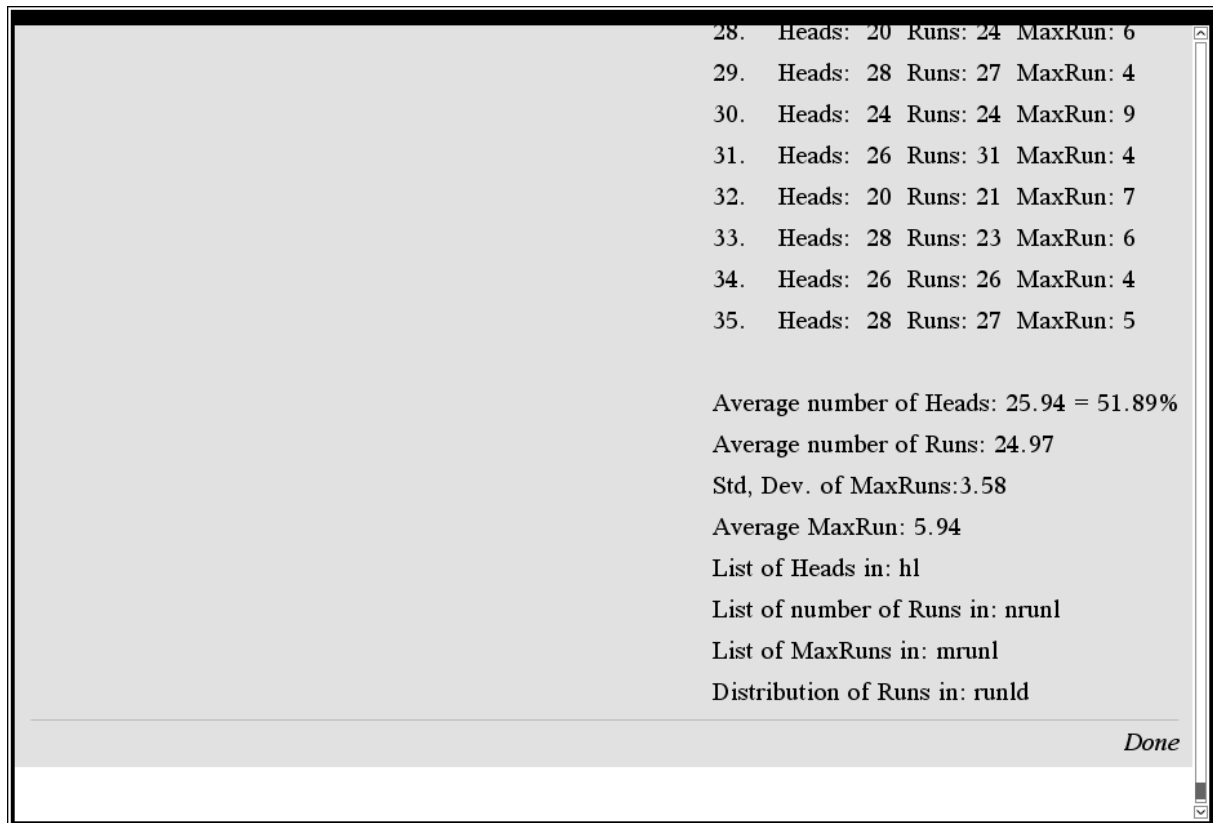
Finally, option 6 is showing the frequencies of the maximum Runs.

Again, the same question: which function lies behind the frequencies?



My program *runtests*(*number of series, tosses per serie*) allows experiments with greater numbers:

runtests(35,50) simulates the 35 student class in one step:

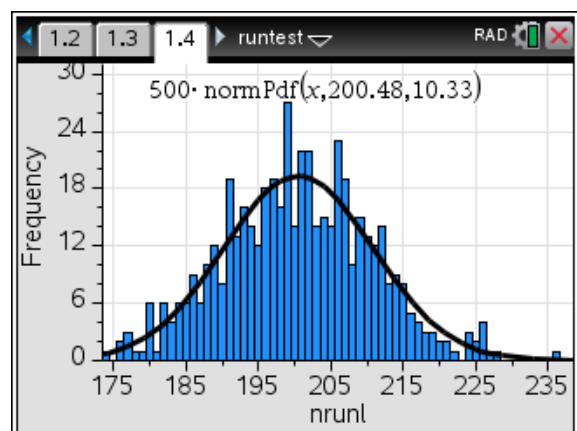
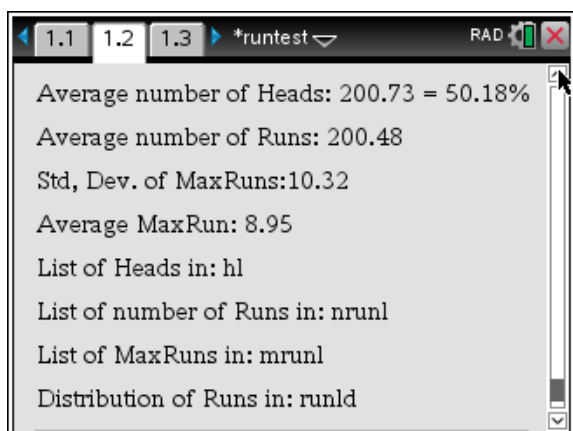


According [2] (Arthur Engel) the standard deviation of the number of Runs is given by

$$\sigma = \frac{1}{2} \sqrt{n-1}.$$

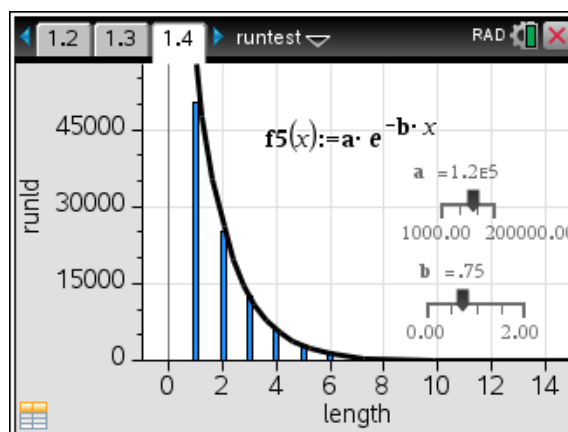
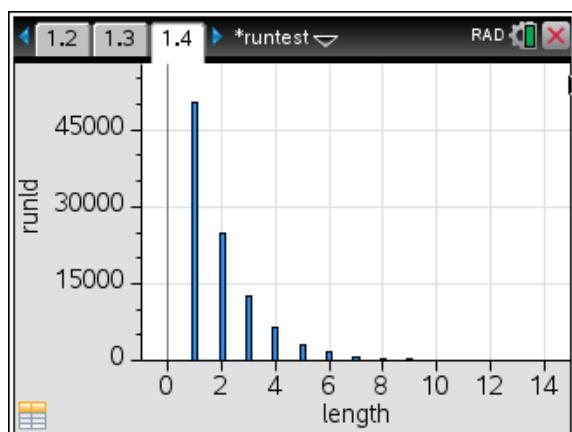
For $n = 50$ this is 3.50. As you can see here and in the next simulation as well, my outcome confirms the formula.

Finally, I execute 500 experiments with 400 coin tosses each. It works very fast on the PC:



We can recognize the bell-shaped normal distribution.

On the next page I try to approximate the distribution of the run lengths by an exponential function.



DERIVE and TI-92 User Forum

Here is a comment on Fritz Tinhof's problem with a logistic regression on the TI-92 (DNL#39, page 5). There were also some answers from TI (David Stoutemyer and Michelle Miller) which will follow Alfred's answer. Josef

Hallo Fritz !

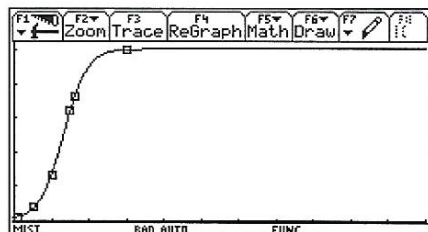
This is not a solution of your problem, I couldn't find one, but perhaps an interesting item.

	F1	F2	F3	F4	F5	F6	F7
	Plot	Setup	Cell	Header	Calc	Util	Stat
DATA							
2	c1	c2	c3	c4	c5		
3	5	4000					
4	10	13500					
5	15	32000					
6	16	36100					
7	30	50000					
8							

If you change the value 100 to 30 in the first column and choose logistic at the Calc menu it works and you will get the solution as shown above.

	F1	F2	F3	F4	F5	F6	F7
	Plot	Setup	Cell	Header	Calc	Util	Stat
DATA							
2	c1	c2	c3	c4	c5		
3	5	a	b	c	d		
4	10						
5	15						
6	16						
7	30						
8							

The grafik looks good too. On the right window you can see the function



If you change to 50 there will be still the same error message.

I can't give you really an answer why, but perhaps this short example may help you.

	F1	F2	F3	F4	F5	F6	F7
	Plot	Setup	Cell	Header	Calc	Util	Stat
DATA							
2	c1	c2	c3	c4	c5		
3	5	a	b	c	d		
4	10						
5	15						
6	16						
7	30						
8							

Regards Alfred Eisler, Tulln, Austria

David Stoutemyer, Texas Instruments

Unfortunately, I wasn't the one who wrote either the TI-92+ logistic regression or the linear system solver. I do know that:

1. There is an optional tolerance that can cause a singular-matrix error when Gaussian elimination is forced to use a relatively small pivot.
 2. Even linear regression equations can be quite nearly singular.
 3. Non-linear regression equations such as the logistic one often start from a guess, and a poor guess could cause an ill-conditioned matrix even though the matrix associated with the final guess is well conditioned.
- I am sorry that I can't be of more help on this example. Perhaps someone else at TI will know more.

Michelle Miller, Texas Instruments

Hello Josef,

I have received some more answers from our algorithm team. Since I do not know German, I am not quite sure exactly what your colleague wants to know, but here are a couple facts in addition to David's response. Please let me know if your colleague wanted some other information or explanation.

1. The algorithms used by the TI-83 Plus and the TI-89/92 Plus basecodes are different. Therefore, they can produce different answers, like the example provided by your colleague. The 89/92 basecode uses a 3-parameter model, the TI-83 Plus (and the Stats app for the 89/92) have a 4-parameter model.
2. As I mentioned in another email, there is an application for the TI-89/92 Plus called Statistics with List Editor (which is free on our website) that does include the TI-83 Plus's algorithm, Logist83, as a choice. This may help teachers if they are trying to get the same answer for a Logistic model on two different TI calculators.
3. There are some noted cases where the Logist83 model on the 89/92 Stats app and the TI-83 Plus model can still produce different results (for example, a Singular Matrix error on the 89/92), but apparently in these cases the data sets are usually not very "logistic-like".

Best Regards,

Michelle

Volker Loose, Germany

Hello all,

using DfW5 I wrote a function $\text{schrV}(f, g, a, b, d) := \text{VECTOR}([i, f(i); i, g(i)], i, a, b, d)$.

Approximating $\text{schrV}(f(x), g(x), -1, 1, 0.2)$ with $f(x) := x^2 - 1$ and $g(x) := -x^2 + 1$.

I got

$$\#5: \begin{bmatrix} -1 & f(-1) \\ -1 & g(-1) \end{bmatrix}, \begin{bmatrix} -0.8 & f(-0.8) \\ -0.8 & g(-0.8) \end{bmatrix}, \begin{bmatrix} -0.6 & f(-0.6) \\ -0.6 & g(-0.6) \end{bmatrix}, \begin{bmatrix} -0.4 & f(-0.4) \\ -0.4 & g(-0.4) \end{bmatrix}.$$

Approximating this I got

$$\#6: \begin{bmatrix} -1 & 0 \\ -1 & 0 \end{bmatrix}, \begin{bmatrix} -0.8 & -0.36 \\ -0.8 & 0.36 \end{bmatrix}, \begin{bmatrix} -0.6 & -0.64 \\ -0.6 & 0.64 \end{bmatrix}, \dots \text{the result I wanted. Why don't I get this result already in the first step?}$$

Volker Loose

Dave Stenenga, Honolulu, Hawaii

I suggest you modify your definition to:

$\text{schrV}(u, v, x, a, b, d) := \text{VECTOR}([i, \text{SUBST}(u, x, i); i, \text{SUBST}(v, x, i)], i, a, b, d)$

and then approximate: $\text{schrV}(x^2 - 1, x^2 + 1, x, -1, 1, .2)$

Al Rich, Honolulu, Hawaii

In Derive, only expressions, NOT functions, can be passed as arguments to user-defined functions. Therefore, the variable needs to be included along with the expressions in a function's formal argument list. For example, if your function is defined as

$$\#7: \text{schrV}(u, v, x, a, b, d) := \text{VECTOR}\left(\begin{bmatrix} x & u \\ x & v \end{bmatrix}, x, a, b, d\right)$$

$$\#8: [f(x) := x^2 - 1, g(x) := -x^2 + 1]$$

$$\#9: \text{schrV}(f(x), g(x), x, -1, 1, 0.2)$$

approximating either of the expressions $\text{schrV}(f(x), g(x), x, -1, 1, 0.2)$ or $\text{schrV}(f(y), g(y), y, -1, 1, 0.2)$ gives the approximated matrix that you desire.