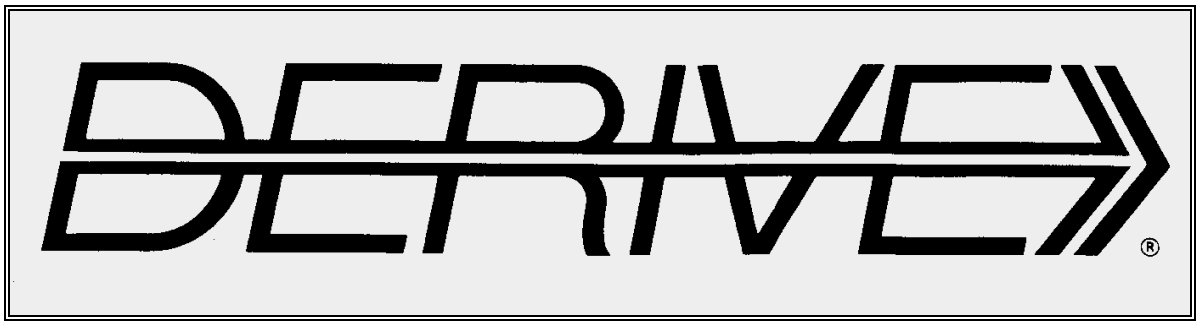
THE DERIVE - NEWSLETTER #75

ISSN 1990-7079

THE BULLETIN OF THE



USER GROUP

+ CAS-TI

Contents:

October 2009

Interesting and recommended websites:

Barry Kissane (Australia) has a great website which collects many promising links. I strongly recommend visiting:

http://wwwstaff.murdoch.edu.au/~kissane/pd/internetmaths.htm

You can download the Proceedings of **CAME 2009**. Thanks to *Djordje Kadijevich* from Serbia.

http://www.megatrend.edu.rs/came_files/CAME%202009-Proceedings.pdf

DUG-Member *René Hugelshofer* (Switzerland) informed about a new 65 pages pdf-paper on quadratics treated with TI-NspireCAS (in German):

Quadratische Funktionen und Gleichungen mit CAS

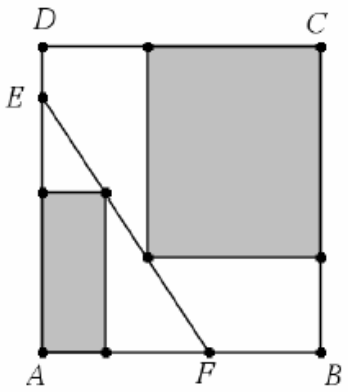
by Benno Frei, René Hugelshofer and Robert Märki

Aufgabe 4.14

Von einer teuren Marmorplatte ist bei der Bearbeitung ein Stück abgebrochen. Die Bruchlinie EF ist ein Geradenstück. Um den Schaden möglichst klein zu halten, schneiden wir aus den beiden Bruchstücken jeweils auf die skizzierte Weise eine rechteckige Platte so aus, dass die Summe der Flächeninhalte möglichst gross wird.

$|BC| = 160$ cm ; $|DC| = 100$ cm
$|AE| = 120$ cm ; $|AF| = 60$ cm

Wie viel Abfall haben wir mindestens?



This paper – and many others can be downloaded from:

http://www.ti-unterrichtsmaterialien.net

DUG-Member *Eberhard Lehmann* published his

Matrizenrechnung, Anwendungen Teil 2

On 352 pages Eberhard demonstrates many applications of matrices reaching from mappings to Markov chains. Eberhard uses TI-Nspire, Voyage 200, DERIVE and his own software ANIMATO. More details can be found at his website

http://home.snafu.de/mirza

If you know about interesting websites and publications then please inform me for sharing the information in our community. Josef

Dear DUG Members,

Welcome to DNL#75 – we can have a little celebration. We need "only" 25 more DNLs to reach the 100!!

I´d like to welcome also a couple of new DUG-Members. It is great that our world wide community is still growing despite the fact that DERIVE is not available any longer. I was very happy receiving Danny Ross Lunsford´s short note on a request which was published in the DERIVE-NEWS mailing list (see page 44).

There are not so many contributions in this newsletter as usual. You will see that Michel´s and José Luis´ articles – both are presentations from ACA 09 – are very extended. It will need the next DNL, too, in order to present the missing papers.

This DNL75-pdf-file has 2 MB because I included the ready made pdf-file of José Luis Galan´s contribution on RANDOM SAMPLES. It would have been too difficult to rewrite it in WORD including all his great illustrations.

In addition to these two papers I included another project for students provided by Roland Schröder. The Dog & Biker problem is a nice application of a recursive procedure and gives the chance to demonstrate students the power of the ITERATES-function of DERIVE.

At the other hand recursive procedures are very suitable for spreadsheet programs. So I tried to transfer this problem to TI-NspireCAS and could implement some kind of animation. We – Roland Schröder and I – have still the problem to find the locus of the dog´s jumps. It would be great if somebody could present the solution.

Tania Koller´s valuable hint (User Forum) demonstrates that students of secondary schools are very familiar with hard- and software and they find solutions for occuring problems. Many thanks to Tania and her bright students.

Best regards as ever

Finally I´d like to remind you once more on our next conference which will be held in Andalusia, Spain. You can find how to submit and details about registration within the next few days.

The Conference website is www.time2010.uma.es.

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

Editor: Mag. Josef Böhm
D´Lust 1, A-3042 Würmla
Austria
Phone: ++43-06604070480
e-mail: nojo.boehm@pgv.at

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

Next issue: December 2009
Deadline 15 November 2009

Preview: Contributions waiting to be published

Some simulations of Random Experiments, J. Böhm, AUT, Lorenz Kopp, GER
Wonderful World of Pedal Curves, J. Böhm
Tools for 3D-Problems, P. Lüke-Rosendahl, GER
Financial Mathematics 4, M. R. Phillips
Hill-Encription, J. Böhm
Simulating a Graphing Calculator in *DERIVE*, J. Böhm
Henon, Mira, Gumowski & Co, J. Böhm
Do you know this? Cabri & CAS on PC and Handheld, W. Wegscheider, AUT
Steiner Point, P. Lüke-Rosendahl, GER
Overcoming Branch & Bound by Simulation, J. Böhm, AUT
Diophantine Polynomials, D. E. McDougall, Canada
Graphics World, Currency Change, P. Charland, CAN
Cubics, Quartics – interesting features, T. Koller & J. Böhm
Logos of Companies as an Inspiration for Math Teaching
Exciting Surfaces in the FAZ / Pierre Charland´s Graphics Gallery
BooleanPlots.mth, P. Schofield, UK
Old traditional examples for a CAS – what´s new? J. Böhm, AUT
Truth Tables on the TI, M. R. Phillips
Advanced Regression Routines for the TIs, M. R. Phillips
Where oh Where is IT? (GPS with CAS), C. & P. Leinbach, USA
Embroidery Patterns, H. Ludwig, GER
Mandelbrot and Newton with *DERIVE*, Roman Hašek, CZ
Snail-shells, Piotr Trebisz, GER
A Conics-Explorer, J. Böhm, AUT
Coding Theory for the Classroom?, J. Böhm, AUT
Tutorials for the NSpireCAS, G. Herweyers, BEL
Some Projects with Students, R. Schröder, GER
Runge-Kutta Unvealed, J. Böhm, AUT
The Horror Octahedron, W. Alvermann, GER
RK6, Heinrich Ludwig, GER

and others

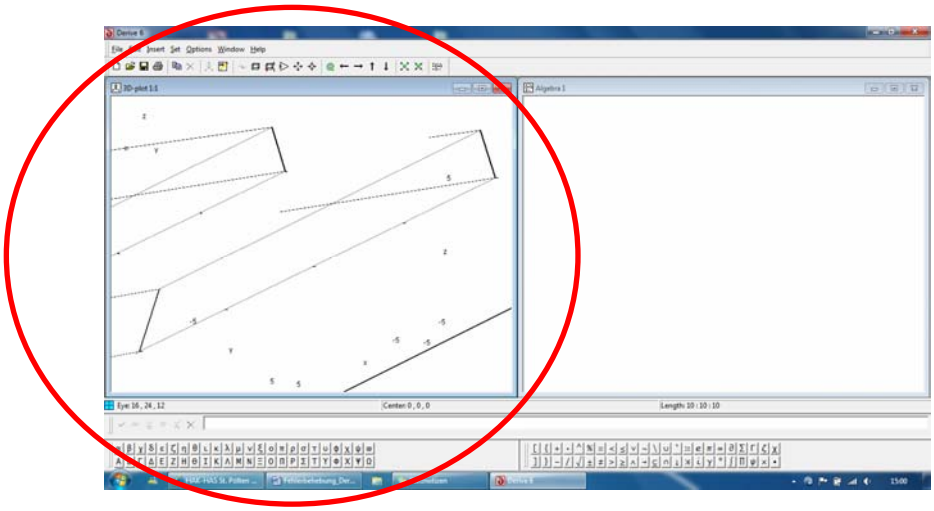
Impressum:
Medieninhaber: *DERIVE* User Group, A-3042 Würmla, D´Lust 1, AUSTRIA
Richtung: Fachzeitschrift
Herausgeber: Mag.Josef Böhm

Tania Koller, Vienna, Austria

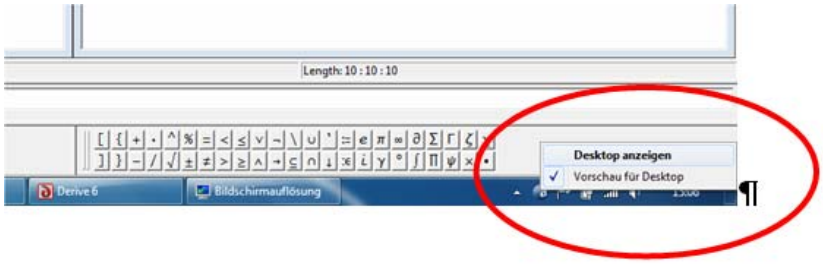
Dear Josef,
Last week we wanted to produce a 3D-representation with DERIVE and four students found a very strange distorted view on their screens. It could not be caused by VISTA because all other screens (also VISTA) were ok. Fortunately I have some bright and eager students, so I can send not only the riddle but also its solution which was found by one of my students in the weekend. Maybe that this could help other users, too.
Regards
Tania

Correction of 3D-plots on Notebooks (Windows 7)

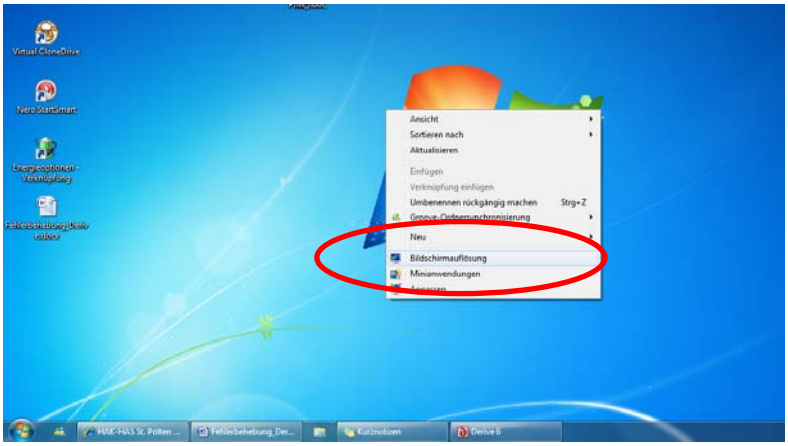
eg. with a resolution of 1366 x 768



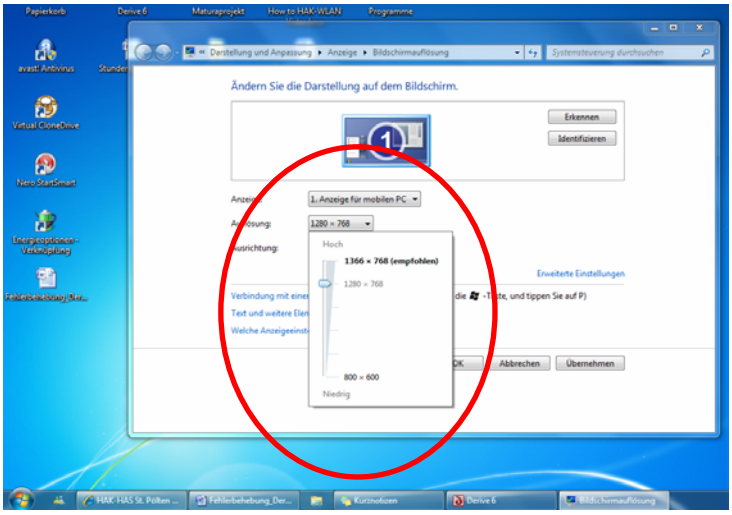
1. Show Desktop – minimze all programs in order to show the desktop
   (shortcut WINDOWS + D) or rightclick on the right border of the systray – Show Desktop



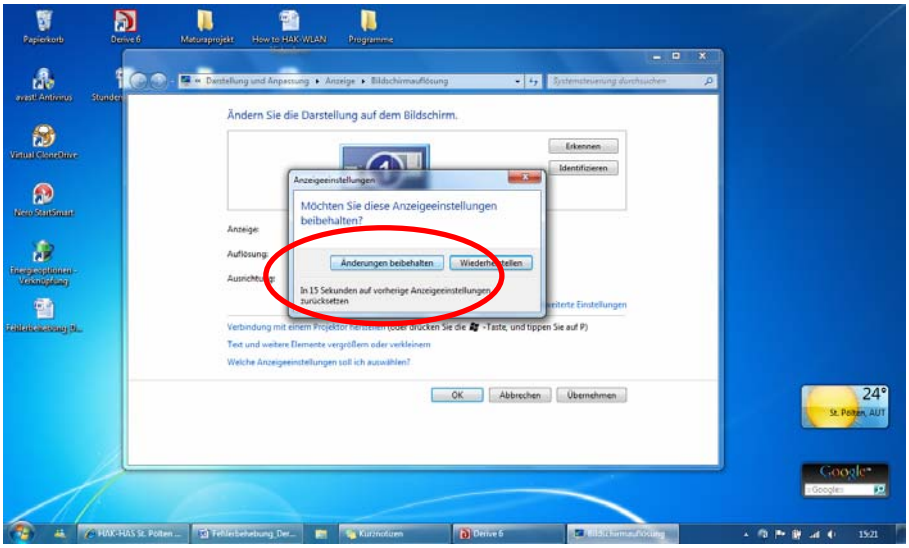
2. Rightclick on the Desktop – Resolution



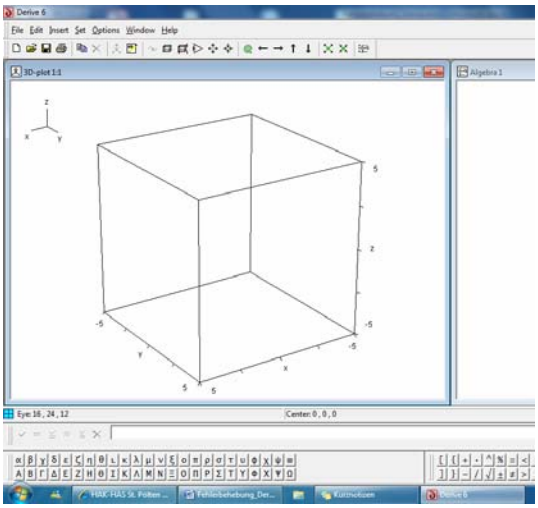
3. Resolution – Click on the Dropdown Menu and set Resolution to 1280 x 768.



4. Click on „Keep Changes“ (two black stripes appear on the screen).



5. Open a new 3D-Plot Window – and now it should look fine!

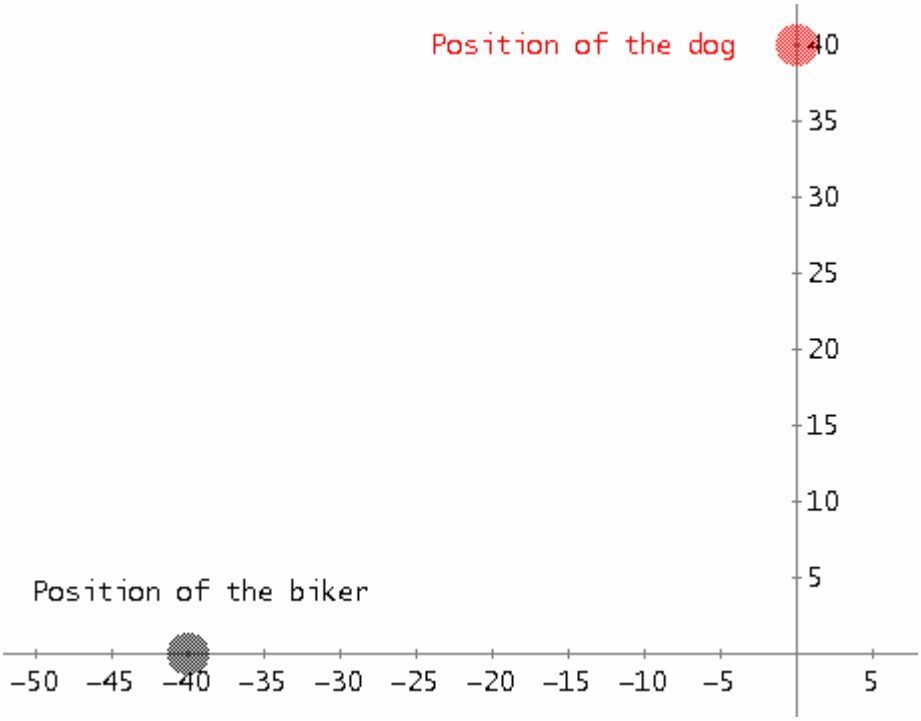


Many thanks to Philipp Wietter, student of Tania Koller at HAK St. Pölten.

Dog and Biker

Roland Schröder, Celle, Germany

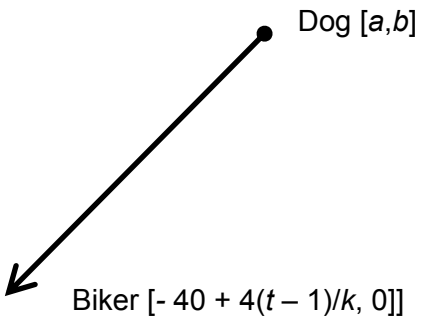
There are dogs which become aggressive when they discover a biker. We will not investigate why this is the case. But we will discuss the question: what is the chance of a biker to escape an aggressive dog? We can imagine the situation as follows:



The biker moves with constant velocity starting from point (-40,0) in direction to the origin and goes ahead in this direction to escape the dog if possible. The dog is watching his surroundings in position (0,40) when he is discovering the biker at (-40,0). The dog is silly enough not to bar the biker´s way but he undertakes 4m jumps (with also constant velocity) in the direction of the – changing – position of the biker. If both have the same velocity the dog will never reach the biker because of his wrong strategy.

If the dog is faster than the biker he will win but the dog gives up “hunting” the biker after 20 jumps. What is the ratio of velocities giving the biker a chance to escape?

We choose the time steps so that one jump of the dog takes one step. The velocity of the dog is the k-fold of the biker´s velocity with $k > 1$ to give the dog a chance. The positions of the dog and the biker at time t are illustrated by the following sketch:



The position of the biker only changes in its *x*-coordinate, which is described by the DERIVE-expression

#1: – 40 + 4·t/k.

The segment (the vector) from Dog to Biker is given by [#1,0] – [a,b] = [#1 – a, – b]. The length of this segment – this is one jump of the dog – should be 4 m. So we have to normalize the segment (= divide by its length) and multiply by 4. We use the factor:

#2: 4/ABS([#1 – a, -b]) = #2: 4/ABS([-40 + 4·t/k – a, –b]).

The jump starting from [*a*, *b*] is presented by #2 · [#1 – a, – b]. This jump starts at [a, b] and ends at [a, b] + #2 · [#1 – a, – b] = [a + #2 · (#1 – a), b – #2·b]. We need a third component in this vector, *t* + 1, which counts the jumps of the dog. We find by recursion the immediate successor element of [a, b, t] in [a + #2 · (#1 – a), b – #2·b, t + 1]. The ITERATES-command is the perfect tool to perform the recursion:

#3: D(k):=ITERATES([a + #2 · (#1 – a), b – #2·b, t + 1], [a, b, t], [0, 40, 0], 20).

For plotting we have to remove the time component. The sequence of points – landing points of the dog – are given by:

#4: Dog(k) := VECTOR([(D(k))↓n↓1, (D(k))↓n↓2], n, 21).

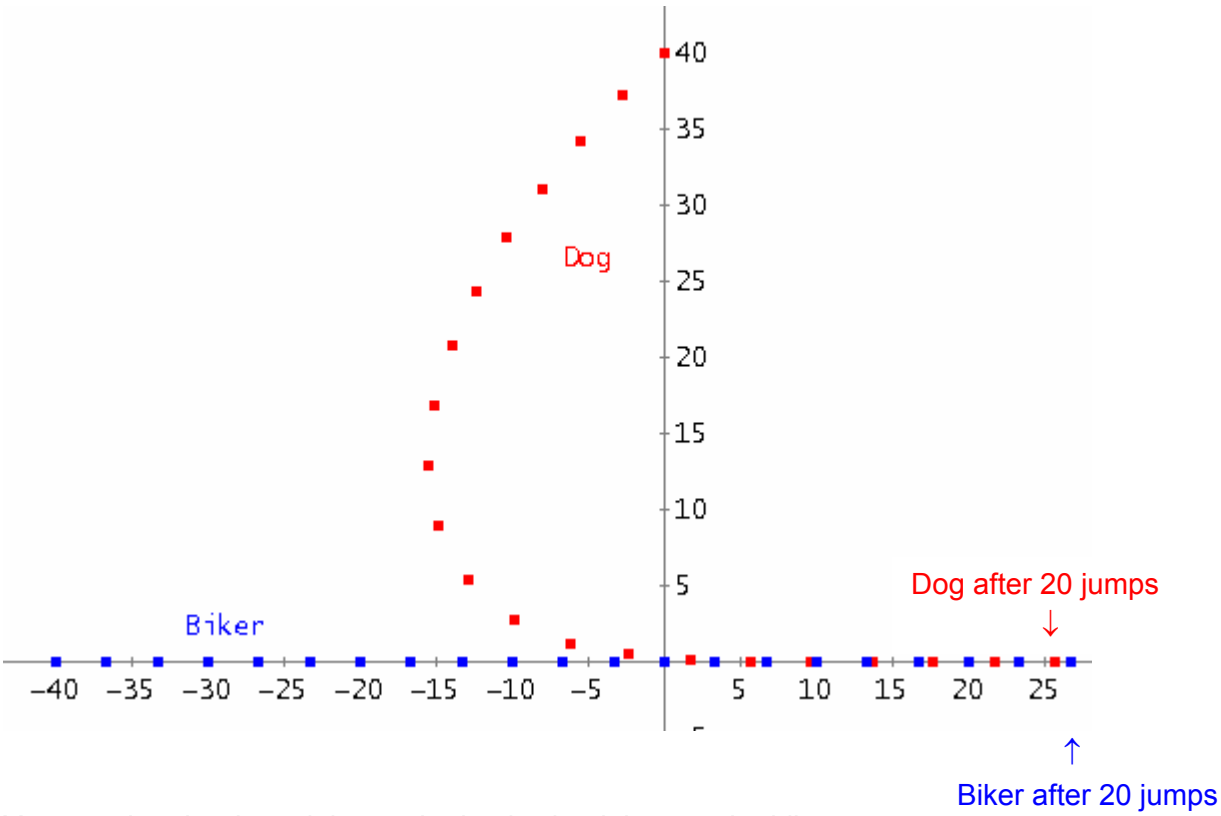
But we want to see the position of the biker, too. So we enter

#5: Bike(k) := VECTOR([-40 + 4·(t – 1)/k, 0], t, 21).

#6: Dog(1.2)

#7: Bike(1.2)

Now plot #6 and #7 (for *k* = 1.2):

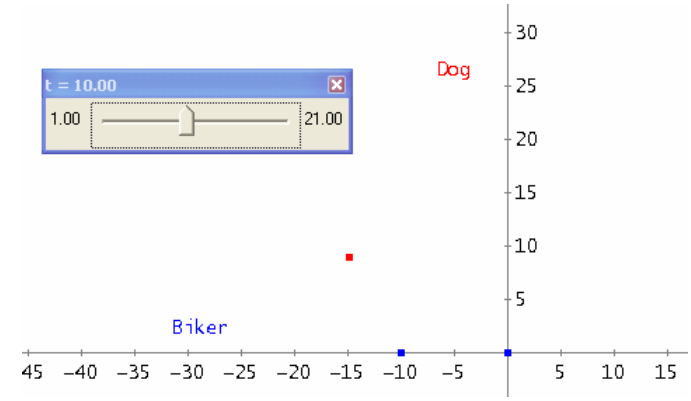


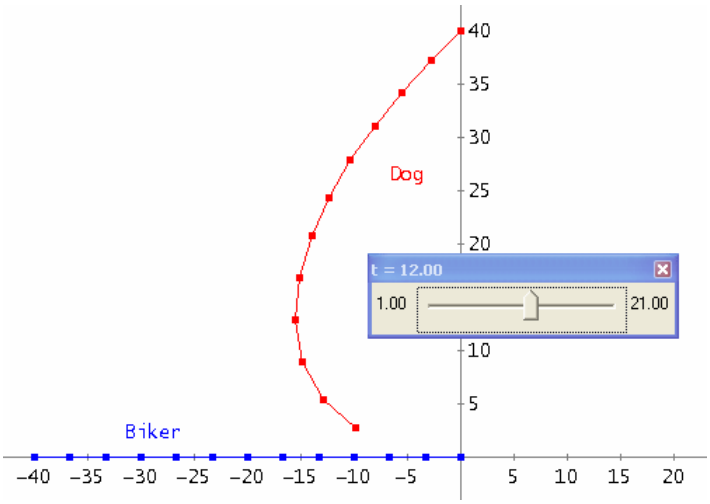
You see that the dog might get the back wheel, but not the biker.

We can use a utility from DNL#62 to perform something like an animation – we can imagine following the jumps:

```
#8:  movepts(list) := VECTOR(list[i]·IF(t = i, 1, ∞), i, DIM(list))

#9:  tracepts(list) := VECTOR(list[i]·IF(t ≥ i, 1, ∞), i, DIM(list))

#10: [movepts(Dog(1.2)), movepts(Bike(1.2))]

#11: [tracepts(Dog(1.2)), tracepts(Bike(1.2))]
```

Introduce a slider for *t* with 1<= *t* <= 21 (20 Intervals) and now you can either perform one jump after the other (#10) or follow the trace of dog and biker (#11).





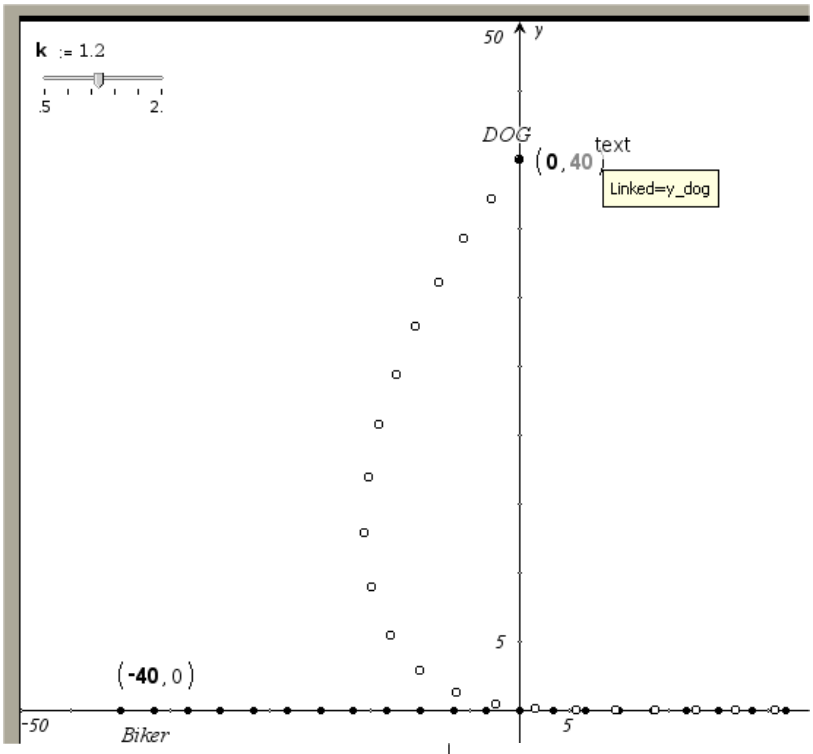
I (Josef) wanted to transfer this nice problem to TI-NSpireCAS because recursion can be performed on a spreadsheet in an easy way. I wanted to have a very flexible model with variable positions of dog and biker and including a slider for the parameter *k* to investigate its influence on the success of the dog – or the biker.

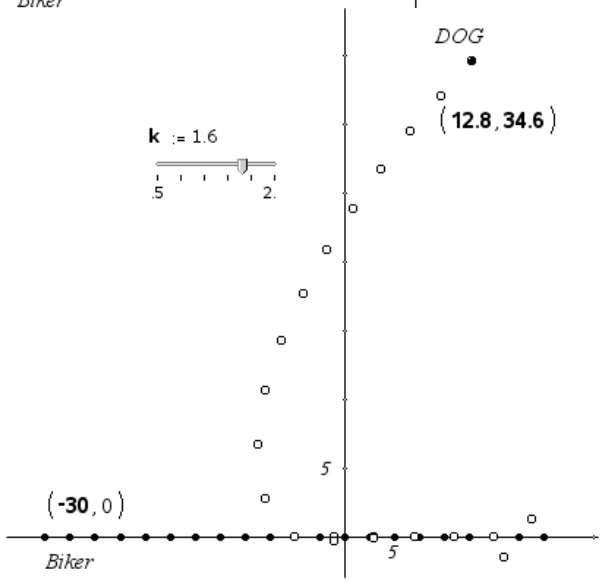
I came across one problem: the vector operations don´t seem to work in the spreadsheet application (eg. norm(x,y) for finding the length of a vector). Philippe Fortin confirmed my conjecture and gave a hint to overcome this problem. I defined a function for the length of the vector and then the spreadsheet application accepted the function.

You are invited to study the TI-Nspire-file. I started with creating the points for the dog and the biker and linking their coordinates to variables which I then used in the spreadsheet.

$le(a,b,c,d):=\text{norm}([c-a \quad d-b])$ *Done*

	A	B	C xb	D yb	E xd	F yd	G
◆							
1	1.200000...	1.000000...	-40.0000...	0.000000...	0.000000...	40.00000...	
2		2.000000...	-36.6666...	0.000000...	-2.82842...	37.17157...	
3		3.000000...	-33.3333...	0.000000...	-5.52111...	34.21363...	
4		4.000000...	-30.0000...	0.000000...	-8.04423...	31.10978...	
5		5.000000...	-26.6666...	0.000000...	-10.3506...	27.84170...	
6		6.000000...	-23.3333...	0.000000...	-12.3730...	24.39064...	
7		7.000000...	-20.0000...	0.000000...	-14.0126...	20.74208...	
8		8.000000...	-16.6666...	0.000000...	-15.1219...	16.89899...	





Application of Computer Algebra (ACA) 2009
Session: Applications and Libraries development in Derive
25-28 June 2009, École de technologie Supérieure, Montréal, Québec, Canada

Another Look at a Trusted Mathematical Assistant

Michel Beaudin
Service des enseignements généraux,
École de technologie supérieure (ETS),
1100, Notre-Dame street west,
Montréal, Québec, Canada, H3C 1K3
Email: michel.beaudin@etsmtl.ca

Abstract

• From the *DERIVE* user manual (version 3, September 1994), we can read the following: "Making mathematics more exciting and enjoyable is the driving force behind the development of the *DERIVE* program". In this talk, we will try to show how some mathematical concepts, studied by engineering students at university level – differential equations, multiple variable calculus, systems of non linear equations –, can be easily illustrated by *DERIVE*. Some will object that any other CAS could do the same: well, this is probably true but, according to us, not as quickly and naturally: "To accomplish this *DERIVE* not only has to be a tireless, powerful and knowledgeable mathematical assistant, it must be an easy, natural, and convenient tool". Consequently, time can be spent to prove some theorem or formula and the computer algebra system helps to reinforce the mathematical concepts. Our examples will also make use of new features added in the latest version of *DERIVE* (version 6.10 released in October 2004); features that were not exploited as should be − *DERIVE* has never been enough documented. But we are still convinced that *Derive 6* was "Far too good just for students"

(http://www.scientific-computing.com/scwmarapr04derive6.html).

1. Introduction

This main goal of this paper is to show how *DERIVE* can be used at university level, for teaching to engineering students. For the past 15 years, we have attended many conferences about using CAS in teaching mathematics: when *Derive* was used in a presentation, it was ─ in the majority of cases ─ for high schools/colleges level presentation. This is normal because *DERIVE*, compared to "big" CAS like MAPLE or MATHEMATICA, is very simple to use and, at lower levels, this is important. But, at university level, even if *DERIVE* is a small system compared to the big ones, it can be used to explore many avenues, in domains such differential equations and multiple variable calculus ─ where, usually, one will use another system, more "powerful". In fact, as far as undergraduate mathematics are concerned ─ the situation would probably be different at research levels ─, *DERIVE* is so simple to use that if some mathematics teacher wants to go from the blackboard to the computer ─ without having to load an already prepared file ─, *DERIVE* is an excellent choice: no waste of time by typing complicated or long commands in order to illustrate daily concepts. This is important for us: we are still teaching mathematics and we need software that will do the job correctly. But we also need software that can do heavy computations when this is needed. *DERIVE* has done the job since the last 15 years. We would like to be able to continue to use *DERIVE* for the upcoming years. Because of its discontinuation, we are afraid that these good moments are behind us: only time will tell.

2. The importance of symbolic integration of piecewise continuous functions in differential equations

If we don't pay attention to the endpoints of an interval [a, b], then the indicator function, defined by CHI(a, x, b), will be the following function:

$$\mathrm{CHI}(a,x,b)=\begin{cases}1 & \text{if } a<x<b\\ 0 & \text{elsewhere}\end{cases}$$

When you are teaching differential equations, you often need to deal with piecewise continuous functions, especially when you study Laplace transforms: each piece is defined on an interval, 2 distinct intervals don't overlap, so CHI is very useful. So, where does the CHI function play an important role? To answer this question, let's take a look at the following problems from a differential equations (DE) course. The first two ones are usually solved by taking the Laplace transform of both sides of the DE. The third one is done by computing the Fourier coefficient of the given signal or by using a table of Fourier series. The fourth one is not related with the CHI function but because it is usually solved by using the method of undetermined coefficients, this will become an opportunity to explore what *DERIVE* is doing when its "DSOLVE2" command is used.

a) Solve $y'' + 2y' + y = f(t),\ y(0) = 0,\ \ y'(0) = 0$ where f is the function in figure 1.

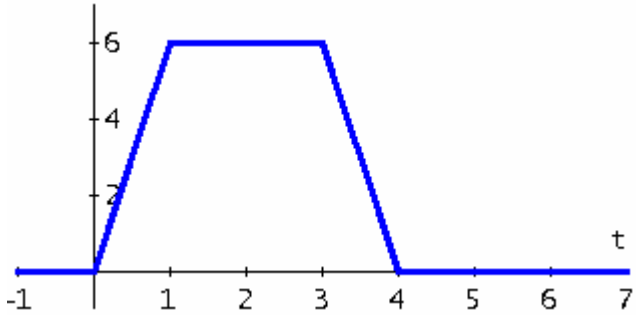


Figure 1

b) Thinking of a mass-spring problem, find the solution of

$$y'' + 4y = 50\delta(t-\pi) - 100\,\delta(t-2\pi),\quad y(0) = 10,\ \ y'(0) = 5,$$

where δ is the «Dirac delta function" (not defined in *DERIVE*).

c) Find the Fourier series of the signal ─ square wave ─ shown in figure 2. What happens to the partial sums near a point of discontinuity?

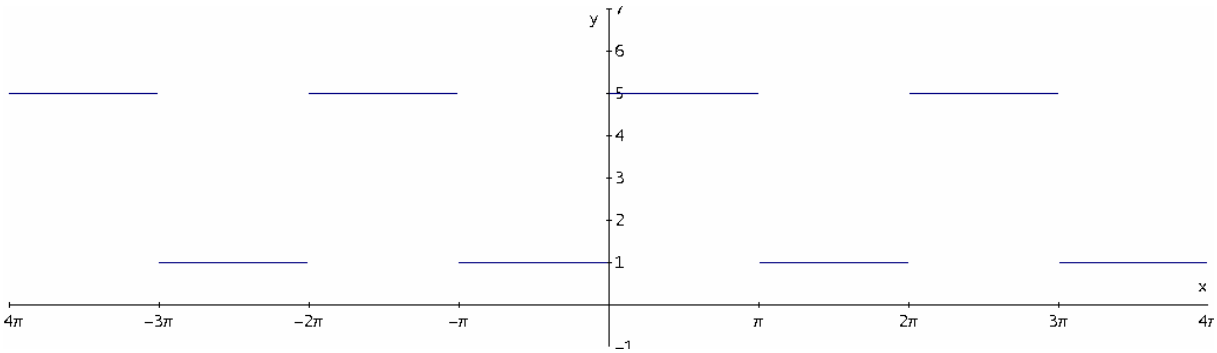


Figure 2

d) Identify the steady-state solution in the mass-spring problem governed by the DE

$$y'' + \frac{1}{5}y' + 49y = 50\cos(\omega t).$$

Here ω is some positive parameter.

Now, let us give the *"DERIVE* solutions" of each of these problems. For problem a), one first needs to note that the function in figure 1 is defined by

$$\begin{cases} 0 & \text{if } t<0 \\ 6t & \text{if } 0<t<1 \\ 6 & \text{if } 1<t<3 \\ 24-6t & \text{if } 3<t<4 \\ 0 & \text{if } t>4 \end{cases}$$

Using Laplace transform techniques, we can find the following solution:

$$y(t)=g(t)u(t)-g(t-1)u(t-1)-g(t-3)u(t-3)+g(t-4)u(t-4),$$

where $g(t)=12e^{-t}+6te^{-t}-12+6t, u(t)=\text{STEP}(t)$. Using *DERIVE*, the command "DSOLVE2_IV" does the job, because this command uses the method of variation of parameters in order to find a particular solution and the software has no problem to integrate piecewise continuous functions. So, we define f in line #1 (figure 3 below): this is the "input" and this is done with linear combination of CHI functions because the intervals don't overlap. And use the "DSOLVE2_IV" command to get the answer or "output" (no need to simplify this to get the graph if the option "simplify before plotting" is on). Of course, if one simplifies line #2, an answer involving SIGN functions will appear on the screen ─ because CHI functions are defined using STEP and STEP comes from SIGN!.

```
#1:   f := 6·t·χ(0, t, 1) + 6·χ(1, t, 3) + (24 - 6·t)·χ(3, t, 4)
#2:   DSOLVE2_IV(2, 1, f, t, 0, 0, 0)
```

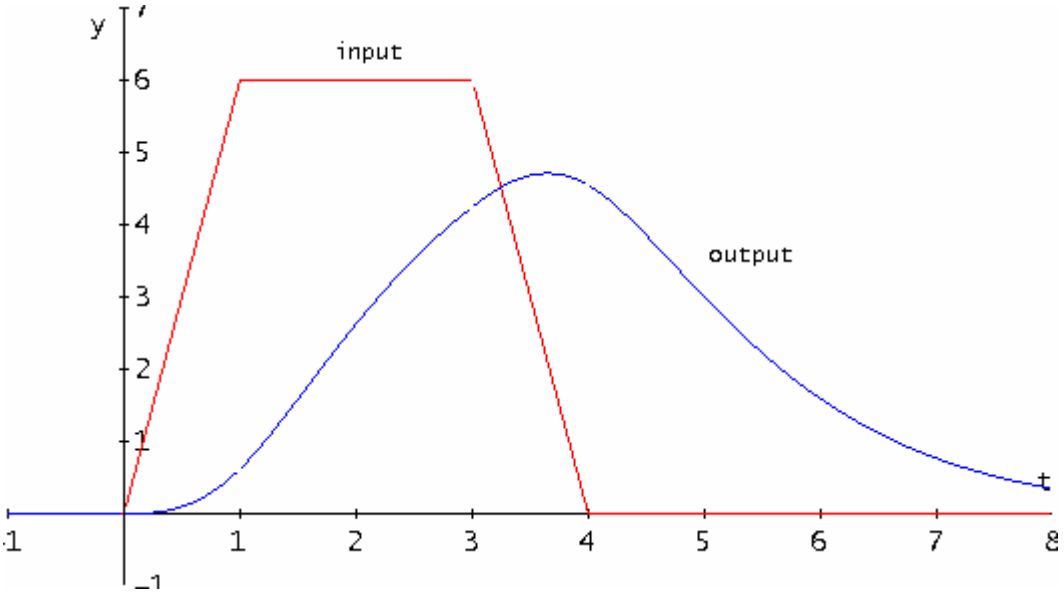


Figure 3

For problem b), students learn that an expression of the form $A\delta(t-t_0)$ means that an infinite force was applied at time $t=t_0$ with total impulse of A (N.s). The expression $A\delta(t-t_0)$ can be defined by the following limit, which does not exist in the classical way.

$$A\delta(t-t_0)=\lim_{\varepsilon\to 0^+} A\frac{1}{\varepsilon}\mathrm{CHI}(t_0,t,t_0+\varepsilon).$$

But if one solves the differential equation with $A\delta(t-t_0)$ replaced by $A\frac{1}{\varepsilon}\mathrm{CHI}(t_0,t,t_0+\varepsilon)$ and then let ε approach 0 from the right, he/she will get the right answer! Of course, the beauty of Laplace transform is that students don't have to use that kind of trick. They simply use the fact that $\delta(t-t_0)\leftrightarrow e^{-t_0 s}$ and find the solution. In order to fully appreciate the effect of the "Dirac delta function", the slider bar of *DERIVE* can be used if one wants to understand why the solution (line #5 in figure 4) is obtained by taking some limits (this has to be done "live" during a talk). The graph of the solution (line #5) is also shown after line #5 (note: there is no discontinuity!).

#1: $\text{input} := \frac{50}{a}\cdot\chi(\pi,\ t,\ \pi+a) - \frac{100}{b}\cdot\chi(2\cdot\pi,\ t,\ 2\cdot\pi+b)$

#2: `output := DSOLVE2_IV(0, 4, input, t, 0, 10, 5)`

#3: `real_ouput := lim lim output` (b→0+, a→0+)

#4: `real_ouput`

#5: $-25\cdot\text{SIGN}(t-2\cdot\pi)\cdot\text{SIN}(2\cdot t)+\frac{25\cdot\text{SIGN}(t-\pi)\cdot\text{SIN}(2\cdot t)}{2}+10\cdot\text{COS}(2\cdot t)-10\cdot\text{SIN}(2\cdot t)$

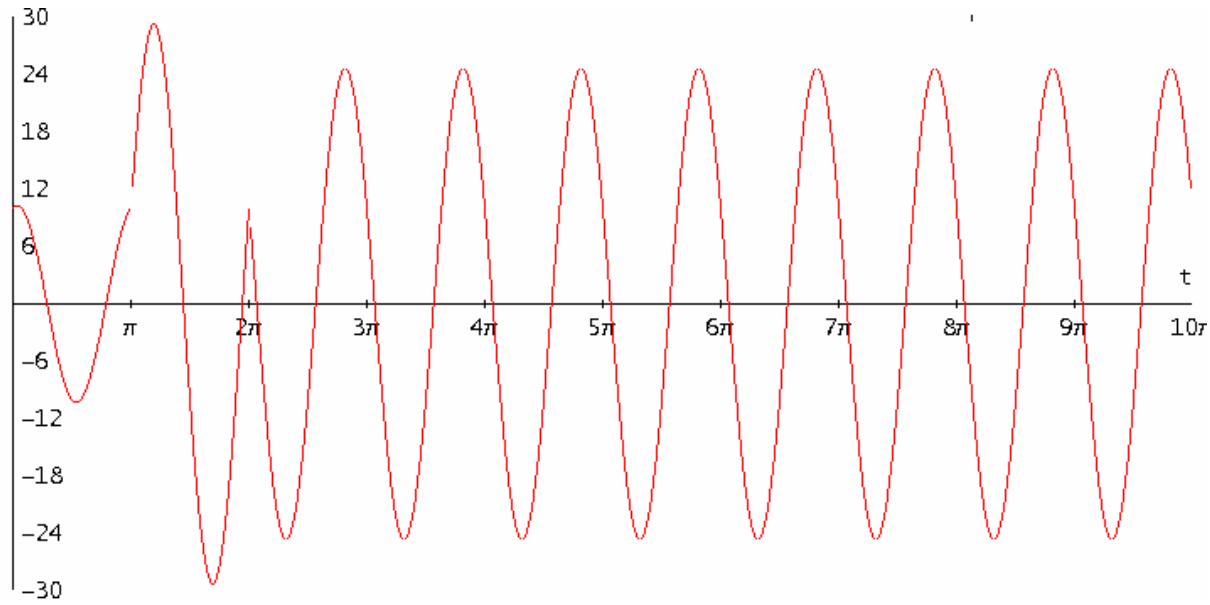


Figure 4

For problem c), because *DERIVE* has a "Fourier series" built-in function, it is quite easy to obtain partial Fourier sums. First, we define the function on its interval of definition, using CHI again and then, extend it periodically using the modulo MOD function of *DERIVE*. Let us recall (fact learned at the first international *DERIVE* conference, in Plymouth, UK, in 1994!!!) that if $g(x)$ is a defined function of the variable x, defined over the interval $[a, b]$ ─ using possibly one or many indicator functions ─, then we can extend g periodically over the entire real line, period being $b - a$, with the help of the definition $f(x) = g(\text{mod}(x - a, b - a) + a)$. A simple partial sum (line #3 and its simplification, line #4) tells us that the partial sum of order $2n - 1$ is given by an expression of the form #5. Differential calculus tells us that the first critical point after 0 is located at $x = \pi/(2n)$, it is a local maximum ─ second derivative test shows this ─ whose y-coordinate is about 5.35795949. *DERIVE* won't find the limit, as n goes to infinity, of expression #5 when $x = \pi/(2n)$. This can be done "by hand", on the blackboard, and we can show that this limit is exactly $3 + 4/\pi\,\text{SI}(\pi)$, where SI stands for Sine Integral.

#1: $g(x) := 5 \cdot \chi(0, x, \pi) + 1 \cdot \chi(\pi, x, 2 \cdot \pi)$

#2: $f(x) := g(\text{MOD}(x, 2 \cdot \pi))$

#3: $\text{FOURIER}(g(x), x, 0, 2 \cdot \pi, 5)$

#4: $$\frac{8 \cdot \text{SIN}(5 \cdot x)}{5 \cdot \pi} + \frac{8 \cdot \text{SIN}(3 \cdot x)}{3 \cdot \pi} + \frac{8 \cdot \text{SIN}(x)}{\pi} + 3$$

#5: $$s(x, n) := 3 + \frac{8}{\pi} \cdot \sum_{k=1}^{n} \frac{\text{SIN}((2 \cdot k - 1) \cdot x)}{2 \cdot k - 1}$$

#6: $$\frac{d}{dx}\, s(x, n) = \frac{4 \cdot \text{SIN}(2 \cdot n \cdot x)}{\pi \cdot \text{SIN}(x)}$$

#7: $$\text{SOLVE}\left(\frac{4 \cdot \text{SIN}(2 \cdot n \cdot x)}{\pi \cdot \text{SIN}(x)}, x\right) = \left(x = -\frac{\pi}{2 \cdot n} \text{ v } x = \frac{\pi}{2 \cdot n} \text{ v } n = 0\right)$$

#8: $$s\left(\frac{\pi}{2000}, 1000\right)$$

#9: 5.357959655

#10: $$3 + \frac{4}{\pi} \cdot \text{SI}(\pi)$$

#11: 5.357959488

Figure 5

For problem d), we said that *DERIVE* is using, within its "DSOLVE2" command, the method of variation of parameters. This gives a very ugly answer compared to the one we get if we use the method of undetermined coefficients. In fact, a particular solution of $y'' + \frac{1}{5}y' + 49y = 50\cos(\omega t)$ will be of the form $\alpha\cos(\varpi t) + \beta\sin(\varpi t)$. Simple computations ─ but we did it using *DERIVE*! ─ show that

$$\alpha = \frac{1250\cdot(49 - \omega^2)}{25\cdot\omega^4 - 2449\cdot\omega^2 + 60025} \wedge \beta = \frac{250\cdot\omega}{25\cdot\omega^4 - 2449\cdot\omega^2 + 60025}$$

What is the good point, here? If you try to obtain this particular solution from the one given by *DERIVE*'s DSOLVE2_IV(1/5, 49, 50cos(ωt), t, 0, 0, 0) command, you will need to type commands from trigonometry! Note that, because we were looking for the steady-state solution, we decided to set the initial conditions at 0. Other point: if we compute $\sqrt{\alpha^2 + \beta^2}$ — this will be useful if we want to find the maximum of the amplitude of the steady-state solution (frequency response) — using the preceding values of α and β, *DERIVE* simplifies it into $\frac{250}{\sqrt{25\varpi^4 - 2449\varpi^2 + 60025}}$ instead of $250\sqrt{\frac{1}{25\varpi^4 - 2449\varpi^2 + 60025}}$. By using the "Display Step" command, we note that *DERIVE* has recognized that the expression $25\varpi^4 - 2449\varpi^2 + 60025$ is always positive because its discriminant is negative!

3. The importance of a "multiple types" 2D plot window

Voyage 200 CAS calculator has 5 different 2D plot windows: function, parametric, polar, sequence and differential equations graphing modes are available (if we also use the 3D plot window for implicit 2D plotting, then we get 6 windows). This is correct because each window comes with particular features: for example, the function graphing window has submenus to find min/max/inflection points of a given function, without having to go back to the HOME screen and compute derivatives. Students will note that there is no "intersection point" item in the F5-Math menu of the parametric 2D plot window; they will learn why and also they will learn how important it is to have a starting point when you want to solve a non linear system of 2 equations in 2 unknowns. Having so many different 2D plot windows is very useful because each window (and each editor) is specialized. The fact that *DERIVE* has a unique 2D plot window becomes important when multiple plots are needed in the same window, especially for the *same problem*. Also (in [2]), we gave reasons of the importance of a fast 2D implicit plotter, and, more generally, we showed, using many examples, that a 2D plot window should also allow the user to put in it what is useful — if you need to plot 2 curves, one explicitly defined, the other implicitly defined, why have to borrow with complicated commands? This is what we mean by the title "multiple types" 2D plot window. Now, we present 3 examples and explain the solution for each.

a) How can we get *DERIVE* to obtain one complex solution of an equation like the following one: $2^x = x^{14} + 2$? Those who are familiar with the LambertW function of *Maple* know that there are many complex solutions for an equation of the type $a^x = x^n$.

b) According to the theorem about existence and uniqueness of a solution of a first order ODE, there is a unique solution to the problem $\frac{dy}{dx} = \frac{x^2 + e^{-x}}{6 + 2y}$, $y(0) = 2$, defined in some interval about 0. How is this solution related with the implicit curve obtained by solving the ODE as a separable one? And how does the Euler numerical method compare?

c) In *DERIVE*, level curves of a function of 2 independent variables have to be plotted in 2D, not on the surface in 3D. How can we visualize this in 3D, using the slider bar?

For problem a), *DERIVE*, in exact mode, can't solve this equation but the NSOLVE command with appropriate bounds find the 3 real solutions:

```
        x    14
NSOLVE(2  = x   - 2, x)

                                                    x = -1.066944162

        x    14
NSOLVE(2  = x   - 2, x, 0, 10)

                                                     x = 1.107074389

        x    14
NSOLVE(2  = x   - 2, x, 1.2, ∞)

                                                    x = 91.14022420
```

Figure 6

But if we are searching a complex solution, one needs to substitute for x the expression $x + iy$, and plot, in the same window, the implicit curves defined by RE(eq($x + iy$)) = 0 and IM(eq($x + iy$)) = 0 where $\text{eq}(z) = 2^z - z^{14} - 2$. The fast 2D plotter and Newton's method (for 2 variables) will give any complex solution (or Newton's method for 1 variable, starting point being complex). Finding a complex solution using the real and imaginary parts of the given equation is a nice way to use complex numbers ("$w \in \mathbb{C} = 0 \Leftrightarrow \text{RE}(w) = 0 \wedge \text{IM}(w) = 0$"). It would have been a good addition to "version 7 of *DERIVE*" to have, within the solve command, the possibility, for a system of 2 equations, to use a starting point without calling the "Newtons" function.

For problem b), let us note first that we can find a neighbourhood of the point (0, 2) ─ obviously limited by the horizontal line $y = -3$ where the derivative becomes infinite ─ where both the function $\frac{x^2 + e^{-x}}{6 + 2y}$ and the partial derivative with respect to y are continuous. So, there exists an interval about the point 0 and a unique function that satisfies the ODE and the initial condition. The "separable" function of *DERIVE* is a good choice to obtain the solution, defined implicitly by $y^2 + 6y = -e^{-x} + \frac{x^3}{3} + 17$. Now, the importance of a 2D plot window, where all kind of plots are possible, becomes clear. Solving for y ─ of course, here it is possible with the well-known second degree formula ─ and keeping the "good" branch leads to

$$\phi(x) := \frac{\sqrt{3} \cdot e^{-x/2} \cdot \left(\sqrt{\left(e^{x} \cdot (x^3 + 78) - 3\right)} - 3 \cdot \sqrt{3} \cdot e^{x/2}\right)}{3}$$

Now, in the same window, here are plots of the implicit solution, the explicit solution (function $\phi(x)$ and points generated by the Euler's numerical method: it is always good to be able to see the numerical solution (Euler's method in this case) and the exact solution (implicit and explicit when possible) in the same window.

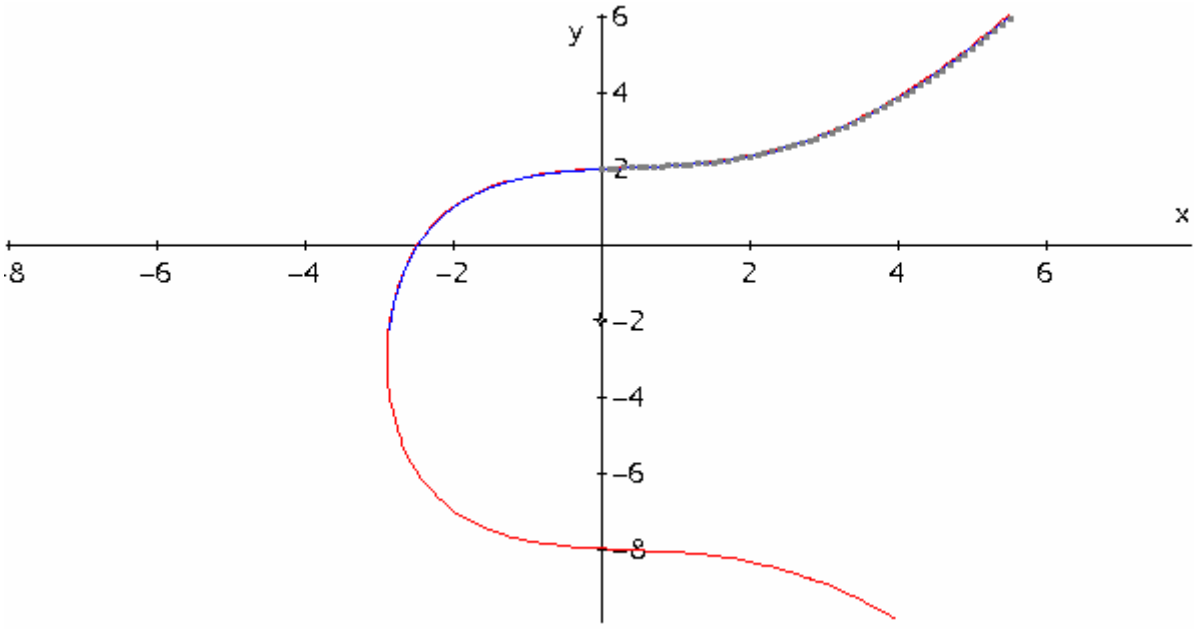


Figure 7

For problem c), let us recall that if $f = f(x, y)$ is a scalar field, then the (implicit) curve defined by $f(x, y) = c$ is called a level curve of f (c being the level). The "old" feature of *DERIVE* that consists to split the screens can be used to see a particular level curve in 2D and in 3D: for the 3D view, simply intersect the surface $z = f(x, y)$ with the plane $z = c$. Here is an example with the function $f(x,y) = x - \frac{x^3}{9} - \frac{y^2}{2}$. Let's plot the level $c = 1$ curve, in 2D and visualize it in 3D (note that one has to make the box turn if axes are to be oriented like they are in 2D):

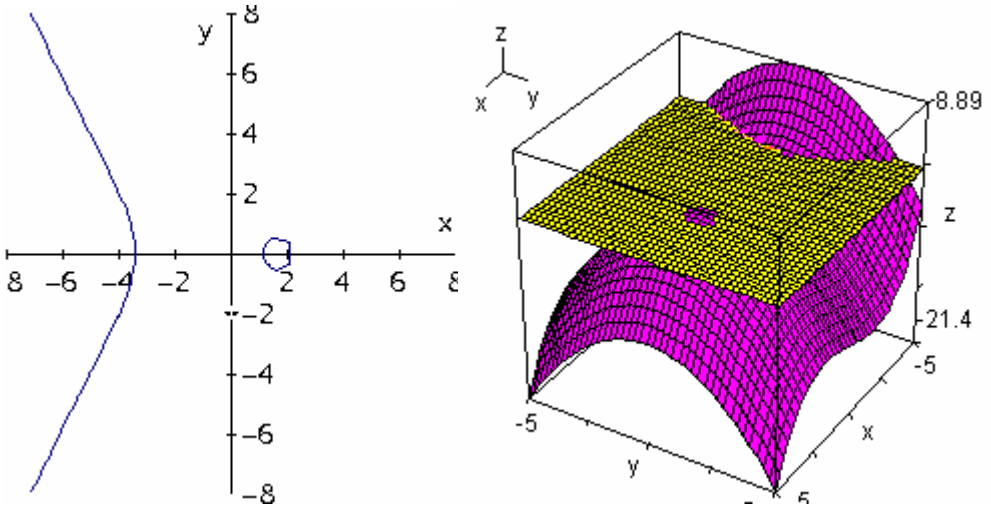


Figure 8

By the way, beginning with *DERIVE* 6, a Gröbner basis function (groebner_basis) started to be used by the author when heavy polynomial systems needed to be solved. That became a nice opportunity to extend the famous row-reduce function used with linear systems.

4. 3D plotting facilities: quite acceptable

Before version 5 of *DERIVE*, 3D plots were very limited. For instance, figure 8 shows, on the right, something that it was impossible to do. Hopefully, David Parker was hired by Texas Instruments to allow *DERIVE* 3D plotting capabilities to be compared with ones of big major CAS. If *DERIVE* 6 does not support implicit 3D plotting, it has parametric 3D plotting ─ one parameter for a space curve, two parameters for a surface ─ that does a good and correct job. Let us concentrate over a single, simple example, an example where *DERIVE* helps the mathematics teacher to concentrate on concepts and not on complicated commands. Suppose you want to generate a torus, taking the circle of radius 1, centered at the point (2, 0, 0) in the plane $y = 0$, and letting it turn around the z-axis. You should get something like this:

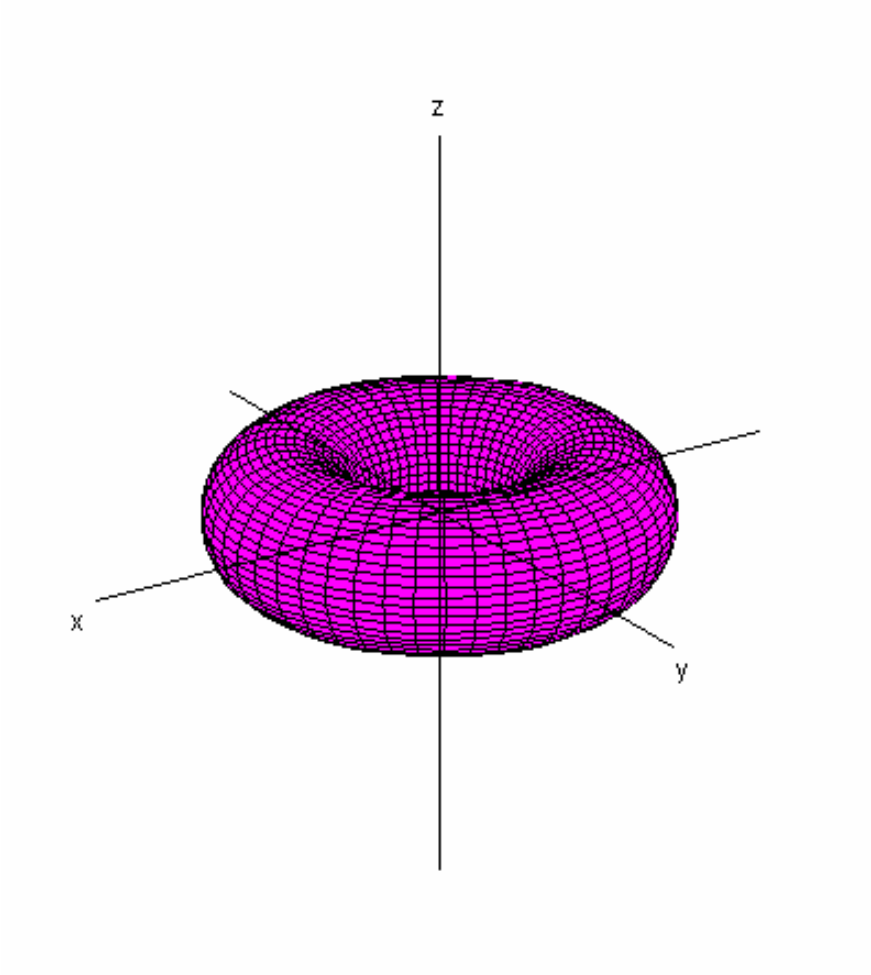
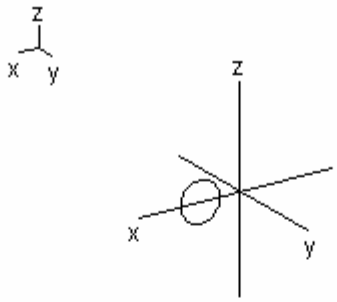


Figure 9

In *DERIVE*, plotting the expression (vector) [2 + cos *t*, 0, sin *t*], where *t* ranges from 0 to 2π, will give the circle ─ of course, one has to find parametric equations for the circle, but students learn how to do this in 2D, so it is quite easy, here! Using the matrix "ROTATE_Z(*s*)" will finish the job. Note that we had to transpose the *matrix* obtained by the *vector* in #1 of figure 10 in order to perform the matrices product in #2 and, then, we had to put the result in vector form in order to get the 3D surface. Before doing this, one can use the slider bar in #5 of figure 10─ replacing *s* by *a* for example ─ and see the circle turning around the *z*-axis: this is something that has to be done "live" with *DERIVE* and if you want to do it, you will see that you don't have to type too many commands. With *DERIVE*, it is easy and fast to get this. Figure 10 concludes this example.

#1: [2 + COS(t), 0, SIN(t)]



#2: ROTATE_Z(s)·[[2 + COS(t), 0, SIN(t)]]'

#3:
$$\begin{bmatrix} \mathrm{COS}(s)\cdot(\mathrm{COS}(t)+2) \\ \mathrm{SIN}(s)\cdot(\mathrm{COS}(t)+2) \\ \mathrm{SIN}(t) \end{bmatrix}$$

#4:
$$\begin{bmatrix} \mathrm{COS}(s)\cdot(\mathrm{COS}(t)+2) \\ \mathrm{SIN}(s)\cdot(\mathrm{COS}(t)+2) \\ \mathrm{SIN}(t) \end{bmatrix}'$$

#5: [[COS(s)·(COS(t) + 2), SIN(s)·(COS(t) + 2), SIN(t)]]

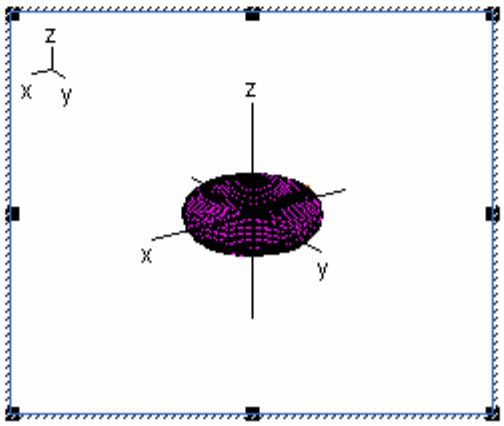


Figure 10

5. Conclusion

This *additional* look at a trusted mathematical assistant ─ limited look where examples concern mainly mathematics at university level ─ is my way to say, again, thank you to Albert Rich and David Stoutemyer, the "fathers" of DERIVE. Also, I want to thank Theresa Shelby, for her nice job regarding the windows versions that appeared after version 3. These persons have created a piece of software that gave me so much enthusiasm during the past 18 years of teaching mathematics to engineering students at ETS, in Montreal, Canada. Also, the TI-92 calculator ─ now Voyage 200 ─ is, as far as we are concerned, a direct result of *DERIVE*'s spirit (I know that Albert Rich won't agree!). And, since 1996, we have started to use both (see [3]): that became more interesting in 1999 when every student at ETS started to buy ─ compulsory purchase ─ a TI-92 Plus (Voyage 200 from 2002) because, as a math teacher, we were able to switch from the calculator to *DERIVE* easily. Also, for many problems, the calculator was sufficient: this was a good reason to continue to think that a laptop computer is not necessary for a student when attending a mathematics course.

DERIVE is no longer on the market. A certain part of its spirit is somewhere in Voyage 200 calculator, whose future is uncertain... Nspire CAS software pretends to become the true successor of *DERIVE*. Again, only time will tell.

6. References

[1] *DERIVE User Manual*, version 3. *A Mathematical Assistant for Your Personal Computer*. Soft Warehouse. Seventh edition, 1994.

[2] Beaudin, Michel. *Teaching Mathematics with CAS to Future Engineers: Some Examples of What We (Absolutely) Need.* Proceedings of TIME-2008 Symposium (Technology and its Integration into Mathematics Education), 22-26 September 2008, Buffelspoort conference centre, South Africa. Josef Böhm, editor, BK Teachware.

[3] Beaudin, Michel. *Using Both: Derive 5 and the TI-92 Plus.* Proceedings of VISIT-ME 2002 symposium, 10-13 July 2002, Vienna, Austria. Josef Böhm, editor, BK Teachware.

[4] Kutzler, Bernhard and Kokol-Voljc, Vlasta. *Introduction to Derive 6*. Texas Instruments, 2003.

Random_distributions.mth: Random samples from distributions with DERIVE

Galán García, José Luis jl_galan@uma.es
Aguilera Venegas, Gabriel gabri@ctima.uma.es
Rodríguez Cielos, Pedro prodriguez@uma.es
Padilla Domínguez, Yolanda ypadilla@ctima.uma.es
Galán García, Mª Ángeles magalan@ctima.uma.es

Department of Applied Mathematic
University of Málaga (Spain)

Abstract

This paper establishes the theoretical aspects which have been considered in order to elaborate the `Random_distiributions` package for DERIVE 6 as well as the description of the different algorithm developed in the package. In section 1 the theory on random number generation is presented (from [Rubinstein, 1981]). After explaining DERIVE's random function (section 1.1) the more efficient algorithms `ran2` and `mzran13` are developed (section 1.2 and 1.3 respectively). Section 2 presents three different general methods for generating continuous distributions together with one for generating discrete distributions. Section 3 is dedicated to describe different algorithms for generating random values from continuous distributions (Uniform, Exponential, Normal, Lognormal, Weibul, Gamma, Beta, Chi-square, Student's t, F, Z, Pareto, Logistic, Cauchy and Irwin-Hall distributions). Section 4 presents different algorithms for generating discrete distributions (Uniform discrete, Bernouille, Rademacher, Binomial, Poisson, Geometric, Negative Binomial and Hypergeometric distributions). In section 5 some algorithms for generating different distributions by approximations are developed. Section 6 is devoted to develop some graphical approaches in order to check graphically how the generated sample fix the distribution. Finally, in sections 7 and 8 some examples and possible extensions to this work are shown.

1 Random Number Generation

The most commonly used methods for generating pseudorandom numbers are *congruential generators.* A congruential method is one that produces a nonrandom sequence of numbers according to some recursive formula based on calculating the residues modulo some integer m of a linear transformation. It is readily seen from this definition that each term of the sequence is available in advance, before the sequence is actually generated. Although these processes are completely deterministic, it can be shown that the numbers generated by the sequence appear to be uniformly distributed and statistically independent.

Congruential methods are based on a fundamental congruence relationship, which may be expressed as:

$$X_{i+1} = a\,X_i + c \pmod{m} \qquad i = 0, 1, 2, \ldots, n \tag{1}$$

where the *multiplier* a, the *increment* c and the *modulus* m are nonnegative integers.

Given an initial starting value X_0 (called the *seed*), (1) yields a congruence relationship (modulo m) for any value i of the sequence $\{X_i\}$. Generators that produce random numbers according to (1) are called *mixed congruential generators.* The random numbers on the unit interval $(0, 1)$ can be obtained by:

$$U_i = \frac{X_i}{m} \tag{2}$$

Clearly, such a sequence will repeat itself in at most m steps, and will therefore be periodic.

As $X_i < m$ for all i, the period of the generator cannot exceed m, that is, the sequence X_i contains at most m different numbers. Because of the deterministic character of the sequence, the entire sequence recurs as soon as any number is repeated. The sequence is said to *get into a loop,* that is, there is a cycle of numbers that is repeated endlessly. Modulus m should be chosen as large as possible and appropriated values of a and c in order to make the *period* p maximum (that is, $p = m$) must be found. When this happens the random number generator has a *full period.* It can be shown that the generator defined in (1) has a full period m, if and only if:

1. c is *relative prime* to m, that is, c and m have no common divisor.
2. $a \equiv 1 \pmod{g}$ for every prime factor g of m.
3. $a \equiv 1 \pmod{4}$ if m is a multiple of 4.

Since most computers utilize either a binary or a decimal digit system, the best selection for m is $m = 2^\beta$ or $m = 10^\beta$, respectively where β is the word–length of the particular binary or decimal computer.

For binary computers, in order to develop a full period generator when $m = 2^\beta$, the parameter c must be odd and $a = 4k + 1$ for some $k \in \mathbb{N}$.

The second widely used generator is the *multiplicative generator:*

$$X_{i+1} = a\, X_i \pmod{m} \qquad\qquad i = 0, 1, 2, \ldots, n \tag{3}$$

which is a particular case of the mixed generator (1) with $c = 0$.

Another common type of generator in which X_{i+1} depends on more than one of the preceding values[1]. For example:

$$\begin{aligned} X_{i+1} &= a_1\, X_{i-j_1} + a_2\, X_{i-j_2} + \cdots + a_k\, X_{i-j_k} + c \pmod{m} \qquad \text{or} \\ X_{i+1} &= a\, X_{i-j_1} \cdot X_{i-j_2} \cdots X_{i-j_k} + c \pmod{m} \end{aligned}$$

Nowadays "the best" generators use combinations of the generators described along this section in order to increase the randomness and the period of the generated sequences.

[1]These generators are often called *Fibonacci* generators because one example is given by the Fibonacci serie:

$$X_{i+1} = X_i + X_{i-1} \pmod{m}$$

1.1 DERIVE's random function

DERIVE's random function uses a mixed generator given by:

$$X_{i+1} = 2,654,435,721\, X_i + 1 \pmod{2^{32}}$$

which satisfies the conditions to be a full period generator, that is, the period of DERIVE's random function is $2^{32} = 4,294,967,296$.

DERIVE's random function `RANDOM(n)` can be used with any $n \in \mathbb{Z}$ with the following meanings:

- If $n > 1$, `RANDOM(n)` simplifies to a random integer in the interval $[0, n)$.
- `RANDOM(1)` simplifies to a random number in the interval $[0, 1)$.
- If $n < 0$, `RANDOM(n)` simplifies to $-n$ and initializes the random number state variable to $-n$.
- `RANDOM(0)` simplifies to the time in centiseconds since the current calendar year began and initializes the random number state variable to that time.

Although this is a "good" generator, the following two subsections describe two different algorithms, implemented in the package `Random_distributions`, which periods are quite much longer and also improve the randomness.

1.2 ran2 algorithm

The `ran2` algorithm was proposed by L'Ecuyer and is described in [Press and Teukolsky, 1992] and [Press et al., 1999].

This algorithm merges the following two multiplicative generators:

$$\begin{aligned} X_{i+1} &= 40014\, X_i \pmod{2^{31} - 85} \\ Y_{i+1} &= 40692\, Y_i \pmod{2^{31} - 249} \end{aligned}$$

This algorithm has been used for a long time as one of the best generators and its period is about $2.3 \cdot 10^{18} = 2,300,000,000,000,000,000$ which is more than $535,510,480$ times longer than DERIVE's random generator period.

The implementation on DERIVE has been carried out "translating" the "C" code developed in [Press et al., 1999] and it uses the following two functions:

- `ran2(n)` which is the main algorithm. This function returns a vector of size n of random numbers in the interval $[0, 1)$.
- `ran2_initialize()` This auxiliar function is used to set the variables and constants needed for the algorithm.

1.3 mzran13 algorithm

The `mzran13` algorithm was proposed by G. Marsaglia and A. Zaman as an alternative to `ran2`. This algorithm is described in [Marsaglia and Zaman, 1994].

This algorithm merges the two generators: a mixed one with a Fibonacci's like one.

$$\begin{aligned} X_{i+1} &= 69069\, X_i + 1,013,904,243 \pmod{2^{32}} \\ Y_{i+1} &= Y_{i-1} - Y_{i-2} - \text{"c"} \pmod{2^{32} - 18} \end{aligned}$$

where the second one is a subtract-with-borrow generator (because of the term "c").

This algorithm has been found to be at least as good as `ran2` but simpler, much faster and with periods "millions and millions" of times longer. Specifically, its period is over

$$2^{94} = 19,807,040,628,566,084,398,385,987,584$$

that is, 8.611.756.795 times longer than `ran2`'s period and $4,611,686,018,427,387,904$ times longer than DERIVE's period.

The implementation on DERIVE has been carried out "translating" the "C" code developed in [Marsaglia and Zaman, 1994] and it uses the following function:

- `mzran13(n)`. This function returns a vector of size n of random numbers in the interval $[0,1)$. Previously, when the package `Random_distribution` is loaded, the needed constants and variables are initialized.

This algorithm is the base for all the random distribution generations developed in this package. This, in the following, when we say that a value u is generated from $\mathcal{U}(0,1)$, this is done using `mzran13` algorithm.

2 Different methods for random variate generation

This section presents some general methods for generating random variables from different continuous and discrete distributions. In the following subsections three general methods for continuous distributions and one for discrete distributions are described.

2.1 Inverse transform method

Let $\mathcal{X}$ be a random variable with cumulative probability distribution function $F_{\mathcal{X}}(x)$. Since $F_{\mathcal{X}}(x)$ is a nondecreasing function, the inverse function $F_{\mathcal{X}}^{-1}(y)$ may be defined for any value of y between 0 and 1 as: $F_{\mathcal{X}}^{-1}(y)$ is the smallest x satisfying $F_{\mathcal{X}}(x) \geq y$, that is,

$$F_{\mathcal{X}}^{-1}(y) = \inf\{F_{\mathcal{X}}(x) \geq y\}, \qquad 0 \leq y \leq 1$$

If $\mathcal{U}$ is uniformly distributed over the interval $(0,1)$, then $\mathcal{X} = F_{\mathcal{X}}^{-1}(\mathcal{U})$. So, to get a value x of the random variable $\mathcal{X}$, a value u from a random uniform variable $\mathcal{U}(0,1)$ can be obtained and compute $x = F_{\mathcal{X}}^{-1}(y)$. Thus, the general algorithm for the inverse transform method is:

1. Generate a value u from $\mathcal{U}(0,1)$.
2. Obtain $x = F_{\mathcal{X}}^{-1}(u)$ as the random number from the variable $\mathcal{X}$.

The only condition needed for this method is that $F_{\mathcal{X}}^{-1}$ exists in an analytical form.

The following DERIVE's program has been developed in the package `Random_distributions` to obtain a formula to generate $\mathcal{X}$ using the inverse transform method:

```
Inverse_transform_method(f, ini := 0, u) :=
Prog
(
  u :∈ Real [0,1],
  Solve(u = INT(f, x, ini, x), x, Real)
)
```

2.2 Composition method

This method is employed by Butler and consists of expressing the probability density function $f_{\mathcal{X}}(x)$ of the distribution to be simulated as a probability mixture of properly selected density functions.

Let $g(x|y)$ be a family of one-parameter density functions, where y is the parameter identifying a unique $g(x)$. If a value of y is drawn from a continuous cumulative function $F_{\mathcal{Y}}(y)$ and then if $\mathcal{X}$ is sampled from the $g(x)$ for that chosen y, the density function for $\mathcal{X}$ will be

$$f_{\mathcal{X}}(x) = \int g(x|y)\, \mathrm{d}F_{\mathcal{Y}}(y)$$

If y is an integer parameter, then

$$f_{\mathcal{X}}(x) = \sum_i P_i\, g(x|y=i)$$

where

$$\sum_i P_i = 1 \qquad ; \qquad P_i > 0 \qquad ; \qquad P_i = P\left[\mathcal{Y} = i\right] \qquad i = 1, 2, \ldots$$

This method may be applied for generating complex distributions from simpler distributions that are themselves easily generated by the inverse transform method or by the acceptance-rejection method described below.

2.3 Acceptance–rejection method

This method is due to von Neumann and consists on sampling a random variate from an appropriate distribution and subjecting it to a test to determine whether or not it will be acceptable for use.

To carry out this method, the probability density function $f_{\mathcal{X}}(x)$ of the variable $\mathcal{X}$ must be expressed as:

$$f_{\mathcal{X}}(x) = C \cdot h(x) \cdot g(x)$$

where $C \geq 1$, $h(x)$ is also a probability density function, and $0 < g(x) \leq 1$. After generating two random values u and y from $\mathcal{U}(0,1)$ and $h(y)$, respectively, the test to see wether or not the inequality $u \leq g(y)$ holds must be done, and:

1. If the inequality holds, then accept y as a variate generated from $f_{\mathcal{X}}(x)$.
2. If the inequality is violated, reject the pair u , y and try again.

So, the general algorithm for the acceptance–rejection method is:

1. Generate a value u from $\mathcal{U}(0,1)$
2. Generate y from the probability density function $h(y)$.
3. If $u \leq g(y)$, return y as the variate generated from $f_{\mathcal{X}}(x)$
4. Go to step 1.

2.4 Inverse transform method for discrete distributions

The inverse transform method is the easier method to use not only for continuous distributions but also for discrete distribution.

Let $\mathcal{X}$ be a random discrete variate which finite or infinite possible values are

$$x_1, x_2, \ldots, x_i, \ldots.$$

Let $F_{\mathcal{X}}(x)$ be its probability mass function given by

$$F(x_i) = P[\mathcal{X} = x_i] = p_i \quad i = 1, 2, \ldots$$

The inverse transform method can be described as follow:

1. Generate u from $\mathcal{U}(0, 1)$.
2. $i := 1$.
3. $p := p_1$.
4. If $u \leq p$ deliver x_i as the generated value.
5. $i := i + 1$.
6. $p := p + p_i$
7. Go to step 4.

On the other hand, the values x_i can be assumed to be all integers and $x_{i+1} = x_i + 1$, because if this is not the case, the correspondence $\phi(x_i) = i$ can be established and consider a new random discrete variate $\mathcal{Y}$ which values are $1, 2, \ldots$ and $F_{\mathcal{Y}}(i) = P[\mathcal{Y} = i] = P[\mathcal{X} = x_i] = p_i$, which is equivalent to $\mathcal{X}$ and verify the above condition.

Let $\mathcal{X} = \{\texttt{ini}, \texttt{ini} + 1, \texttt{ini} + 2, \ldots\}$ for some $\texttt{ini} \in \mathbb{Z}$ with probability mass function $F_{\mathcal{X}}(x) := P[\mathcal{X} = x] = p_x \quad ; \quad x = \texttt{ini}, \texttt{ini} + 1, \texttt{ini} + 2, \ldots$

The following DERIVE's program has been developed in the package `Random_distributions` to generate an element of $\mathcal{X}$ using the inverse transform method:

```
random_discrete_aux(F, ini := 0, aleat, i_, p) :=
Prog
(
  i_ := ini,
  p := SUBST(F, x, i_),
  Loop
  (
    If aleat ≥ p,
      RETURN i_),
    i_ := i_ + 1,
    p := p + SUBST(F, x, i_)
  )
)
```

while the following DERIVE's program generate a sample of size n of $\mathcal{X}$.

```
random_discrete(n := 1, F, ini := 0, vecaleat) :=
Prog
(
  vecaleat := random_uniform(n),
  VECTOR(random_discrete_aux(F, ini, vecaleat sub j), j, n)
)
```

3 Continuous distributions random generation

This section describes generating procedures for different continuous distributions.

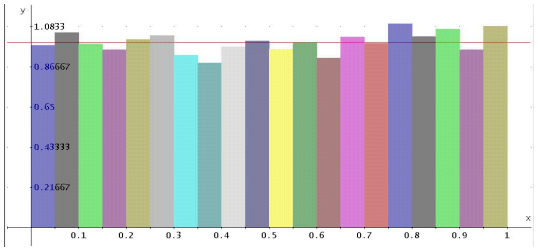
All these continuous distributions are presented with their probability density functions and with a drawing of a bars diagram, obtained by the DERIVE's algorithm developed in the package `Random_distributions`, together with the plot of the corresponding probability density function in order to see graphically if it is a "good" sample of generated values (see section 6.

See [Rubinstein, 1981], [Galán, 1991] and [Wikipedia, 2009] for further information on the algorithm described below.

3.1 Uniform distribution

A random variable $\mathcal{X}$ has an uniform distribution in the interval (a, b) ($\mathcal{X} \rightsquigarrow \mathcal{U}(a, b)$) if its probability density function is:

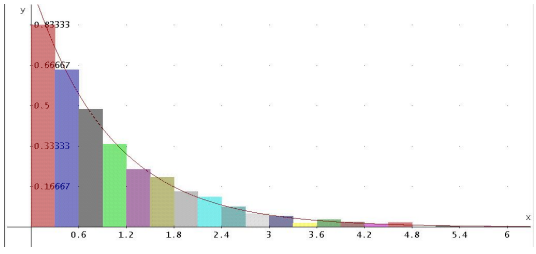
$$f_{\mathcal{X}}(x) = \begin{cases} \dfrac{1}{b-a} & x \in (a, b) \\ 0 & \text{otherwise} \end{cases}$$



$\mathcal{U}(0, 1)$

To generate $\mathcal{X}$, first u must be generated from $\mathcal{U}(0, 1)$ and then return $\boxed{a + (b - a)\, u}$ as the generated value.

3.2 Exponential distribution



$\mathcal{E}(1)$

A random variable $\mathcal{X}$ has an exponential distribution with parameter $\lambda > 0$ ($\mathcal{X} \rightsquigarrow \mathcal{E}(\lambda)$) if its probability density function is:

$$f_{\mathcal{X}}(x) = \begin{cases} \dfrac{1}{\lambda}\, \mathrm{e}^{-x/\lambda} & x \in [0, \infty) \\ 0 & \text{otherwise} \end{cases}$$

To generate $\mathcal{X}$ the inverse transform method is used:

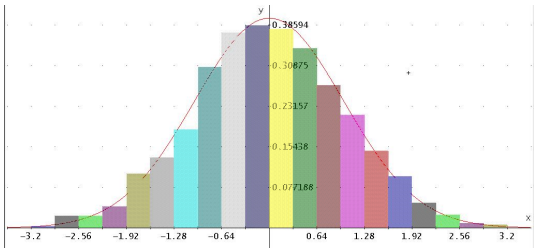
$$\mathcal{U} = F_{\mathcal{X}}(x) = \int_0^x \frac{1}{\lambda}\, \mathrm{e}^{-t/\lambda}\, \mathrm{d}t = 1 - \mathrm{e}^{-x/\lambda} \quad \Longrightarrow \quad \mathcal{X} = -\lambda\, \ln(1 - \mathcal{U}) \equiv -\lambda\, \ln(\mathcal{U})$$

Thus, to generate $\mathcal{X}$, u must be generated from $\mathcal{U}(0, 1)$ and then return $\boxed{x = -\lambda \ln u}$ as the generated value.

3.3 Normal distribution

A random variable $\mathcal{X}$ has a normal distribution with parameters μ and σ ($\mathcal{X} \rightsquigarrow \mathcal{N}(\mu, \sigma)$) if its probability density function is:

$$f_{\mathcal{X}}(x) = \frac{1}{\sigma\sqrt{2\pi}}\, \mathrm{e}^{-\frac{(x-\mu)^2}{2\sigma^2}} \qquad x \in \mathbb{R}$$



$\mathcal{N}(0,1)$

To generate $\mathcal{X}$, Box and Muller theorem establishes that:

If $\mathcal{U}_1 \rightsquigarrow \mathcal{U}(0,1)$ and $\mathcal{U}_2 \rightsquigarrow \mathcal{U}(0,1)$ then

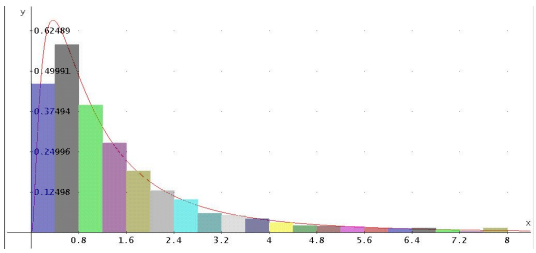
$$\mathcal{Z}_1 = \sqrt{-2\ln(\mathcal{U}_1)}\,\cos(2\pi\mathcal{U}_2) \qquad \text{and} \qquad \mathcal{Z}_2 = \sqrt{-2\ln(\mathcal{U}_1)}\,\sin(2\pi\mathcal{U}_2)$$

are independent standard normal deviates $\mathcal{N}(0,1)$

On the other hand, if $\mathcal{Z} \rightsquigarrow \mathcal{N}(0,1)$ then $(\mu + \sigma\,\mathcal{Z}) \rightsquigarrow \mathcal{N}(\mu, \sigma)$

Thus, to generate $\mathcal{X}$, u_1 and u_2 must be generated from $\mathcal{U}(0,1)$ and then return any of the values $\boxed{\mu + \sigma\sqrt{-2\ln(u_1)}\,\cos(2\pi\, u_2)}$ or $\boxed{\mu + \sigma\sqrt{-2\ln(u_1)}\,\sin(2\pi\, u_2)}$ as the generated value.

3.4 Lognormal distribution



$\mathcal{LN}(0,1)$

If the random variable $\mathcal{Z} \rightsquigarrow \mathcal{N}(\mu, \sigma)$ then $\mathcal{X} = \mathrm{e}^{\mathcal{Z}}$ has the lognormal distribution with parameters μ and σ ($\mathcal{X} \rightsquigarrow \mathcal{LN}(\mu, \sigma)$). Its probability density function is:

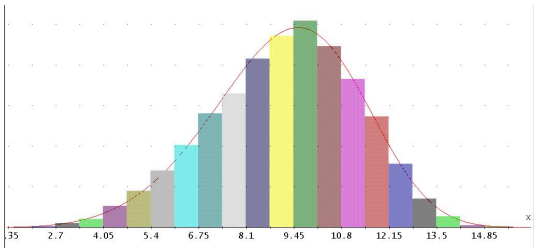
$$f_{\mathcal{X}}(x) = \begin{cases} \dfrac{1}{x\sigma\sqrt{2\pi}}\, \mathrm{e}^{-\frac{(\ln x-\mu)^2}{2\sigma^2}} & x \in [0, \infty) \\ 0 & \text{Otherwise} \end{cases}$$

To generate $\mathcal{X}$, z must be generated from $\mathcal{N}(\mu, \sigma)$ and then return $\boxed{x = \mathrm{e}^z}$ as the generated value.

3.5 Weibul distribution

A random variable $\mathcal{X}$ has a Weibul distribution with parameters $\alpha > 0$ and $\beta > 0$ ($\mathcal{X} \rightsquigarrow \mathcal{W}(\alpha, \beta)$) if its probability density function is:

$$f_{\mathcal{X}}(x) = \begin{cases} \dfrac{\alpha}{\beta^{\alpha}}\, x^{\alpha-1}\, \mathrm{e}^{-(\frac{x}{\beta})^{\alpha}} & x \in [0, \infty) \\ \\ 0 & \text{Otherwise} \end{cases}$$



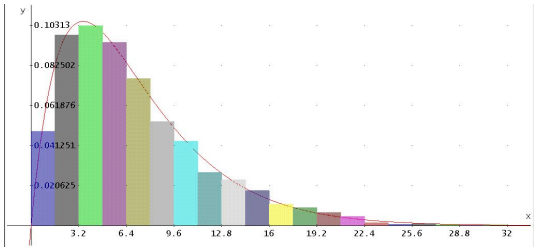
$\mathcal{W}(5, 10)$

To generate $\mathcal{X}$ the inverse transform method can be used:

$$\begin{aligned} \mathcal{U} &= F_{\mathcal{X}}(x) = \int_0^x \frac{\alpha}{\beta^{\alpha}}\, t^{\alpha-1}\, \mathrm{e}^{-(\frac{t}{\beta})^{\alpha}}\, \mathrm{d}t = 1 - \mathrm{e}^{-(\frac{x}{\beta})^{\alpha}} \qquad \Longrightarrow \\ \left(\frac{\mathcal{X}}{\beta}\right)^{\alpha} &= -\ln(1-\mathcal{U}) \equiv -\ln \mathcal{U} \rightsquigarrow \mathcal{E}(1) \qquad \Longrightarrow \\ \mathcal{X} &\equiv \beta \left(\mathcal{E}(1)\right)^{1/\alpha} \end{aligned}$$

To generate $\mathcal{X}$, v must be generated from $\mathcal{E}(1)$ and then return $\boxed{x = \beta v^{1/\alpha}}$ as the generated value.

3.6 Gamma distribution



$\mathcal{G}(2, 3.5)$

A random variable $\mathcal{X}$ has a Gamma distribution with parameters $\alpha > 0$ and $\beta > 0$ ($\mathcal{X} \rightsquigarrow \mathcal{G}(\alpha, \beta)$) if its probability density function is:

$$f_{\mathcal{X}}(x) = \begin{cases} \dfrac{x^{\alpha-1}\, \mathrm{e}^{-x/\beta}}{\beta^{\alpha}\, \Gamma(\alpha)} & x \in [0, \infty) \\ \\ 0 & \text{Otherwise} \end{cases}$$

The inverse transform method cannot be applied since $F_{\mathcal{X}}^{-1}(x)$ does not exist in an explicit form.

In this case, different algorithms have been developed in order to generate samples of the Gamma distribution. Finally, the main algorithm chooses which is the appropriated one depending on the values of parameters α and β.

3.6.1 random_gamma1

This algorithm is valid for values of $\alpha > 1$. The following two properties of gamma distribution are the base to develop this algorithm:

1. $\mathcal{G}(1, \beta) = \mathcal{E}(\beta)$.
2. If $\mathcal{X}_1 \rightsquigarrow \mathcal{G}(\alpha_1, \beta)$ and $\mathcal{X}_2 \rightsquigarrow \mathcal{G}(\alpha_2, \beta)$ then $\mathcal{X} = \mathcal{X}_1 + \mathcal{X}_2 \rightsquigarrow \mathcal{G}(\alpha_1 + \alpha_2, \beta)$, that is, gamma distribution is reproductive with respect its first parameter.

Let $\alpha > 1$, $m = \text{floor}(\alpha)$ and $\delta = \alpha - m$ where $\text{floor}(\alpha)$ is the integer part of α. In order to generate $\mathcal{X} \rightsquigarrow \mathcal{G}(\alpha, \beta)$, a mixture of $\mathcal{G}(m, \beta)$ and $\mathcal{G}(m+1, \beta)$ with probabilities $1-\delta$ and δ respectively can be used. On the other hand, in order to generate them, m or $m+1$ variables from $\mathcal{G}(1, \beta) = \mathcal{E}(\beta)$ must be generated as shown in section 3.2. Thus, given $\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_{m+1} \rightsquigarrow \mathcal{U}(0,1)$:

$$\begin{aligned} \mathcal{X} &= -\beta \ln(\mathcal{U}_1) - \beta \ln(\mathcal{U}_2) - \cdots - \beta \ln(\mathcal{U}_m) = -\beta \ln\left(\prod_{i=1}^{m} \mathcal{U}_i\right) \rightsquigarrow \mathcal{G}(m, \beta) \qquad \text{and} \\ \mathcal{Y} &= -\beta \ln(\mathcal{U}_1) - \beta \ln(\mathcal{U}_2) - \cdots - \beta \ln(\mathcal{U}_{m+1}) = -\beta \ln\left(\prod_{i=1}^{m+1} \mathcal{U}_i\right) \rightsquigarrow \mathcal{G}(m+1, \beta) \end{aligned}$$

The algorithm is:

1. Get $u, u_1, u_2, \ldots, u_m, u_{m+1}$ from $\mathcal{U}(0,1)$.
2. Let $x = \prod_{i=1}^{m} u_i$.
3. If $\delta \leq u$ then let $x = x \cdot u_{m+1}$.
4. Return $-\beta \ln(x)$ as the generate value.

3.6.2 random_gamma2

If $0 < \alpha < 1$ then $\mathcal{X} = \mathcal{Y} \cdot \mathcal{V}$ where $\mathcal{X} \rightsquigarrow \mathcal{G}(\alpha, \beta)$; $\mathcal{Y} \rightsquigarrow \mathcal{B}\text{e}(\alpha, 1-\alpha)$ and $\mathcal{V} \rightsquigarrow \mathcal{E}(\beta)$. (Beta distribution $\mathcal{B}\text{e}$ is described in section 3.7). Thus, `random_gamma2` algorithm to generate $\mathcal{X} \rightsquigarrow \mathcal{G}(\alpha, \beta)$ $(0 < \alpha < 1)$ can be described by:

1. Generate y from $\mathcal{B}\text{e}(\alpha, 1-\alpha)$ (using algorithm `random_beta4` described in section 3.7.2).
2. Generate v from $\mathcal{E}(\beta)$.
3. Return $y \cdot v$ as generated value.

3.6.3 random_gamma5

This is an acceptance–rejection method due to Cheng and describes gamma generation $\mathcal{G}(\alpha, 1)$ for $\alpha > 1$. Let us remember that the acceptance–rejection method is based in the following decomposition:

$$f_{\mathcal{X}}(x) = C \cdot h(x) \cdot g(x)$$

Cheng's procedure uses:

$$\begin{aligned} h(x) &= \begin{cases} \dfrac{\lambda\ \mu\ x^{\lambda-1}}{(\mu + x^{\lambda})^2} & x \geq 0 \\ 0 & \text{Otherwise} \end{cases} \\ C &= \frac{4\alpha^{\alpha}}{\Gamma(\alpha)\ \mathrm{e}^{\alpha}\ \lambda} \\ g(x) &= x^{\alpha-\lambda}\ \left(\mu + x^{\lambda}\right)^2 \frac{\mathrm{e}^{\alpha-x}}{4\ \alpha^{\alpha+\lambda}} \qquad \text{where} \\ \lambda &= \sqrt{2\alpha - 1} \qquad ; \qquad \mu = \alpha^{\lambda} \end{aligned}$$

Setting $a = \dfrac{1}{\lambda}$, $b = \alpha - \ln 4$ and $c = \alpha + a$ Cheng's algorithm can be written as:

1. Get u_1 and u_2 from $\mathcal{U}(0,1)$.
2. Let $v = a \ln\left(\frac{u_1}{1-u_1}\right)$.
3. Let $x = \alpha\, e^v$.
4. If $b + c\, v - x \geq \ln(u_1^2 u_2)$ return x as the generated value for $\mathcal{G}(\alpha,1)$.
5. Go to step 1.

3.6.4 random_gamma9

This algorithm uses the approximation to Gamma distribution by Normal distribution when α and β are not "small".

Specifically, if $\mathcal{Z} \rightsquigarrow \mathcal{N}\left(\ln\left(\frac{\alpha}{\beta}\right) - \frac{1}{2\alpha}, \sqrt{\frac{1}{\alpha}}\right)$ then $\mathcal{G}(\alpha,\beta) \approx e^{\mathcal{Z}}$. Thus, the algorithm is:

1. Generate z from $\mathcal{N}\left(\ln\left(\frac{\alpha}{\beta}\right) - \frac{1}{2\alpha}, \sqrt{\frac{1}{\alpha}}\right)$.
2. Deliver $x = e^z$ as the generated value for $\mathcal{G}(\alpha,\beta)$.

3.6.5 random_gamma

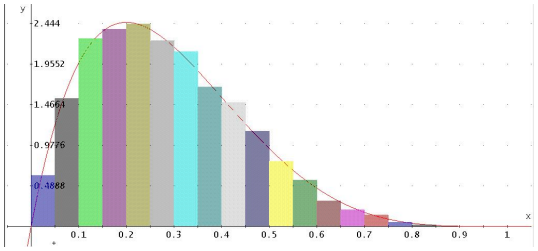
Finally, the following algorithm runs the above algorithms depending on the parameters α and β in order to generate a sample of size n from $\mathcal{G}(\alpha,\beta)$:

```
random_gamma(n := 1, α := 1, β := 1) :=
Prog
(
  If (α = 1, return random_exponential(n, β)),
  If (α < 1, return random_gamma2(n, α, β)),
  If (β = 1, return random_gamma5(n, α, β)),
  If (α > 20 and β > 20, return random_gamma9(n, α, β)),
  return random_gamma1(n, α, β)
)
```

3.7 Beta distribution

$\mathcal{X}$ has a Beta distribution with parameters $\alpha > 0$ and $\beta > 0$ ($\mathcal{X} \rightsquigarrow \mathcal{B}e(\alpha,\beta)$) if its probability density function is:

$$f_{\mathcal{X}}(x) = \begin{cases} \dfrac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\cdot\Gamma(\beta)}\, x^{\alpha-1}(1-x)^{\beta-1} & x \in [0,1] \\ 0 & \text{Otherwise} \end{cases}$$



$\mathcal{B}e(2,5)$

The inverse transform method cannot be applied since $F_{\mathcal{X}}^{-1}(x)$ does not exist in an explicit form.

In this case, as in the case of Gamma distribution, different algorithms have been developed in order to generate samples of the Beta distribution. Finally, the main algorithm chooses which is the appropriated one depending on the values of parameters α and β.

3.7.1 random_beta1

This algorithm is based in the following result:

> If $\mathcal{Y}_1 \rightsquigarrow \mathcal{G}(\alpha, 1)$ and $\mathcal{Y}_2 \rightsquigarrow \mathcal{G}(\beta, 1)$ then $\mathcal{X} = \dfrac{\mathcal{Y}_1}{\mathcal{Y}_1 + \mathcal{Y}_2} \rightsquigarrow \mathcal{B}\mathrm{e}(\alpha, \beta)$.

Its implementation is therefore trivial:

1. Generate y_1 and y_2 from $\mathcal{G}(\alpha, 1)$ and $\mathcal{G}(\beta, 1)$ respectively.
2. Deliver $x = \dfrac{y_1}{y_1 + y_2}$ as the generated value for $\mathcal{B}\mathrm{e}(\alpha, \beta)$.

3.7.2 random_beta4

This algorithm has been developed for using in algorithm `random_gamma2`. It is due to Jöhnk and is based on the following result:

Let $\mathcal{U}_1 \rightsquigarrow \mathcal{U}(0, 1)$ and $\mathcal{U}_2 \rightsquigarrow \mathcal{U}(0, 1)$ and let $\mathcal{Y}_1 = \mathcal{U}_1^{1/\alpha}$ and $\mathcal{Y}_2 = \mathcal{U}_2^{1/\beta}$. If $\mathcal{Y}_1 + \mathcal{Y}_2 < 1$ then $\mathcal{X} = \dfrac{\mathcal{Y}_1}{\mathcal{Y}_1 + \mathcal{Y}_2} \rightsquigarrow \mathcal{B}\mathrm{e}(\alpha, \beta)$.

The algorithm is:

1. Generate u_1 and u_2 from $\mathcal{U}(0, 1)$.
2. Set $y_1 = u_1^{1/\alpha}$ and $y_2 = u_2^{1/\beta}$.
3. If $y_1 + y_2 < 1$ deliver $x = \dfrac{y_1}{y_1 + y_2}$ as the generated value for $\mathcal{B}\mathrm{e}(\alpha, \beta)$.
4. Go to step 1.

3.7.3 random_beta7

This algorithm uses the approximation to Beta distribution by Normal distribution when α and β are not "large enough".

Specifically, if $\mathcal{X} \rightsquigarrow \mathcal{B}\mathrm{e}(\alpha, \beta)$ then $\ln\left(\dfrac{\mathcal{X}}{1 - \mathcal{X}}\right) \approx \mathcal{N}(\mu, \sigma)$ where $\mu = \ln\left(\dfrac{\alpha}{\beta}\right) + \dfrac{\alpha - \beta}{2\alpha\beta}$ and $\sigma = \sqrt{\dfrac{\alpha + \beta}{\alpha\beta}}$.

Thus,

$$\ln\left(\frac{x}{1 - x}\right) = z \qquad \Longrightarrow \qquad x = \frac{\mathrm{e}^z}{(1 + \mathrm{e}^z)}$$

and hence, the algorithm is:

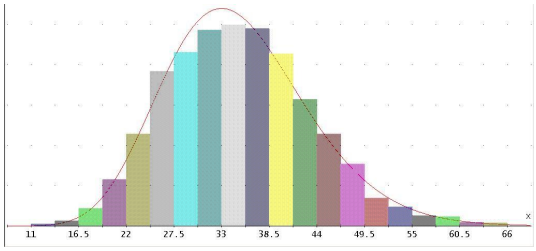
1. Generate z from $\mathcal{N}\left(\ln\left(\dfrac{\alpha}{\beta}\right) + \dfrac{\alpha - \beta}{2\alpha\beta}, \sqrt{\dfrac{\alpha + \beta}{\alpha\beta}}\right)$.
2. Deliver $x = \dfrac{\mathrm{e}^z}{(1 + \mathrm{e}^z)}$ as the generated value for $\mathcal{B}\mathrm{e}(\alpha, \beta)$.

3.7.4 random_beta

Finally, the following algorithm runs the above algorithms depending on the parameters α and β in order to generate $\mathcal{B}e(\alpha, \beta)$:

```
random_beta(n := 1, α := 1, β := 1) :=
If ( α < 4 or β < 4,
    random_beta1(n, α, β),
    random_beta7(n, α, β)
    )
```

3.8 Chi–Square distribution



$\chi^2(35)$

Let $\mathcal{Z}_i \rightsquigarrow \mathcal{N}(0,1) \quad i = 1, \ldots, k \quad k$ standard normal independent distributions. In this case, $\mathcal{X} = \sum_{i=1}^{k} \mathcal{Z}_i^2$ has the chi–square distribution with k degrees of freedom ($\mathcal{X} \rightsquigarrow \chi^2(k)$).

Its probability density function is given by:

$$f_{\mathcal{X}}(x) = \begin{cases} \dfrac{x^{k/2-1}\ \mathrm{e}^{-x/2}}{2^{k/2}\ \Gamma\left(\frac{k}{2}\right)} & x \in [0, \infty) \\ \\ 0 & \text{Otherwise} \end{cases}$$

Although the algorithm to generate $\mathcal{X} \rightsquigarrow \chi^2(k)$ would be trivial by definition, it would need k values from $\mathcal{N}(0,1)$ which require many operations if k is "large". Thus, in the next two sections, two different algorithms which improve (in number of operations) the "definition algorithm" are described.

3.8.1 random_chi_square2

This algorithm is valid for "large" values of k (say $k > 30$) and it uses the following approximation from the standard normal distribution:

If $\mathcal{X} \rightsquigarrow \chi^2(k)$ then $\mathcal{Z} = \sqrt{2\mathcal{X}} - \sqrt{2k-1}$ is such that $\mathcal{Z} \rightsquigarrow \mathcal{N}(0,1)$

Solving for $\mathcal{X}$, $\mathcal{X} = \dfrac{\left(\mathcal{Z} + \sqrt{2k-1}\right)^2}{2}$. Thus, to generate $\mathcal{X} \rightsquigarrow \chi^2(k) \quad z$ must be generated from $\mathcal{N}(0,1)$ and then return $\boxed{x = \dfrac{\left(z + \sqrt{2k-1}\right)^2}{2}}$ as the generated value.

3.8.2 random_chi_square3

This algorithm is based in the fact that $\chi^2(k)$ is a particular case of a gamma density. Specifically, $\chi^2(k) \equiv \mathcal{G}\left(\dfrac{k}{2}, 2\right)$. Thus, to generate $\mathcal{X} \rightsquigarrow \chi^2(k), \quad g$ must be generated from $\mathcal{G}\left(\dfrac{k}{2}, 2\right)$ and then return $\boxed{x = g}$ as the generated value.

3.8.3 random_chi_square

Finally, the following algorithm runs the above algorithms depending on the parameter k in order to generate $\chi^2(k)$:

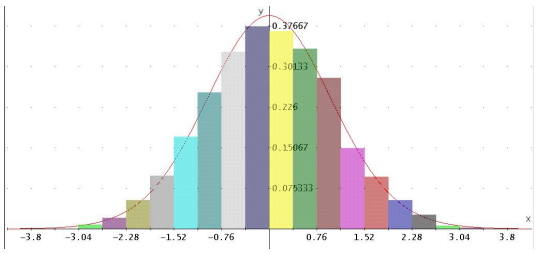
```
random_chi_square(n := 1, k := 1) :=
If ( k > 30,
    random_chi_square2(n, k),
    random_chi_square3(n, k)
    )
```

3.9 Student's t distribution

Let $\mathcal{Z} \rightsquigarrow \mathcal{N}(0,1)$ and $\mathcal{Y} \rightsquigarrow \chi^2(k)$ independents. Then $\mathcal{X} = \dfrac{\mathcal{Z}}{\sqrt{\mathcal{Y}/k}}$ has a Student's t distribution with k degrees of freedom ($\mathcal{X} \rightsquigarrow t(k)$).

Its probability density function is:

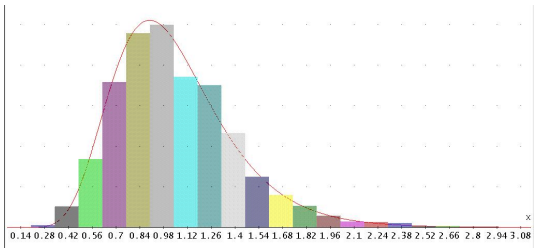
$$f_{\mathcal{X}}(x) = \frac{\Gamma\left(\frac{k+1}{2}\right)}{\sqrt{k\pi}\,\Gamma\left(\frac{k}{2}\right)} \left(1 + \frac{x^2}{k}\right)^{-(k+1)/2} \qquad x \in \mathbb{R}$$



$t(35)$

To generate $\mathcal{X} \rightsquigarrow t(k)$, z must be generated from $\mathcal{N}(0,1)$ and y from $\chi^2(k)$ and then return $\boxed{x = \dfrac{z}{\sqrt{y/k}}}$ as the generated value.

3.10 F distribution



$\mathcal{F}(32, 45)$

Let $\mathcal{Y}_1 \rightsquigarrow \chi^2(k_1)$ and $\mathcal{Y}_2 \rightsquigarrow \chi^2(k_2)$ independents. Then $\mathcal{X} = \dfrac{\mathcal{Y}_1/k_1}{\mathcal{Y}_2/k_2}$ has a $\mathcal{F}$ distribution with k_1 and k_2 degrees of freedom ($\mathcal{X} \rightsquigarrow \mathcal{F}(k_1, k_2)$).

Its probability density function is given by:

$$f_{\mathcal{X}}(x) = \begin{cases} \dfrac{\Gamma\left(\frac{k_1+k_2}{2}\right)\left(\frac{k_1}{k_2}\right)^{k_1/2}\, x^{k_1/2-1}}{\Gamma\left(\frac{k_1}{2}\right)\Gamma\left(\frac{k_2}{2}\right)\left(1+\frac{k_1}{k_2}x\right)^{(k_1+k_2)/2}} & x \in (0,\infty) \\ 0 & \text{Otherwise} \end{cases}$$

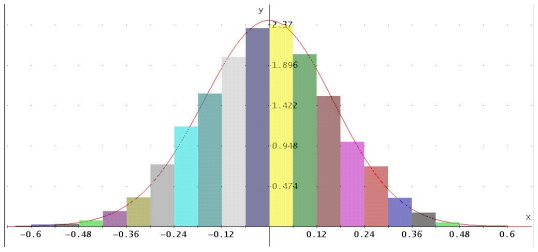
To generate $\mathcal{X} \rightsquigarrow F(k_1, k_2)$, y_1 and y_2 must be generated from $\chi^2(k_1)$ and $\chi^2(k_2)$ respectively and then return $\boxed{x = \dfrac{y_1/k_1}{y_2/k_2}}$ as the generated value.

3.11 Z distribution

Let $\mathcal{Y} \rightsquigarrow \mathcal{F}(k_1, k_2)$. Then, the variable $\mathcal{X} = \dfrac{\ln(\mathcal{Y})}{2}$ has a $\mathcal{Z}$ distribution with k_1 and k_2 degrees of freedom ($\mathcal{X} \rightsquigarrow \mathcal{Z}(k_1, k_2)$).

Its probability density function is given by:

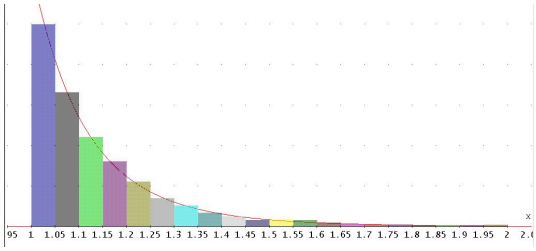
$$f_{\mathcal{X}}(x) = \frac{2\,\Gamma\left(\frac{k_1+k_2}{2}\right)\left(\frac{k_1}{k_2}\right)^{k_1/2}\,\mathrm{e}^{k_1 x}}{\Gamma\left(\frac{k_1}{2}\right)\Gamma\left(\frac{k_2}{2}\right)\left(1+\frac{k_1\,\mathrm{e}^{2x}}{k_2}\right)^{(k_1+k_2)/2}} \qquad x \in \mathbb{R}$$



$\mathcal{Z}(32, 45)$

To generate $\mathcal{X} \rightsquigarrow \mathcal{Z}(k_1, k_2)$, y must be generated from $\mathcal{F}(k_1, k_2)$ and then return $\boxed{x = \dfrac{\ln(y)}{2}}$ as the generated value.

3.12 Pareto distribution



$\mathcal{P}\mathrm{a}(8, 1)$

A random variable $\mathcal{X}$ has a Pareto distribution with parameters $\alpha > 0$ and $x_0 > 0$ ($\mathcal{X} \rightsquigarrow \mathcal{P}\mathrm{a}(\alpha, x_0)$) if its probability density function is:

$$f_{\mathcal{X}}(x) = \begin{cases} \dfrac{\alpha}{x_0}\left(\dfrac{x_0}{x}\right)^{\alpha+1} & x \in [x_0, \infty) \\ 0 & \text{otherwise} \end{cases}$$

To generate $\mathcal{X}$, the inverse transform method can be used since:

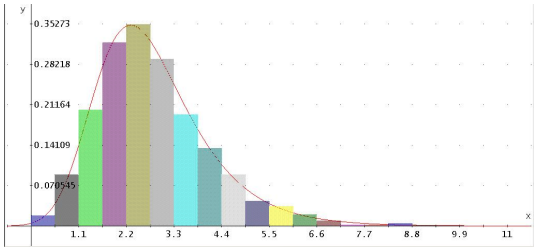
$$\mathcal{U} = F_{\mathcal{X}}(x) = \int_{x_0}^{x} \frac{\alpha}{x_0}\left(\frac{x_0}{t}\right)^{\alpha+1} \mathrm{d}t = 1 - \left(\frac{x_0}{x}\right)^{\alpha} \quad \Longrightarrow \quad \mathcal{X} = \frac{x_0}{(1-\mathcal{U})^{1-\alpha}} \equiv \frac{x_0}{\mathcal{U}^{1-\alpha}}$$

Thus, to generate $\mathcal{X} \rightsquigarrow \mathcal{P}\mathrm{a}(\alpha, x_0)$, u must be generated from $\mathcal{U}(0, 1)$ and then deliver $\boxed{\dfrac{\mathtt{x_0}}{\mathtt{u}^{1/\alpha}}}$ as the generated value.

3.13 Logistic distribution

A random variable $\mathcal{X}$ has a Logistic distribution with parameter $\alpha > 0$ ($\mathcal{X} \rightsquigarrow \mathcal{L}(\alpha)$) if its probability density function is:

$$f_{\mathcal{X}}(x) = \frac{\alpha\,\mathrm{e}^{-x}}{(1+\mathrm{e}^{-x})^{\alpha+1}} \qquad x \in \mathbb{R}$$



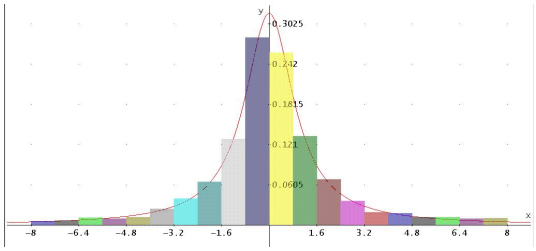
$\mathcal{L}(10)$

To generate $\mathcal{X}$, the inverse transform method can be used since:

$$\mathcal{U} = F_{\mathcal{X}}(x) = \int_{-\inf}^{x} \frac{\alpha\ \mathrm{e}^{-t}}{(1+\mathrm{e}^{-t})^{\alpha+1}}\,\mathrm{d}t = \frac{1}{(1+\mathrm{e}^{-x})^{\alpha}} \quad \Longrightarrow \quad \mathcal{X} = -\ln\left(\frac{1-\mathcal{U}^{1/\alpha}}{\mathcal{U}^{1/\alpha}}\right)$$

Thus, in order to generate $\mathcal{X} \rightsquigarrow \mathcal{L}(\alpha)$, u must be generated from $\mathcal{U}(0,1)$ and then deliver $\boxed{-\ln\left(\frac{1-u^{1/\alpha}}{u^{1/\alpha}}\right)}$ as the generated value.

3.14 Cauchy distribution



$\mathcal{C}(0,1)$

A random variable $\mathcal{X}$ has a Cauchy distribution with parameters $\alpha \geq 0$ and $\beta > 0$ ($\mathcal{X} \rightsquigarrow \mathcal{C}(\alpha,\beta)$) if its probability density function is:

$$f_{\mathcal{X}}(x) = \frac{\beta}{\pi\left[\beta^2 + (x-\alpha)^2\right]} \qquad x \in \mathbb{R}$$

To generate $\mathcal{X}$, the inverse transform method can be used since:

$$\mathcal{U} = F_{\mathcal{X}}(x) = \int_{-\infty}^{x} \frac{\beta}{\pi\left[\beta^2 + (t-\alpha)^2\right]}\,\mathrm{d}t = \frac{1}{2} + \frac{\operatorname{atan}\left(\frac{x-\alpha}{\beta}\right)}{\pi} \quad \Longrightarrow \quad \mathcal{X} = \alpha + \beta\tan\left[\pi\left(\mathcal{U} - \frac{1}{2}\right)\right]$$

Thus, in order to generate $\mathcal{X} \rightsquigarrow \mathcal{C}(\alpha,\beta)$, u must be generated from $\mathcal{U}(0,1)$ and then deliver $\boxed{\alpha + \beta\tan\left[\pi\left(u - \frac{1}{2}\right)\right]}$ as the generated value.

3.15 Irwin-Hall distribution

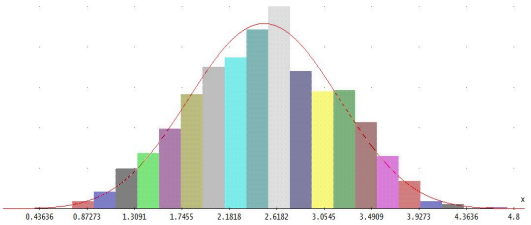
Let $\mathcal{U}_k \rightsquigarrow \mathcal{U}(0,1)$ $k = 1,\ldots,n$ n uniform independent distributions. In this case, $\mathcal{X} = \sum_{k=1}^{n} \mathcal{U}_k$ follows the Irwin-Hall distribution with parameter n ($\mathcal{X} \rightsquigarrow \mathcal{IH}(n)$).

Its probability density function is given by:

$$f_{\mathcal{X}}(x) = \frac{1}{2\,(n-1)!} \sum_{k=0}^{n} (-1)^k \binom{n}{k} (x-k)^{n-1} \operatorname{sgn}(x-k)$$

with $x \in [0,n]$ and $\operatorname{sgn}(x)$ the function given by:

$$\operatorname{sgn}(x) = \begin{cases} -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ 1 & \text{if } x > 0 \end{cases}$$



$\mathcal{IH}(5)$

To generate $\mathcal{X} \rightsquigarrow \mathcal{IH}(n)$, $u_1, u_2, \ldots, u_n$ values must be generated from $\mathcal{U}(0,1)$ and then return $\boxed{x = \sum_{k=1}^{n} u_k}$ as the generated value.

4 Discrete distributions random generation

In this section different procedures are presented in order to generate discrete distributions. The inverse transform method for discrete distributions described in section 2.4 is used in order to generate a sample of the distribution. The only thing to do is to use the `random_discrete` function developed in the same section with the corresponding parameters.

4.1 Uniform discrete distribution

$\mathcal{X} = \{a, a+1, \ldots, b\}$ has an uniform discrete distribution with parameters a and b ($\mathcal{X} \rightsquigarrow \mathcal{U}_{\mathcal{D}}(a,b)$) if its distribution mass function F is:

$$F(x) = P[\mathcal{X} = x] = \frac{1}{b-a+1} \qquad ; \qquad x = a, a+1, \ldots, b$$

Thus, in order to generate a sample of size n from an $\mathcal{U}_{\mathcal{D}}(a,b)$, the following DERIVE code can be used:

```
random_uniform_discrete(n := 1, a := 0, b := 1) :=
    random_discrete(n,1/(b-a+1),a)
```

4.2 Bernouille distribution

$\mathcal{X} = \{0,1\}$ has a Bernouille distribution with parameter p ($\mathcal{X} \rightsquigarrow \mathcal{B}\text{er}(p)$) if its distribution mass function F is:

$$F(x) = P[\mathcal{X} = x] = p^x (1-p)^{1-x} \qquad ; \qquad x = 0, 1$$

or, equivalently, $F(0) = 1 - p$ and $F(1) = p$.

Thus, in order to generate a sample of size n from a $\mathcal{B}\text{er}(p)$, the following DERIVE code can be used:

```
random_bernouille(n := 1, p := 1/2) := random_discrete(n,p^x(1 - p)^(1-x),0)
```

4.3 Rademacher distribution

$\mathcal{X} = \{-1, 1\}$ has a Rademacher distribution ($\mathcal{X} \rightsquigarrow \mathcal{R}\text{ad}$) if its distribution mass function F is:

$$F(x) = P[\mathcal{X} = x] = \frac{1}{2} \qquad ; \qquad x = -1, 1$$

If $\mathcal{Y} \rightsquigarrow \mathcal{B}\text{er}(1/2)$ then $\mathcal{X} = 2\mathcal{Y} - 1 \rightsquigarrow \mathcal{R}\text{ad}$. Thus, in order to generate a sample of size n from a $\mathcal{R}\text{ad}$, the following DERIVE code can be used:

```
random_rademacher(n := 1) := VECTOR(2k-1,k,random_bernouille(n,1/2))
```

4.4 Binomial distribution

$\mathcal{X} = \{0, 1, \ldots, n\}$ has a binomial distribution with parameters n and p ($\mathcal{X} \rightsquigarrow \mathcal{B}\mathrm{i}(n, p)$) if its distribution mass function F is:

$$F(x) = P[\mathcal{X} = x] = \left(\begin{array}{c} n \\ x \end{array} \right) p^x (1-p)^{n-x} \qquad ; \qquad x = 0, 1, \ldots, n$$

Thus, in order to generate a sample of size m from a $\mathcal{B}\mathrm{i}(n, p)$, the following DERIVE code can be used:

```
random_binomial(m := 1, n := 100, p := 1/2) :=
    random_discrete(m, ncom(n,x) p^x(1 - p)^(n-x),0)
```

4.5 Poisson distribution

$\mathcal{X} = \{0, 1, 2, \ldots\}$ has a poisson distribution of parameter λ ($\mathcal{X} \rightsquigarrow \mathcal{P}(\lambda)$) if its distribution mass function F is:

$$F(x) = P[\mathcal{X} = x] = \frac{\mathrm{e}^{-\lambda}\ \lambda^x}{x!} \qquad ; \qquad x = 0, 1, 2, \ldots$$

Thus, in order to generate a sample of size n from a $\mathcal{P}(\lambda)$, the following DERIVE code can be used:

```
random_poisson(n := 1, λ := 1) := random_discrete(n, e^(-λ) λ^x / x!,0)
```

4.6 Geometric distribution

$\mathcal{X} = \{0, 1, 2, \ldots\}$ has a Geometric distribution with parameter p ($\mathcal{X} \rightsquigarrow \mathcal{G}\mathrm{e}(p)$) if its distribution mass function F is:

$$F(x) = P[\mathcal{X} = x] = p\ (1-p)^x \qquad ; \qquad x = 0, 1, 2, \ldots$$

Thus, in order to generate a sample of size n from a $\mathcal{G}\mathrm{e}(p)$, the following DERIVE code can be used:

```
random_geometric(n := 1, p := 1/2) := random_discrete(n,p (1 - p)^x,0)
```

4.7 Negative Binomial distribution

$\mathcal{X} = \{0, 1, 2, \ldots\}$ has a negative binomial distribution with parameters r and p ($\mathcal{X} \rightsquigarrow \mathcal{NB}(r, p)$) if its distribution mass function F is:

$$F(x) = P[\mathcal{X} = x] = \left(\begin{array}{c} r+x-1 \\ x \end{array} \right) p^r (1-p)^x \qquad ; \qquad x = 0, 1, 2, \ldots$$

Thus, in order to generate a sample of size n from a $\mathcal{NB}(r, p)$, the following DERIVE code can be used:

```
random_negative_binomial(n := 1, r := 10, p := 1/2) :=
    random_discrete(n, ncom(r+x-1,x) p^r(1 - p)^x,0)
```

4.8 Hypergeometric distribution

$\mathcal{X} = \{\max(0, n_1 + m - n), \ldots, \min(n_1, m)\}$ has an hypergeometric distribution with parameters n, m and n_1 ($\mathcal{X} \rightsquigarrow \mathcal{H}(n, m, n_1)$) if its distribution mass function F is:

$$F(x) = P[\mathcal{X} = x] = \frac{\begin{pmatrix} n_1 \\ x \end{pmatrix} \begin{pmatrix} n - n_1 \\ m - x \end{pmatrix}}{\begin{pmatrix} n \\ m \end{pmatrix}} \quad ; \quad x = \max(0, n_1 + m - n), \ldots, \min(n_1, m)$$

Thus, in order to generate a sample of size l from a $\mathcal{NB}(r, p)$, the following DERIVE code can be used:

```
random_hypergeometric(l:=1, n := 100, m := 50, n1 := 50) :=
    random_discrete(l, ncom(n1,x)ncom(n-n1,m-x)/ncom(n,m) ,max(0, n1 + m - n))
```

5 Approximative algorithms

In this section several algorithms used to generate different distributions by approximations are presented.

The main reason in order to use such algorithm is that they produce "good" samples and they are quite much faster than other "exact" algorithm. In fact, some of the algorithms described above are approximative algorithm (`random_gamma9`, `random_beta7` and `random_chi_square2` are approximative algorithms for $\mathcal{G}(\alpha, \beta)$, $\mathcal{B}e(\alpha, \beta)$ and $\chi^2(k)$ distributions which use $\mathcal{N}(\mu, \sigma)$ distribution as approximation).

In the followings subsections some other approximative algorithms which have been implemented in `Random_distribution` package are developed.

5.1 Approximation to Binomial distribution by Poisson distribution

Let $\mathcal{X} \rightsquigarrow \mathcal{B}i(n, p)$. If p is "small" then

$$\mathcal{B}i(n, p) \approx \mathcal{P}(n\,p)$$

Thus, the algorithm to generate a sample of size m from the binomial distribution approximated by a poisson distribution is:

```
random_binomial_approx_poisson(m:=1, n := 100, p:= 0.01) :=
    random_poisson(m,np)
```

5.2 Approximation to Binomial distribution by Normal distribution

Let $\mathcal{X} \rightsquigarrow \mathcal{B}i(n, p)$. If n is "large" and p is not close to 0 or 1, then

$$\mathcal{B}i(n, p) \approx \mathcal{N}\left(n\,p, \sqrt{n\,p\,(1-p)}\right)$$

It can be shown that "good" approximations are reached when $n\,p > 10$ with $0 << p < 0.5$ or $n\,(1-p) > 10$ with $0.5 < p << 1$ ($<<$ indicates less than but not close to).

On the other hand, as $\mathcal{B}i(n, p)$ has only nonnegative integer values, the nearest nonnegative integer to the value returned by $\mathcal{N}\left(n\,p, \sqrt{n\,p\,(1-p)}\right)$ must be chosen as the generated value for

$\mathcal{X}$. It is easy to understand that the nearest nonnegative integer to a value z can be found by the operation:

$$\max\left(0, \text{floor}\left(z+\frac{1}{2}\right)\right)$$

where $\text{floor}(x)$ is the integer part of x.

Thus, the algorithm to generate a sample of size m from the binomial distribution approximated by a normal distribution is:

```
random_binomial_approx_normal(m:=1, n := 100, p:= 1/4) :=
    vector(max(0,floor(k+1/2)),k,random_normal(m,np,sqrt(np(1-p))))
```

5.3 Approximation to Poisson distribution by Normal distribution

Let $\mathcal{X} \rightsquigarrow \mathcal{P}(\lambda)$. If $\lambda > 10$ then $\mathcal{P}(\lambda) \approx \mathcal{N}\left(\lambda, \sqrt{\lambda}\right)$

On the other hand, as $\mathcal{P}(\lambda)$ has only nonnegative integer values, the nearest nonnegative integer to the value returned by $\mathcal{N}\left(\lambda, \sqrt{\lambda}\right)$ must be chosen as the generated value for $\mathcal{X}$. As shown in section 5.2, this can be easily done by the operation:

$$\max\left(0, \text{floor}\left(z+\frac{1}{2}\right)\right)$$

Thus, the algorithm to generate a sample of size n from the poisson distribution approximated by a normal distribution is:

```
random_poisson_approx_normal(n := 1, λ:= 15) :=
    vector(max(0,floor(k+1/2)),k,random_normal(n,λ,sqrt(λ)))
```

5.4 Approximation to Geometric distribution by Exponential distribution

If $\mathcal{X} \rightsquigarrow \mathcal{G}\text{e}(p)$ then $\mathcal{X} \approx \mathcal{E}\left(\ln\left(\frac{1}{1-p}\right)\right)$

On the other hand, as $\mathcal{G}\text{e}(p)$ has only nonnegative integer values, the nearest nonnegative integer to the value returned by $\mathcal{E}\left(\ln\left(\frac{1}{1-p}\right)\right)$ must be chosen as the generated value for $\mathcal{X}$. The nearest nonnegative integer to a value $z > 0$ (exponential distribution has only positive real numbers) can be found by the operation:

$$\text{floor}\left(z+\frac{1}{2}\right)$$

where $\text{floor}(x)$ is the integer part of x.

Thus, the algorithm to generate a sample of size n for the Geometric distribution approximated by a exponential distribution is:

```
random_geometric_approx_exponential(n := 1, p:= 1/2) :=
    vector(floor(k+1/2),k,random_exponential(n,ln(1/(1-p))))
```

6 Graphical Approach

In order to check graphically if the generated sample fix the distribution, two different graphical functions have been developed.

- `bars_diagram(v, k, mi, ma)`: it returns the needed instructions to plot the bars diagram of vector `v` divided in `k` classes (`k`=20, by default). The optional parameters `mi` (minimum) and `ma` (maximum) are the minimum and maximum values of vector `v` unless other values are considered. This values will be used to set the plot range.

- `density_function(name, para)`: it returns the corresponding density function of distribution `name` which parameters are set in vector `para`. The possible values for `name` are: "Uniform", "Exponential", "Normal", "Lognormal", "Cauchy", "Weibul", "Gamma", "Beta", "Chi-square", "t", "F", "Z", "Pareto", "Logistic" or "IrwinHall".

  Note that these functions have been developed for continuous distributions.

7 Examples

As an example, let us describe the steps needed to obtain a sample of size $n = 1000$ from a Normal distribution with parameters $\mu = 3$ and $\sigma = 2$:

1. Load the utility file `Random_distributions.mth` in DERIVE 6.1.
2. Write down the following instruction in the Author Line: `random_normal(1000,3,2)`
3. Do not simplify but Approximate the expression (in order case, time would be too long).
4. Variable `res` stores the result.

   Now, in order to check graphically if the generated sample fix the distribution, the steps to follow are:

5. Approximate the expression: `bars_diagram(res)`
6. Follows the instructions given after this execution (the appropriate range for the plot) and plot the result.
7. Simplify the expression: `density_funciton("Normal",[3,2])` and plot the result.

Remind that `res` variable stores the sample (of size $n = 1000$ in the example). If, the sample does not fix the density function, a new sample can be generated again going to step 1.

In order to see more examples of different generated samples from discrete and continuous distributions, as well as, examples of different plots, please refer to files `Random_distributions.dfw` and `Random_distributions.mth`. These files are distributed together with this paper and are also allocate at `http://www.matap.uma.es/profesor/jl_galan/Derive/Packages`.

8 Future work

In this section some possible future work lines are described:

- *Development of different programs to generate other discrete or continuous distributions*: although in this work the most common distribution has been considered, there exist many more distributions which can be generated. This work shows different ways of generating samples for both discrete and continuous distributions. Even more, since some generic algorithm has been developed they may be used to generate almost any new distribution (mainly for discrete distributions).

- *Development of graphical approaches for discrete distributions*: since the graphical approach developed has been optimized for continuous distributions, some graphical approaches for discrete distributions could be also considered in future.

- *Development of applications*: there exist many applications which requires working with samples of different distributions. This kind of applications can be considered using this work.

- *Checking the obtained result theoretically*: although a graphical approach has been developed to check if the generated sample fix the corresponding distribution, different theoretical approach can be considered in order to check the results.

- *Translation of these algorithms*: Although this package has been developed in DERIVE, it can be easily translated to other Computer Algebra Systems (CAS) or even to a portable platform as JAVA.

References

[Galán, 1991] Galán, J. L. (1991). *Simulación de Variables Aleatorias*. Proyecto Fin de Carrera, Universidad de Málaga.

[Marsaglia and Zaman, 1994] Marsaglia, G. and Zaman, A. (1994). Some portable very-long-period random number generators. *Computers in Physics*, 8(1):117–121.

[Press and Teukolsky, 1992] Press, W. H. and Teukolsky, S. A. (1992). Portable Random Number Generators. *Computers in Physics*, 6(5):522–524.

[Press et al., 1999] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1999). *Numerical Recipes in C. The Art of Scientific Computing*. Cambridge University Press, United States.

[Rubinstein, 1981] Rubinstein, R. Y. (1981). *Simulation and the Monte Carlo Method*. John Wiley & Sons, New York.

[Wikipedia, 2009] Wikipedia.org, the free encyclopedia (2009). *List of Probability Distributions*. `http://en.wikipedia.org/wiki/List_of_probability_distributions`.

From: DERIVE computer algebra system [mailto:DERIVE-NEWS@JISCMAIL.AC.UK] On Behalf Of Volker Loose
Sent: Thursday, October 08, 2009 9:10 AM
To: DERIVE-NEWS@JISCMAIL.AC.UK
Subject: nsolve

nsolve gives only 1 solution of an equation. I tried to write a function vnsolve that gives more solutions, using the vector- and the nsolve- function.It seems to be a mistake in my function but I can't find it. Has anyone an idea?

Many thanks
Volker

#1: $f(x) := e^x - x - 2$

#2: $vnsolve(f, u, o, n) := VECTOR\left(\left[\begin{array}{c} i \\ u + \frac{(i-1)\cdot(o-u)}{n} \\ u + \frac{i\cdot(o-u)}{n} \\ NSOLVE\left(f\cdot x, x, u + \frac{(i-1)\cdot(o-u)}{n}, u + \frac{i\cdot(o-u)}{n}\right) \end{array}\right], i, 1, n, 1\right)$

#3: `vnsolve(f(x), -5, 5, 2)`

#4: $\left[\left[\begin{array}{c} 1 \\ -5 \\ 0 \\ x = 0 \end{array}\right], \left[\begin{array}{c} 2 \\ 0 \\ 5 \\ x = 0 \end{array}\right]\right]$

DNL: Do you find the bug? Volker defined in #1 function f(x) and then in #2 he used f as a parameter for his function. Please note in the 4th component of the vector `NSOLVE(f·x, …)`. This should be f(x) but is interpreted as a product f·x. Replace f in the parameter list and f·x in the NSOLVE-command by, say f_, then `vnsolve` works.

But if there are more solutions which are close then this procedure does not work properly.
Try `vnsolve(SIN(x^2)-e^x,-7,7,10)` which should give 16 zeros. `vnsolve` gives only 5.
Jim FitzSimmons suggested another procedure:

$$ITERATE\left(APPEND\left(NSOLUTIONS\left(\frac{f(x)}{\prod(x-y, y, v)}, x, -5, 5\right), v\right), v, [], 2\right)$$

[1.146193220, -1.841405660]

In case of more and close zeros it does not a satisfying job, too. In DNL#63 I tried to define a function finding the zeros. See `mysolutions`:

$mysolutions2(SIN(x^2) - e^x, x, -7, 7)$

[-6.864608198, -6.632014374, -6.390542007, -6.140135705, -5.878326268, -5.605319329, -5.316900028, -5.013919234, -4.688491003, -4.343104006, -3.960923705, -3.548961687, -3.062349296, -2.522602266, -1.720968219, -0.7149689691]

At the occasion of this exchange of e-mails Danny Ross Lunsford wrote:

about DERIVE-NEWS@JISCMAIL.AC.UK

I am very happy to see that this list still exists. Derive is still a nearly ideal math tool and I hope it lives on.

I have nothing to add, except this: I completely agree with Danny, Josef.