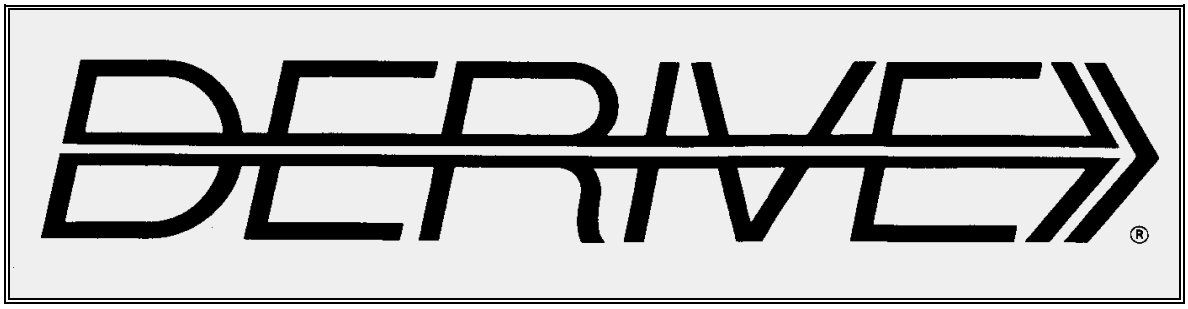
THE DERIVE - NEWSLETTER #127

ISSN 1990-7079

THE BULLETIN OF THE



USER GROUP

+ CAS-TI

Contents:

October 2022

Fred Tydeman sent the following mail

Von: DERIVE computer algebra system DERIVE-NEWS@JISCMAIL.AC.UK
Im Auftrag von Fred J. Tydeman
Gesendet: Donnerstag, 13. Oktober 2022 15:34
An: DERIVE-NEWS@JISCMAIL.AC.UK
Betreff: Integral with STEP()

Given:

0.0 <= z <= 0.5
0.0 <= y <= 0.5
0.0 <= x <= 0.5
0.5 <= w <= 1.0

I am trying to find what percentage of the time is the sum w+x+y+z greater than 1.5

It seems to me that the Derive expression:

INT(INT(INT(INT(STEP(w+x+y+z-1.5),z,0,0.5),y,0,0.5),x,0,0.5),w,0.5,1)

would find that.

Since, if each of the four variables is in the middle of their domain, the sum is the cutoff point I care about, it seems to me that the answer should be around 50%. Also, 1.5 is the middle of the range of possible sums, e.g., 0.5 <= sum <= 2.5

However, if I have Derive simplify the above expression, it gets 1/32.

Can someone see a flaw in my logic or my Derive expression?

--- Fred J. Tydeman Tydeman Consulting

Michel Beaudin gave an answer within a few hours:

If you divide your integral by the one of the "total volume" which is the integral with the same bounds but with 1 as integrand instead of the step function, you get ½ as expected. Is this too simple?

$$\frac{\int_{0.5}^{1}\int_{0}^{0.5}\int_{0}^{0.5}\int_{0}^{0.5} f\,dx\,dy\,dz\,dw}{\int_{0.5}^{1}\int_{0}^{0.5}\int_{0}^{0.5}\int_{0}^{0.5} 1\,dx\,dy\,dz\,dw} = \frac{1}{2}$$

Dear DUG-Members,

Yes, I know, this issue of our newsletter is very late this time. All what I can do is apologizing and trying to explain the reason(s):

(1) We had much work in house and garden during summer and fall. So, I started producing DNL#127 very (too) late.

(2) It needed some communication with the authors of the contributions. Communication with Michel Beaudin works perfectly but communication with Felipe (Mexico) was difficult because his internet connection was out of order for some time caused by tropical storms in his region.

(3) We spent two – great weeks – in Sicily. I must admit that we enjoyed travelling again after Covid-Lockdown (two pictures are below).

Concordia temple in Agrigento

Aetna (view from Taormina)

(4) And last, but not least, the contribution on Sparse Matrices needed much more work and time as expected.

I thought that it would be easy work to transfer Bhuvanesh Bhatt's TI-92 functions to TI-Nspire and DERIVE: my first problem was that I didn't know anything about treating sparse matrices – and to my surprise I found rich resources in the web. So, as many times before, the contribution received its momentum.

And, moreover, when I checked the TI-functions with some matrices I found out that they didn't work correctly – and it was not so easy for me to improve the code. I am quite sure that my code can be improved. I let two functions as a programming exercise for our members:

LIL-format of sparse matrices and SparseRnd for Nspire are not included (this might be a challenge for the Nspire-programmers!).

Many websites treat the sparse matrices based on PYTHON. Dear PYTHON users, it would be very welcome if you would demonstrate the functions written in PYTHON. You can find a rich selection of websites on the last page.

I was very much impressed by Felipe's the technical drawings produced with DERIVE.

Dear members, I hope that next the DNL will be in time and remain until December

Always Yours, Josef

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

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

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

Next issue: December 2022

Preview: Contributions waiting to be published

Some simulations of Random Experiments, J. Böhm, AUT, Lorenz Kopp, GER
Wonderful World of Pedal Curves, J. Böhm, AUT
Simulating a Graphing Calculator in *DERIVE*, J. Böhm, AUT
Cubics, Quartics – Interesting features, T. Koller & J. Böhm, AUT
Logos of Companies as an Inspiration for Math Teaching
Exciting Surfaces in the FAZ, BooleanPlots.mth, P. Schofield, UK
Old traditional examples for a CAS – What´s new? J. Böhm, AUT
Mandelbrot and Newton with *DERIVE*, Roman Hašek, CZ
Tutorials for the NSpireCAS, G. Herweyers, BEL
Dirac Algebra, Clifford Algebra, Vector-Matrix-Extension, D. R. Lunsford, USA
Another Approach to Taylor Series, D. Oertel, GER
Charge in a Magnetic Field, H. Ludwig, GER
More Applications of TI-Innovator™ Hub and TI-Innovator™ Rover
Surfaces and their Duals, Cayley Symmetroid, J. Böhm, AUT
A Collection of Special Problems, W. Alvermann, GER
DERIVE Bugs? D. Welz, GER
Tweening & Morphing with TI-NspireCX-II-T, J. Böhm. AUT
The Gap between Poor and Rich, J. Böhm, AUT
More functions from M. Myers and from Bhuvanesh's Mathtools-library
TaxiCab Conics, Two alternate Approaches to Conics, R. Haas, USA
QR-Code light, Random numbers following a given distribution
Properties of Polygons, Quartiles, F. de Jesús Martínez Vargas, Mexico
Penalty shootout mathematics, B. Grabinger, GER
153 is another Special Number, and others

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

In the last issues of our DNL I was pleased to present Michel Beaudin's Résumé 1 in five parts. Michel provided his students with a bundle of exercises. He sent this list together with some of the problems solved. (I allowed myself to add DERIVE-solutions.)

Exercise list for Summary 1

Problems 1 Solving an equation by the fixed-point method and by Newton's method

a) Use the fixed-point method to solve the following equation: $x\cosh(x)=1$. What happens with the "similar" equation $x\cosh(3x)=1$? Resolve each of these two equations using Newton's method.

b) Use the fixed-point method to find the smallest positive solution of the following equation: $x=\tan(x)$. We'll rewrite the equation in the form $x=\arctan(x)+\pi$ (why by the way?).

c) Use the fixed-point method to find the only real solution to the equation $x^3+6x-3=0$ (give a simple reason why there cannot be more than one real solution here). Resume with Newton's method.

d) Use the fixed-point method to find the only real solution to the equation $2\arcsin(x)=\arccos(3x)$. Then find the exact solution, too!

e) Use Newton's method to find one of the six solutions of the polynomial equation

$$p(z)\equiv z^6+z^4-12z^3+28z^2-48z+161=0.$$

Use the 3D graph of $|p(z)|^2$ or the 2D graphs of the implicit equations $u(x,y)=0$ and $v(x,y)=0$ where $p(x+iy)=u(x,y)+iv(x,y)$ to find a starting point for the procedure.

f) Consider the equation $x^6=(1.1)^{-x}$.

i) Find the smallest real solution of the above equation by the fixed-point method. Make sure that the assumptions of the theorem are satisfied.

ii) Also find a complex solution in the neighborhood of the origin using Newton's method (one variable with complex starting point). To find this starting point, consider the curves of the real and imaginary parts of the equation (implicit 2D graphics required!).

Solution of Problem 1a)

Use the fixed-point method to solve the following equation: $x\cosh(x)=1$. What happens with the "similar" equation $x\cosh(3x)=1$? Resolve each of these two equations using Newton's method.

At the beginning recall that the fixed-point method is suitable for an equation of the form $x = g(x)$. If we have verified that the continuous function g sends a closed interval I into itself, then there exists at least one solution r of the equation $g(x) = x$. If moreover g is derivable on the interior of I and there exists a constant K such that $0<|g'(x)|\le K<1,$ then this solution is unique and can be obtained by iteration using the sequence defined by $x_{n+1}=g(x_n)$. This sequence converges whatever the starting point x_0 chosen in I might be.

As for Newton's method, it is used to solve an equation of the form $f(x) = 0$ where function f is sufficiently derivable. Assumptions such as that $|f'(x)|$ is minorized and that $|f''(x)|$ is majorized on an interval containing the root r we are looking for assure us that the sequence defined by $x_{n+1}=x_n-\dfrac{f(x_n)}{f'(x_n)}$ converges to r.

A graph, on same axes, of the curves $y = x\cosh(x)$ and $y = 1$ shows an intersection in the neighborhood of $x = 0.75$. And for the curves $y = x\cosh(3x)$ and $y = 1$, a graph on same axes shows an intersection around $x = 0.45$. See figure 1:

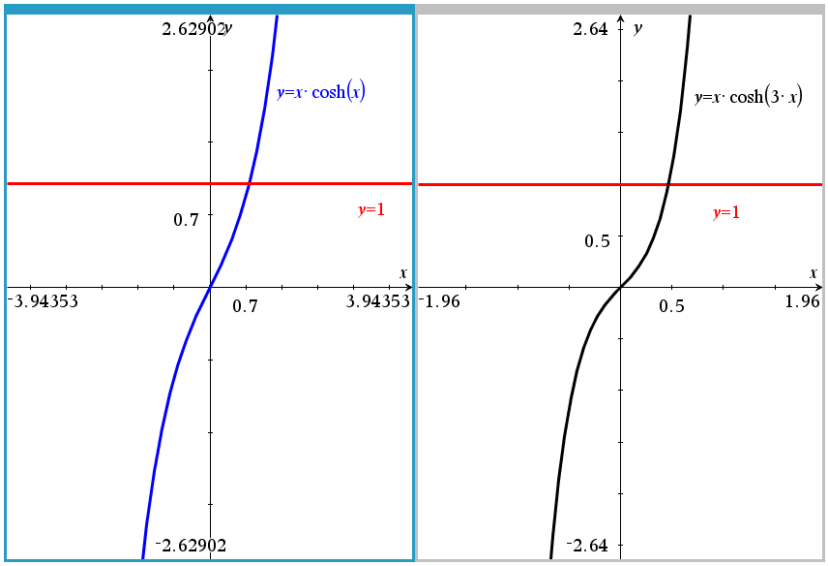


Figure 1

On the other hand, the fixed-point method does not seem to work for the second equation, but it does for the first one. Newton's method will work for both equations.

To apply the fixed point method, it is natural and easy to isolate the "x" in the right side of the function equation: $x = \dfrac{1}{\cosh(x)}$. This is all the more justified since the cosh function never will become zero: $\cosh(x) \equiv \dfrac{e^x + e^{-x}}{2}$. Here is a table of iterations of the sequence $x_{n+1} = g(x_n)$: we see a convergence for the sequence $x_{n+1} = \dfrac{1}{\cosh(x_n)}$, $x_0 = 0.76$ and the "cobweb" confirms that also starting from $x_0 = 0.5$: see figure 2.

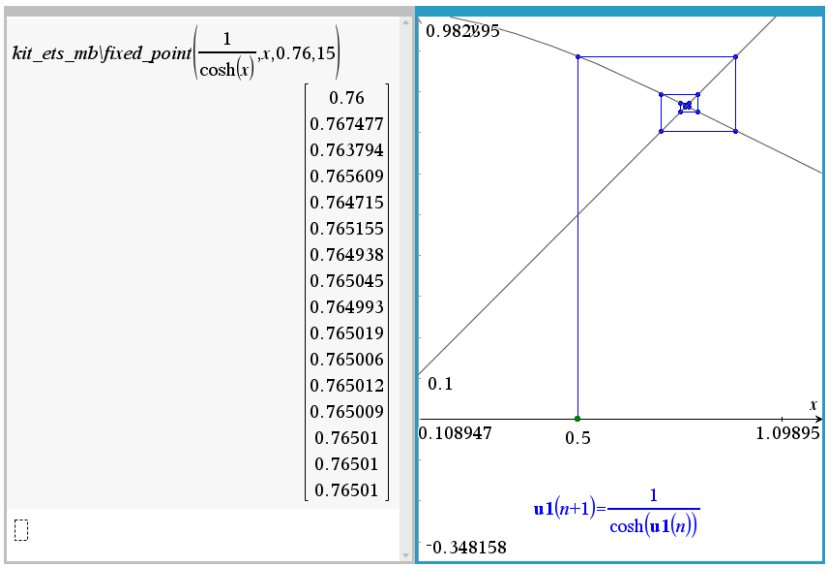


Figure 2

This is not surprising since we can easily verify by graphs that the function $g(x)=\dfrac{1}{\cosh(x)}$ sends the interval [0, 1] into itself and that the absolute value of its derivative is always less than a constant $K < 1$. Therefore, whatever the starting point in the interval [0, 1], the sequence would have converged to 0.76501.

On the other hand, the function $g(x)=\dfrac{1}{\cosh(3x)}$ also sends the interval [0, 1] into itself but the derivative of its absolute value is above 1. The cobweb shows that the sequence $x_{n+1}=\dfrac{1}{\cosh(3x_n)}$, $x_0=0.45$ oscillates between two values: see figure 3.

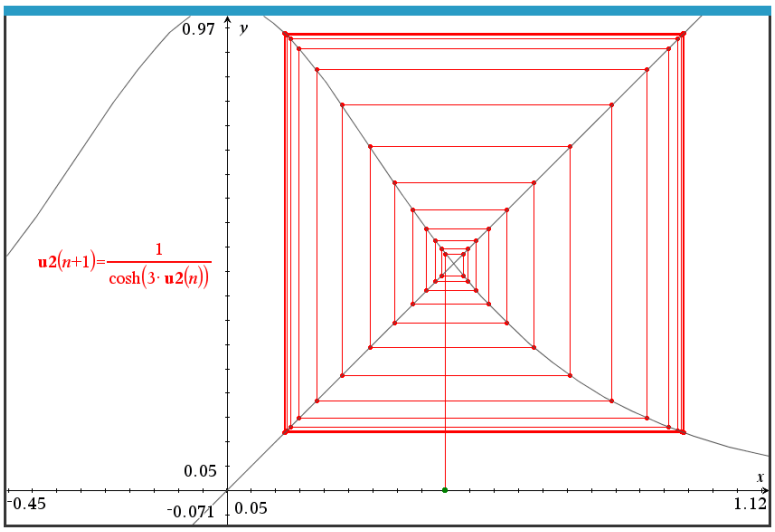


Figure 3

Finally, Newton's method converges very quickly for both equations: Figure 4 (left part) shows how the method can be applied by calculating the function $x-\dfrac{f(x)}{f'(x)}$, while the right part shows the use of a home-made function from the kit_ets_mb library, where one does not necessarily have to define "*f*" as a function: an expression is sufficient:

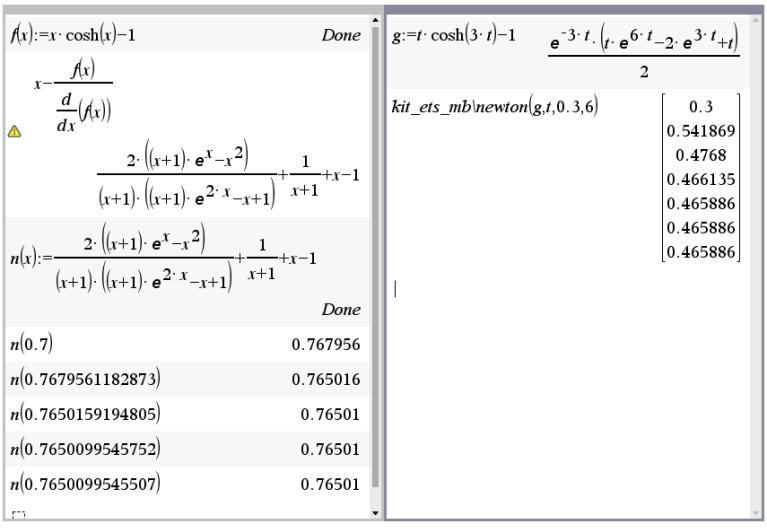


Figure 4

How to treat problem 1a with DERIVE:

$$\left[\text{ITERATES}\left(\frac{1}{\text{COSH}(x)}, x, 0.76, 15\right)\right]'$$

$$\begin{bmatrix} 0.76 \\ 0.7674765988 \\ 0.7637942932 \\ 0.7656087904 \\ 0.7647148964 \\ 0.7651553182 \\ 0.7649383353 \\ 0.7650452396 \\ 0.7649925701 \\ 0.7650185194 \\ 0.7650057347 \\ 0.7650120335 \\ 0.7650089302 \\ 0.7650104591 \\ 0.7650097059 \\ 0.7650100770 \end{bmatrix}$$

Cobweb(g,x,a,n) produces a set of matrices which will plot a cobweb diagram for the fixpoint iterative scheme x=g(x).

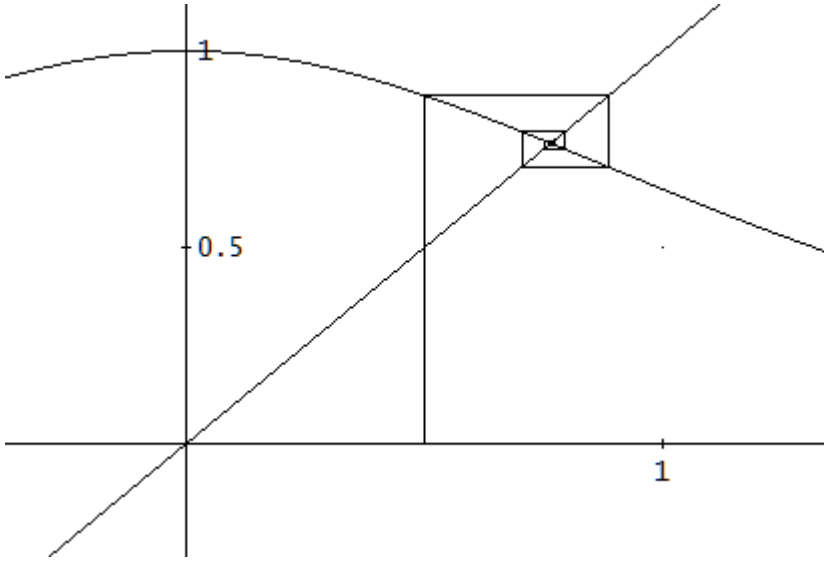
Ensure that in the plot window you set Set > Coordinate System > Rectangular and Option > Display > Points > Connected & Small

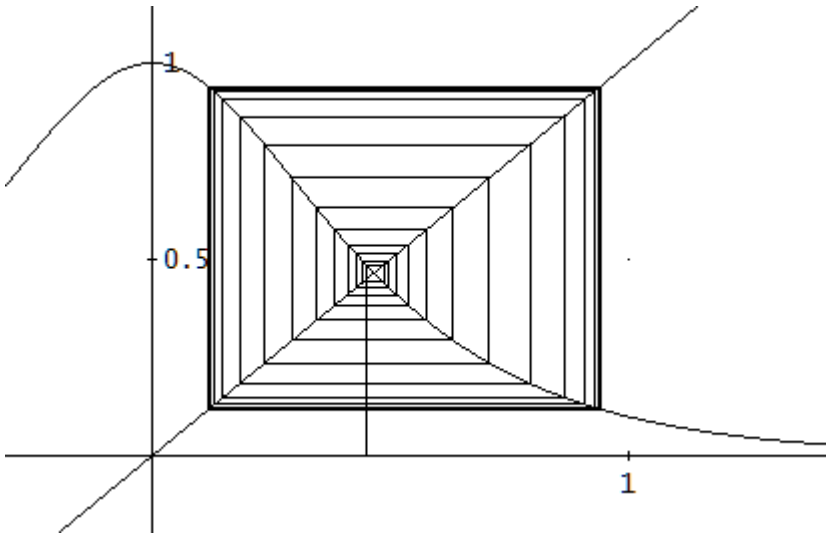
$$\text{COBWEB\_AUX}(v, g, x, n) := \left[x, g, \begin{bmatrix} v_1 & 0 \\ & \\ v_1 & v_2 \end{bmatrix}, \text{VECTOR}\left(\begin{bmatrix} v_r & v_{r+1} \\ & \\ v_{r+1} & v_{r+1} \\ & \\ v_{r+1} & v_{r+2} \end{bmatrix}, r, 1, n-1\right)\right]$$

$$\text{COBWEB}(g, x, a, n) := \text{COBWEB\_AUX}(\text{ITERATES}(g, x, a, n), g, x, n)$$

$$\text{COBWEB}\left(\frac{1}{\text{COSH}(x)}, x, 0.5, 20\right)$$

$$\text{COBWEB}\left(\frac{1}{\text{COSH}(3 \cdot x)}, x, 0.45, 200\right)$$





[NEWTON(t·COSH(3·t) − 1, t, 0.3, 6)]'

$$\begin{bmatrix} 0.3 \\ 0.5418692532 \\ 0.4767998584 \\ 0.4661350255 \\ 0.4658864571 \\ 0.4658863258 \\ 0.4658863258 \end{bmatrix}$$

$$f(x) := x \cdot \mathrm{COSH}(x) - 1$$

$$n(x) := x - \frac{f(x)}{f'(x)}$$

$$n(0.7) = 0.7679561182$$

$$n(0.7679561182) = 0.7650159194$$

$$n(0.7650159194) = 0.7650099545$$

$$n(0.7650099545) = 0.7650099545$$

Problems 2 Solving third degree polynomial equations. Cardan's formula.

a) Take the equation $x^3+6x-3=0$ (problem 1c) and show that its exact solution is

$$\frac{\left(4\sqrt{41}+12\right)^{1/3}}{2}-\frac{\left(4\sqrt{41}-12\right)^{1/3}}{2}.$$

b) Show that the three solutions of the equation $x^3+6x-3=0$ obtained by simply changing the sign of the coefficient of the linear term in the previous question are given by

$$\left\{-2\cdot\sqrt{2}\cdot\sin\left(\frac{\sin^{-1}\left(\frac{3\cdot\sqrt{2}}{8}\right)}{3}\right),2\cdot\sqrt{2}\cdot\sin\left(\frac{\sin^{-1}\left(\frac{3\cdot\sqrt{2}}{8}\right)}{3}+\frac{\pi}{3}\right),-2\cdot\sqrt{2}\cdot\cos\left(\frac{\sin^{-1}\left(\frac{3\cdot\sqrt{2}}{8}\right)}{3}+\frac{\pi}{6}\right)\right\}$$

c) Find, using the Cardan formula (after getting rid of the quadratic term) the real solution of the equation $2x^3+\frac{1}{2}x^2-4x+10=0.$

d) Find, using an appropriate trigonometric substitution (after getting rid of the quadratic term) the three real solutions of the equation $2x^3+x^2-4x-1=0.$

e) It can be difficult to simplify nested radicals. Test this by using the Cardan formula to find the real root of the equation $x^3+6x-20=0.$ A moment of attention shows that this solution is however $x = 2$. Moreover, one could (quite quickly) find $x = 2$ using the rational roots theorem by considering the divisors of 1 and -20.

Solution of Problem 2c)

Find, using the Cardan formula (after getting rid of the quadratic term) the real solution of the equation $2x^3+\frac{1}{2}x^2-4x+10=0.$

Note that the graph of the polynomial $2x^3+\frac{1}{2}x^2-4x+10=0$ shows a single real root but the presence of two critical points. Indeed, the derivative $6x^2+x-4$ has two zeros. Also, the equation is equivalent to $4x^3+x^2-8x+20=0$ and if there is a rational zero p/q, then p must be a divisor of 20 and q a divisor of 4. The possibilities are thus for p: ±1, ±2, ±4, ±5, ±10 and ±20 and for q: ±1, ±2 and ±4. So this rational number would be found among the following numbers: ±1, ±1/2, ±1/4, ±2, ±4, ±5, ±5/2, ±5/4, ±10 and ±20. As the direct calculation shows, none of these numbers is a zero of the given polynomial:

$pol(x):=4\cdot x^3+x^2-8\cdot x+20$ — *Done*

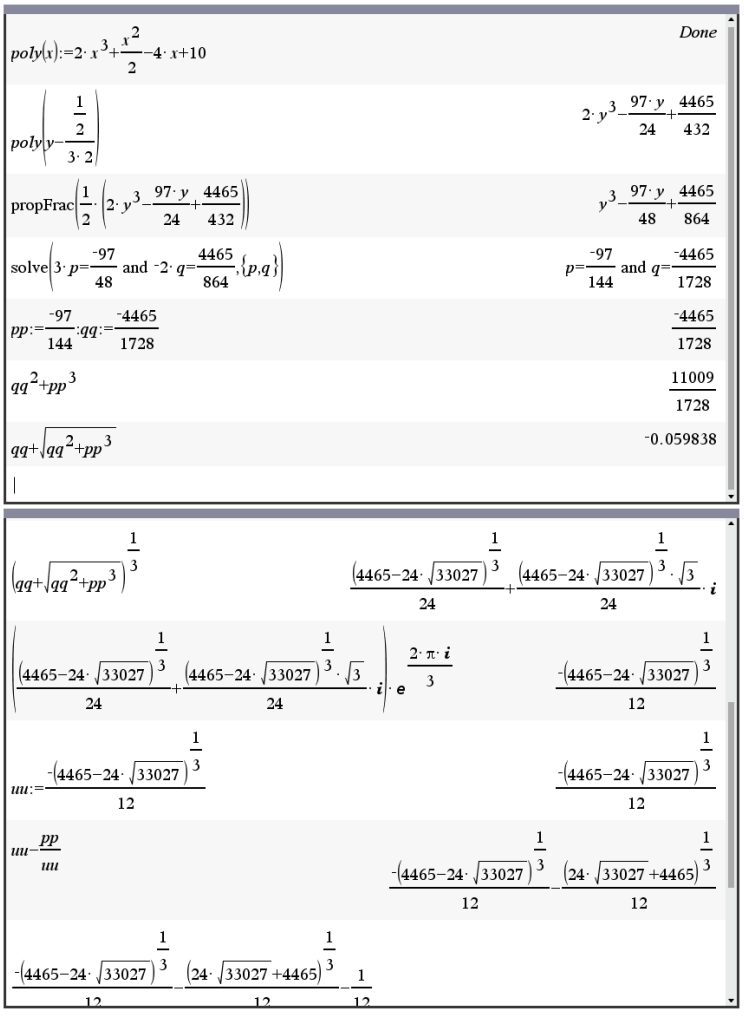
$pol\left(\left\{1,\frac{1}{2},\frac{1}{4},2,4,5,\frac{5}{2},\frac{5}{4},10,20\right\}\right)$ $\left\{17,\frac{67}{4},\frac{145}{8},40,260,505,\frac{275}{4},\frac{155}{8},4040,32260\right\}$

$pol\left(\left\{-1,\frac{-1}{2},\frac{-1}{4},-2,-4,-5,\frac{-5}{2},\frac{-5}{4},-10,-20\right\}\right)$ $\left\{25,\frac{95}{4},22.8,-188,-415,\frac{-65}{4},\frac{95}{4},-3800,-31420\right\}$

Figure 5

To apply the Cardan formula, we will get rid of the quadratic term in the polynomial $\text{poly}(x) = 2x^3 + \frac{1}{2}x^2 - 4x + 10$. Then we will consider the polynomial resulting in the variable "y" of the form $y^3 + 3py - 2q$ (so its dominant coefficient will have been made equal to 1, which is allowed since the expression is 0). We will see that the expression $q^2 + p^3$ is positive.

On the other hand, one must be careful when calculating the "cubic root" of $q + \sqrt{q^2 + p^3}$ since if p is negative, it is then possible that $q + \sqrt{q^2 + p^3}$ is negative. And this is precisely the case in our problem as shown in figure 6 below where we see that it is about –0.059838. And the "cubic root" of a negative number is a complex number (since the software works with the "main branch": in Nspire, the "settings" had to be changed). We will have to choose the real solution of the equation $u^3 = q + \sqrt{q^2 + p^3}$. This can also be done by calculating $\left(q + \sqrt{q^2 + p^3}\right)^{1/3}$ and multiplying by $e^{2\pi i/3}$. By posing u this result, we know that then $y = u - \frac{p}{u}$ and we derive the value of "x" from it. Figure 6 shows a session of Nspire with all the calculations to get there. Obviously, the answer obtained is approximated in 2.18667 which is the solution returned by a numerical solver.



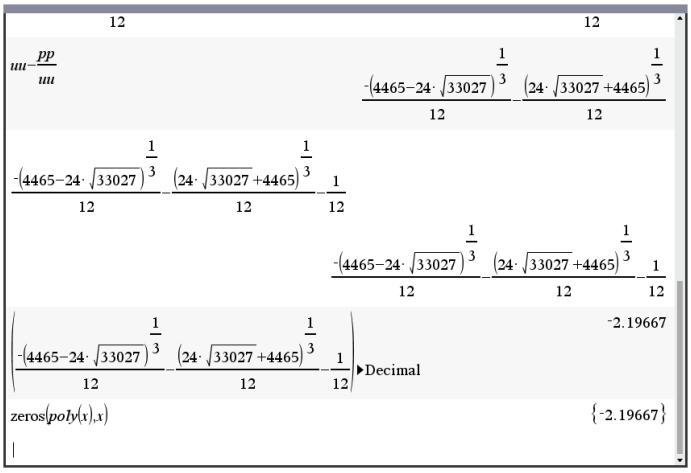


Figure 6

This is the solution procedure using DERIVE:

#1: $poly(x) := 2 \cdot x^3 + \frac{x^2}{2} - 4 \cdot x + 10$

#2: $EXPAND\left(\frac{1}{2} \cdot \left(2 \cdot x^3 + \frac{x^2}{2} - 4 \cdot x + 10\right)\right) = x^3 + \frac{x^2}{4} - 2 \cdot x + 5$

#3: $EXPAND\left(poly\left(y - \frac{\frac{1}{2}}{3 \cdot 2}\right)\right) = 2 \cdot y^3 - \frac{97 \cdot y}{24} + \frac{4465}{432}$

#4: $EXPAND\left(\frac{1}{2} \cdot \left(2 \cdot y^3 - \frac{97 \cdot y}{24} + \frac{4465}{432}\right)\right) = y^3 - \frac{97 \cdot y}{48} + \frac{4465}{864}$

#5: $SOLUTIONS\left(\left[3 \cdot p = -\frac{97}{48}, -2 \cdot q = \frac{4465}{864}\right], [p, q]\right) = \left[\left[-\frac{97}{144}, -\frac{4465}{1728}\right]\right]$

#6: $\left[pp := -\frac{97}{144}, qq := -\frac{4465}{1728}\right]$

#7: $qq^2 + pp^3 = \frac{11009}{1728}$

#8: $qq + \sqrt{(qq^2 + pp^3)} = \frac{\sqrt{33027}}{72} - \frac{4465}{1728}$

#9: $(qq + \sqrt{(qq^2 + pp^3)})^{1/3} = \frac{(4465 - 24 \cdot \sqrt{33027})^{1/3}}{24} + \frac{i \cdot (13395 \cdot \sqrt{3} - 216 \cdot \sqrt{11009})^{1/3}}{24}$

$$\#10:\quad uu := \left(\frac{(4465 - 24\cdot\sqrt{33027})^{1/3}}{24} + \frac{i\cdot(13395\cdot\sqrt{3} - 216\cdot\sqrt{11009})^{1/3}}{24}\right)\cdot e^{2\cdot\pi\cdot i/3}$$

$$\#11:\quad uu := -\frac{(4465 - 24\cdot\sqrt{33027})^{1/3}}{12}$$

$$\#12:\quad uu - \frac{pp}{uu} = -\frac{(4465 - 24\cdot\sqrt{33027})^{1/3}}{12} - \frac{(24\cdot\sqrt{33027} + 4465)^{1/3}}{12}$$

$$\#13:\quad uu - \frac{pp}{uu} - \frac{1}{12} = -\frac{(4465 - 24\cdot\sqrt{33027})^{1/3}}{12} - \frac{(24\cdot\sqrt{33027} + 4465)^{1/3}}{12} - \frac{1}{12}$$

$$\#14:\quad uu - \frac{pp}{uu} - \frac{1}{12} = -2.196666328$$

$$\#15:\quad \text{SOLUTIONS}\left(2\cdot x^3 + \frac{x^2}{2} - 4\cdot x + 10 = 0,\ x,\ \text{Real}\right)$$

$$\#16:\quad \left[-\frac{(4465 - 24\cdot\sqrt{33027})^{1/3}}{12} - \frac{(24\cdot\sqrt{33027} + 4465)^{1/3}}{12} - \frac{1}{12}\right]$$

#17: [-2.196666328]

Solution of Problem 2d)

Find, using an appropriate trigonometric substitution (after getting rid of the quadratic term) the three real solutions of the equation $2x^3 + x^2 - 4x - 1 = 0$.

Note that the polynomial $p(x) = 2x^3 + x^2 - 4x - 1 = 0$. changes sign between 2 and –1, between –1 and 0 and between 1 and 2

$$p(-2) = -5 < p(-1) = 2,\ p(-1) = 2 > p(0) = -1 \text{ and } p(1) = -2 < p(2) = 11.$$

By continuity, p thus has three real roots. The Nspire session in Figure 7 gives the solution (the polynomial p is called *fl* here). We observe the elimination of the quadratic term, which introduces the polynomial *po*. We perform the trigonometric substitution and then choose to take the "5/3"). After the figure we will explain how to solve a trigonometric equation "by hand" in order to understand what the software has done.

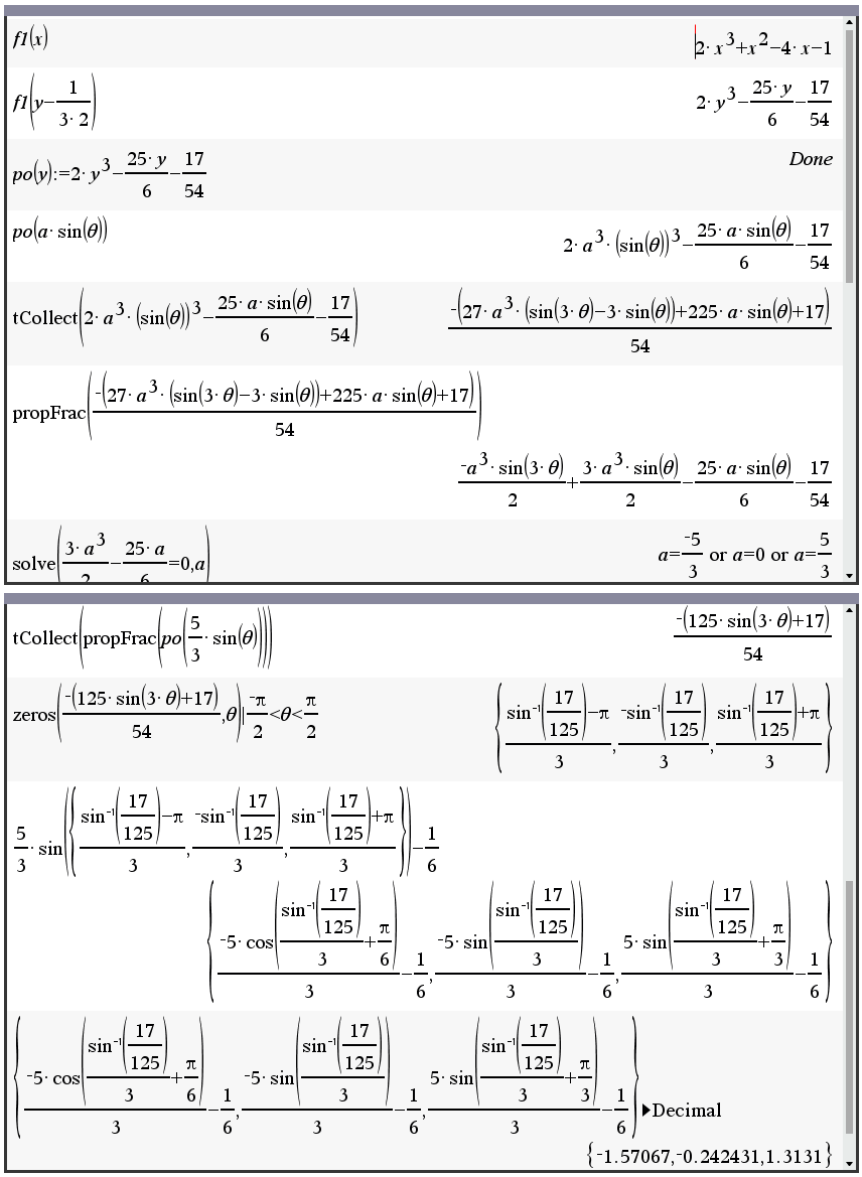


Figure 7

Remark: on the interval $-\pi/2 < \theta < \pi/2$ how to solve the equation $\dfrac{-(125\sin(3\theta)+17)}{54} = 0$?

This is equivalent to solving the equation $\sin(3\theta) = -\dfrac{17}{125}$. Now we know that, for real x and a between –1 and 1, all solutions of the equation $\sin(x) = a$ are given by

$$\arcsin(a) + 2k\pi, \quad \pi - \arcsin(a) + 2k\pi \quad (k \in \mathbb{Z}\ .$$

The arcsine function being odd, all solutions of the equation $\sin(3\theta) = -\dfrac{17}{125}$ are given by $\dfrac{-\sin^{-1}\left(\dfrac{17}{125}\right) + 2k\pi}{3}$ and $\dfrac{\pi + \sin^{-1}\left(\dfrac{17}{125}\right) + 2k\pi}{3}$. We must not forget that $-\pi/2 < \theta < \pi/2$. This keeps the following three values: $-\dfrac{1}{3}\sin^{-1}\left(\dfrac{17}{125}\right)$, $\dfrac{\pi + \sin^{-1}\left(\dfrac{17}{125}\right)}{3}$ and $\dfrac{\sin^{-1}\left(\dfrac{17}{125}\right) - \pi}{3}$.

Again, the DERIVE solution:

#19: Problems 2d)

#20: $f1(x) := 2 \cdot x^3 + x^2 - 4 \cdot x - 1$

#21: $\text{EXPAND}\left(f1\left(y - \frac{1}{3 \cdot 2}\right)\right) = 2 \cdot y^3 - \frac{25 \cdot y}{6} - \frac{17}{54}$

#22: $po(y) := 2 \cdot y^3 - \frac{25 \cdot y}{6} - \frac{17}{54}$

#23: $po(a \cdot \text{SIN}(\theta)) = 2 \cdot a^3 \cdot \text{SIN}(\theta)^3 - \frac{25 \cdot a \cdot \text{SIN}(\theta)}{6} - \frac{17}{54}$

#24: Trigonometry := Collect

#25: $po(a \cdot \text{SIN}(\theta)) = -\frac{a^3 \cdot \text{SIN}(3 \cdot \theta)}{2} + \left(\frac{3 \cdot a^3}{2} - \frac{25 \cdot a}{6}\right) \cdot \text{SIN}(\theta) - \frac{17}{54}$

#26: $\text{SOLVE}\left(\frac{3 \cdot a^3}{2} - \frac{25 \cdot a}{6}, a\right) = \left(a = -\frac{5}{3} \vee a = \frac{5}{3} \vee a = 0\right)$

#27: $po\left(\frac{5}{3} \cdot \text{SIN}(\theta)\right) = -\frac{125 \cdot \text{SIN}(3 \cdot \theta)}{54} - \frac{17}{54}$

#28: $\text{SOLUTIONS}\left(-\frac{125 \cdot \text{SIN}(3 \cdot \theta)}{54} - \frac{17}{54} = 0, \theta\right)$

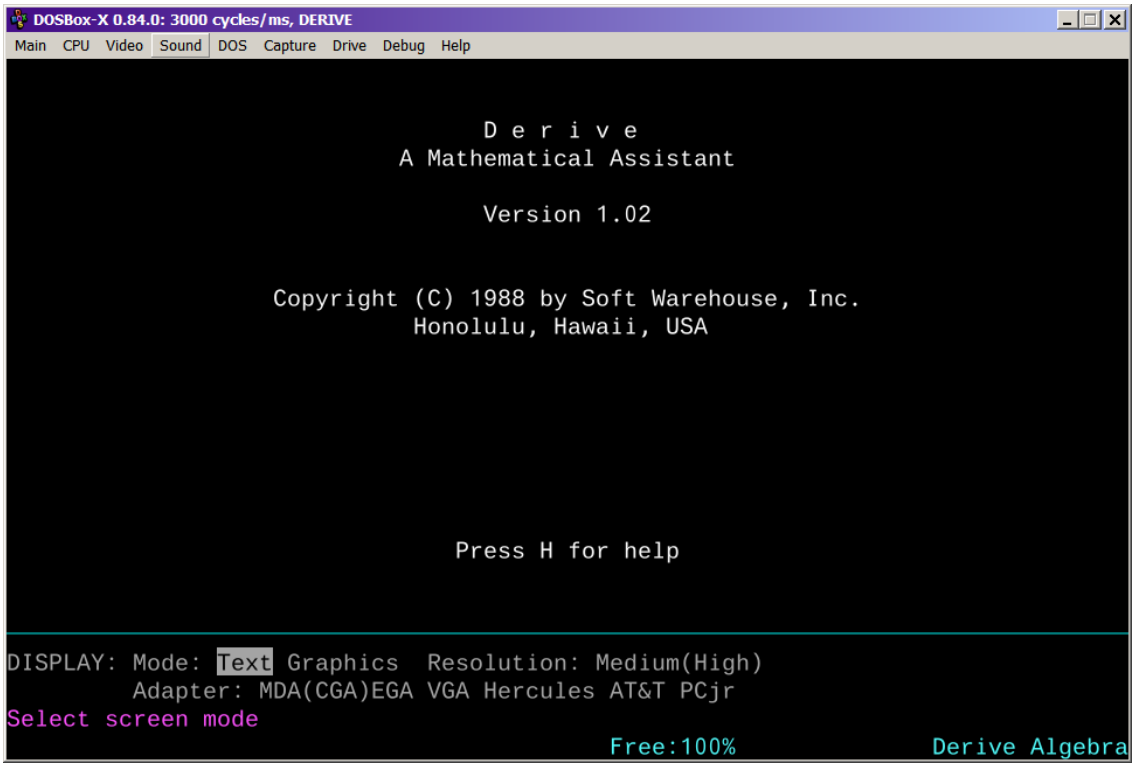
#29: $\left[\frac{\text{ATAN}\left(\frac{17 \cdot \sqrt{426}}{2556}\right)}{3} + \frac{\pi}{3}, \frac{\text{ATAN}\left(\frac{17 \cdot \sqrt{426}}{2556}\right)}{3} - \frac{\pi}{3}, -\frac{\text{ATAN}\left(\frac{17 \cdot \sqrt{426}}{2556}\right)}{3}\right]$

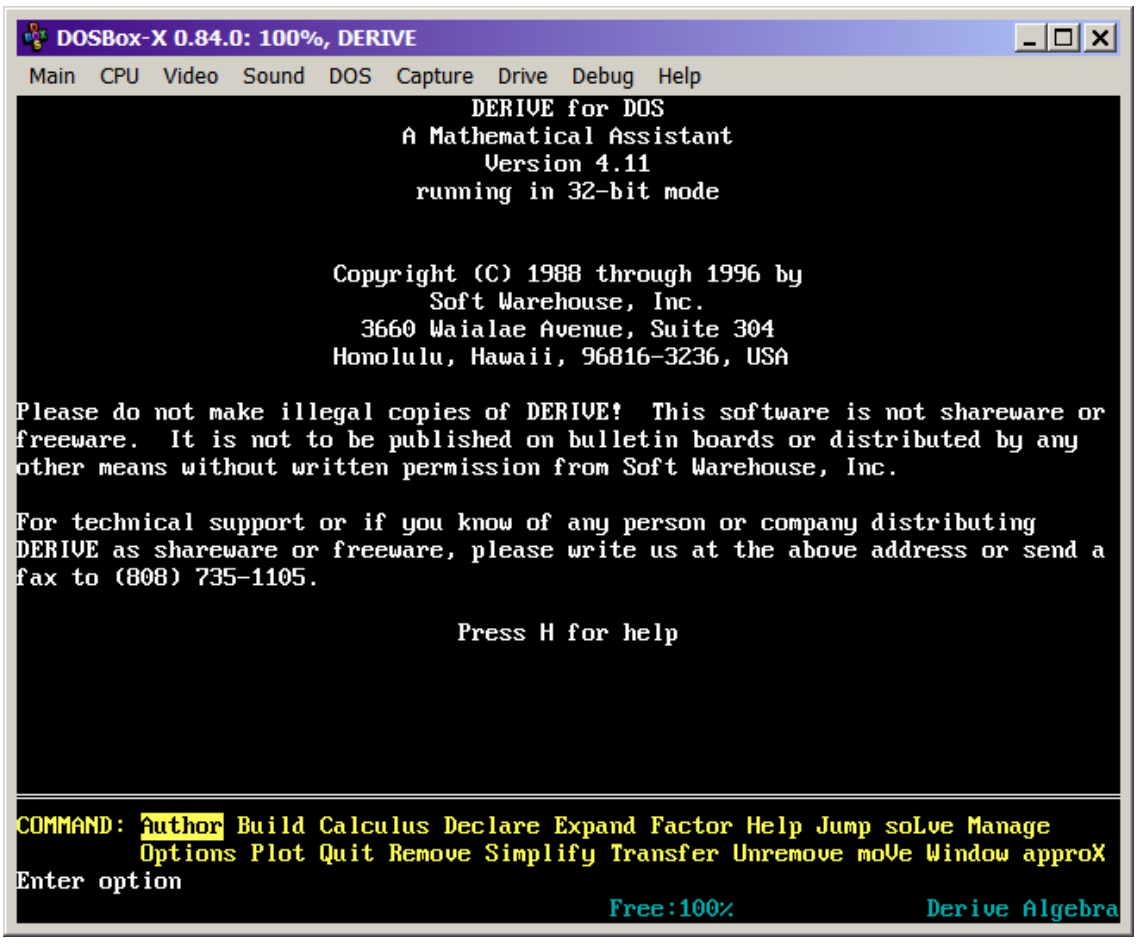
#30: $\text{VECTOR}\left(\frac{5}{3} \cdot \text{SIN}(t) - \frac{1}{6}, t, \left[\frac{\text{ATAN}\left(\frac{17 \cdot \sqrt{426}}{2556}\right)}{3} + \frac{\pi}{3}, \frac{\text{ATAN}\left(\frac{17 \cdot \sqrt{426}}{2556}\right)}{3} - \frac{\pi}{3}, -\frac{\text{ATAN}\left(\frac{17 \cdot \sqrt{426}}{2556}\right)}{3}\right]\right)$

#31: $\left[5 \cdot \text{SIN}\left(\frac{\text{ATAN}\left(\frac{17 \cdot \sqrt{426}}{2556}\right)}{3} + \frac{\pi}{3}\right) - \frac{1}{6}, -\frac{5 \cdot \text{COS}\left(\frac{\text{ACOT}\left(-\frac{17 \cdot \sqrt{426}}{2556}\right)}{3}\right)}{3} - \frac{1}{6}, -\frac{5 \cdot \text{SIN}\left(\frac{\text{ATAN}\left(\frac{17 \cdot \sqrt{426}}{2556}\right)}{3}\right)}{3} - \frac{1}{6}\right]$

#32: [1.313099034, −1.570668057, −0.2424309764]

Dr. Hepperle provided a time list of most of the DERIVE for DOS versions (will possibly appear in next DNL). Here are screen shots of his first and his last DERIVE version for DOS





Polygons – Part 2

Felipe de Jesus Martinez Vargas, Mexico

Notice: θ is angle of anticlockwise rotation in all functions.
θ = 0 by default, i.e. θ must only be entered as last parameter if it differs from 0.

• Function Triangle

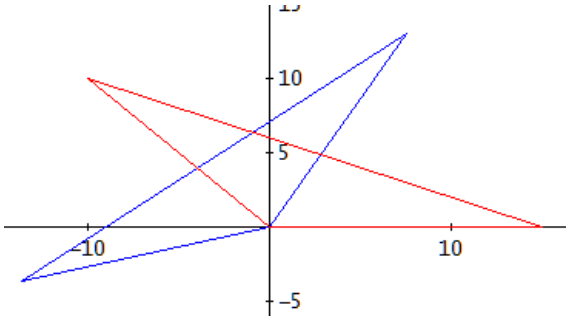
Generates triangles.

Syntax: Triangle(b,h,a,θ)
b: x-base, h: y-displacement of upper vertex, a: x-displacement of upper vertex
The triangle is defined by the points [(0,0), (b,0),(a,h)]; rotation center is origin.

$$\text{triangle}(b,\ h,\ a,\ \theta := 0) := \begin{bmatrix} 0 & 0 \\ b & 0 \\ a & h \\ 0 & 0 \end{bmatrix} \cdot \text{Rot}(\theta)$$

triangle(15, 10, −10)

$$\text{triangle}\left(15,\ 10,\ -10,\ \frac{\pi}{3}\right)$$



• Function rTriangle

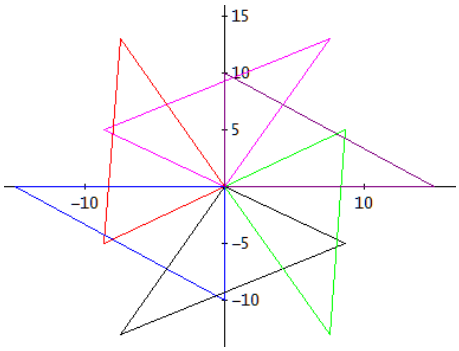
Generates right triangles.

Syntax: rTriangle(b,h,θ)
b: x-base, h: y-displacement of the upper vertex
Right angle is in the origin; triangle is defined by [(0,0), (b,0), (0,a)].

$$\text{rTriangle}(b,\ h,\ \theta := 0) := \begin{bmatrix} 0 & 0 \\ b & 0 \\ 0 & h \\ 0 & 0 \end{bmatrix} \cdot \text{Rot}(\theta)$$

rTriangle(15, 10)

$$\text{VECTOR}\left(\text{rTriangle}(15,\ 10,\ \theta),\ \theta,\ 0,\ 2\cdot\pi,\ \frac{\pi}{3}\right)$$



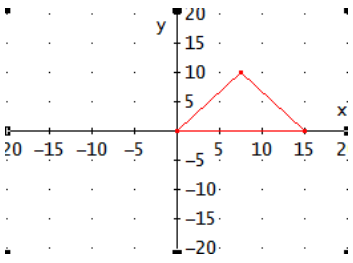
• Function iTriangle

Generates isosceles triangles.

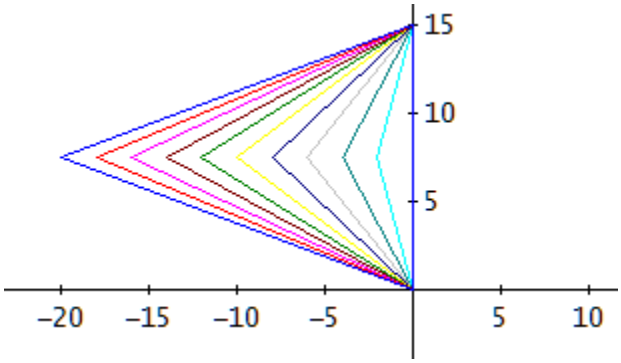
Syntax: iTriangle(b,h,θ)
b: base length, h: y-displacement of the upper vertex (altitude)
Left vertex of the bease is the origin; triangle is defined by [(0,0), (b,0), (b/2,h)].

$$\text{iTriangle}(b,\ h,\ \theta := 0) := \begin{bmatrix} 0 & 0 \\ b & 0 \\ \frac{b}{2} & h \\ 0 & 0 \end{bmatrix} \cdot \text{Rot}(\theta)$$

iTriangle(15, 10)



$$\text{VECTOR}\left(\text{iTriangle}\left(15,\ H,\ \frac{\pi}{2}\right),\ H,\ 0,\ 20,\ 2\right)$$



• Function **eTriangle**

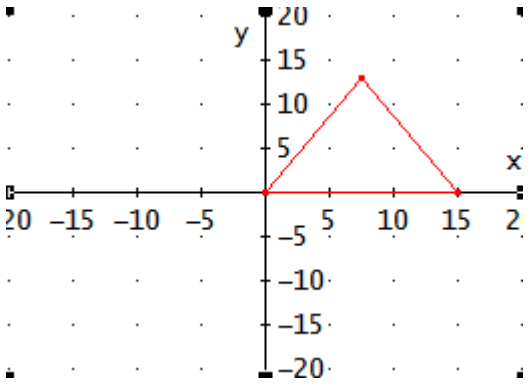
Generates equilateral triangles.

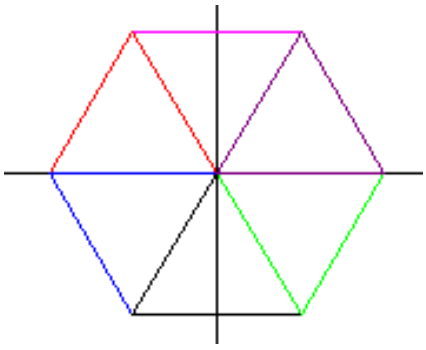
Syntax: eTriangle(a,θ)
a: side length
Left base vertex is the origin; base on the x-axis

$$\text{eTriangle}(a,\ \theta := 0) := \begin{bmatrix} 0 & 0 \\ a & 0 \\ \frac{a}{2} & \frac{\sqrt{3}}{2}\cdot a \\ 0 & 0 \end{bmatrix}\cdot\text{Rot}(\theta)$$

eTriangle(15)

$$\text{VECTOR}\left(\text{eTriangle}(15,\ \theta),\ \theta,\ 0,\ 2\cdot\pi,\ \frac{\pi}{3}\right)$$



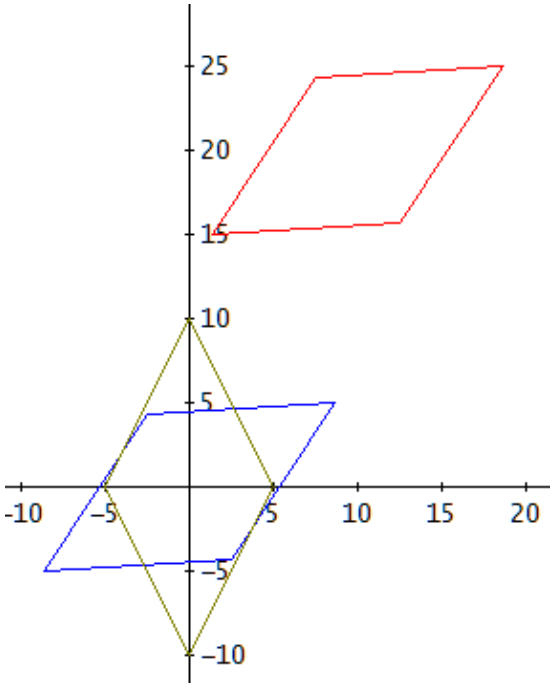


• Function **Rhombus**

Generates a rhombus.

Syntax: Rhombus(p, q, θ)
p: horizontal diagonal; q: vertical diagonal
Center of the rhombus is the origin.

$$\text{Rhombus}(p, q, \theta := 0) := \begin{bmatrix} \frac{p}{2} & 0 \\ 0 & \frac{q}{2} \\ -\frac{p}{2} & 0 \\ 0 & -\frac{q}{2} \\ \frac{p}{2} & 0 \end{bmatrix} \cdot \text{Rot}(\theta)$$



Rhombus(10, 20)

$$\text{Rhombus}\left(10, 20, \frac{2\cdot\pi}{3}\right)$$

$$\text{Translate}\left(\text{Rhombus}\left(10, 20, \frac{2\cdot\pi}{3}\right), 10, 20\right)$$

• Function **Square**

Generates squares.

Syntax: Square(a, θ)
a: side

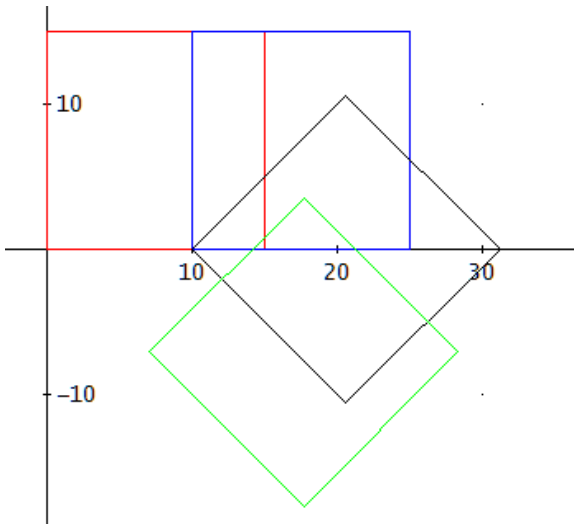
$$\text{Square}(a, \theta := 0) := \begin{bmatrix} 0 & 0 \\ a & 0 \\ a & a \\ 0 & a \\ 0 & 0 \end{bmatrix} \cdot \text{Rot}(\theta)$$

Square(15)

Translate(Square(15), 10, 0)

$$\text{Translate}\left(\text{Square}\left(15, -\frac{\pi}{4}\right), 10, 0\right)$$

$$\text{Rotate}\left(\text{Translate}(\text{Square}(15), 10, 0), -\frac{\pi}{4}\right)$$



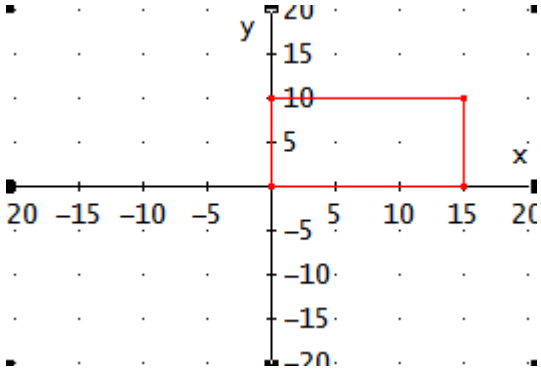
• Function **Rectangle**

Generates rectangles.

Syntax: Rectangle(b,h,θ)
b: horizontal base, h: altitude
Left vertex is in the origin.

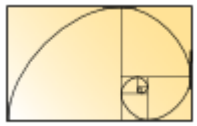
$$\text{Rectangle}(b, h, \theta := 0) := \begin{bmatrix} 0 & 0 \\ b & 0 \\ b & h \\ 0 & h \\ 0 & 0 \end{bmatrix} \cdot \text{Rot}(\theta)$$

Rectangle(15, 10, 0)



• Function goldenRect

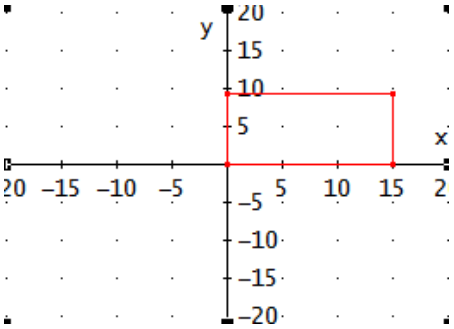
Generates "Golden rectangles"; sides in golden section ratio



Syntax: goldenRect(b,θ)
b: base (longer side horizontal with left vertex in the origin)

```
goldenRect(b, θ := 0, δ, h) :=
  Prog
    δ := (1 + √5)/2
    h := b/δ
    [0, 0; b, 0; b, h; 0, h; 0, 0]·Rot(θ)

goldenRect(15)
```



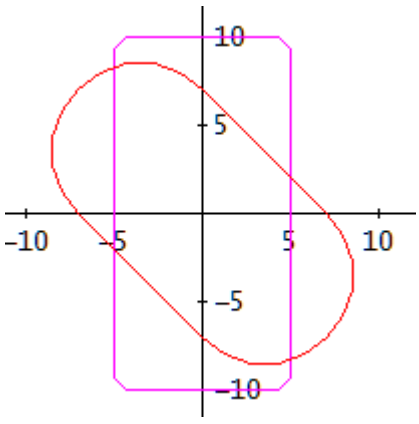
Next one is more interesting:

• Function roundedRect

Generates rectangles with rounded vertices

Syntax: roundedRect(h, b, r, θ)
h: outside vertical length; b: outside horizontal length;
r: radius of the corner

$$\left[\text{roundedRect}(20, 10, 1),\ \text{roundedRect}\left(20, 10, 5, \frac{\pi}{4}\right)\right]$$

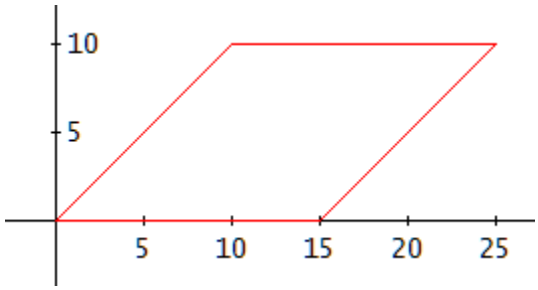


• Function Parallelogram

Generates parallelograms.

Syntax:Parallelogramm(b,a,h)
b: base side (horizontal starting in origin);
a: x-displacement of the upper side; h: altitude of the p.

$$\text{Parallelogram}(b,\ a,\ h,\ \theta := 0) := \begin{bmatrix} 0 & 0 \\ b & 0 \\ b + a & h \\ a & h \\ 0 & 0 \end{bmatrix} \cdot \text{Rot}(\theta)$$



Parallelogram(15, 10, 10)

• Function **Trapezoid**

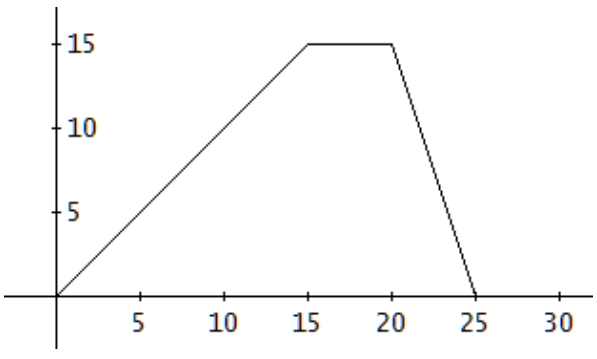
Generates trapezoids.

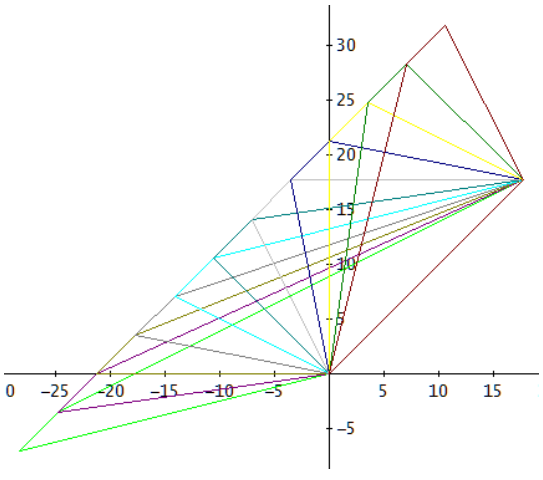
Syntax: Trapezoid(a,b,c,h,θ)
a: top side; b: base side;
c: x-displacement of the top side (= x-coordinate of endpoint of left leg);
h: altitude

$$\text{Trapezoid}(a,\ b,\ c,\ h,\ \theta := 0) := \begin{bmatrix} 0 & 0 \\ b & 0 \\ c + a & h \\ c & h \\ 0 & 0 \end{bmatrix} \cdot \text{Rot}(\theta)$$

Trapezoid(5, 25, 15, 15)

$$\text{VECTOR}\left(\text{Trapezoid}\left(5,\ 25,\ c,\ 15,\ \frac{\pi}{4}\right),\ c,\ -25,\ 25,\ 5\right)$$





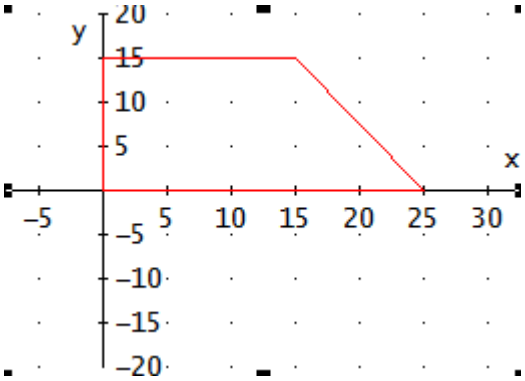
• Function **rTrapezoid**

Generates right trapezoids.

Syntax: rTrapezoid(a,b,h,θ)
a: top side; b: base side; h: altitude
The right angle is in the origin and the trapezoids is directed to the right

$$rTrapezoid(a, b, h, \theta := 0) := \begin{bmatrix} 0 & 0 \\ b & 0 \\ a & h \\ 0 & h \\ 0 & 0 \end{bmatrix} \cdot Rot(\theta)$$

rTrapezoid(15, 25, 15, 0)

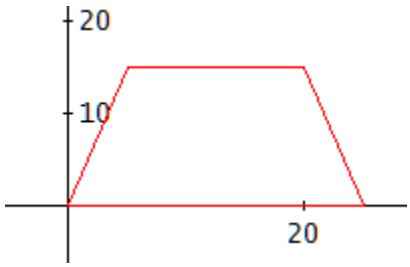


• Function iTrapezoid

Generates isosceles trapezoids.

Sytax: iTrapezoid(a,b,h,θ)
a: top side; b: base side; h: altitude
Left bottom edge in the origin.

$$iTrapezoid(a, b, h, \theta := 0) := \begin{bmatrix} 0 & 0 \\ b & 0 \\ a + \frac{b - a}{2} & h \\ \frac{b - a}{2} & h \\ 0 & 0 \end{bmatrix} \cdot Rot(\theta)$$

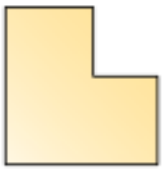


iTrapezoid(15, 25, 15)

• Function L

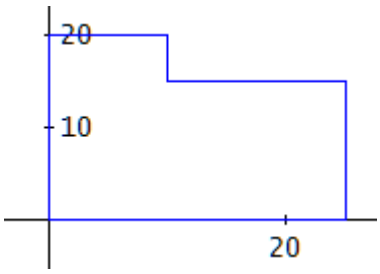
Generates polygons in form of an L.

Syntax: L(a,b,c,h,θ)
a: top side; b: base side; c: altitude of upper step
h: total altitude



$$L(a, b, c, h, \theta := 0) := \begin{bmatrix} 0 & 0 \\ b & 0 \\ b & h - c \\ a & h - c \\ a & h \\ 0 & h \\ 0 & 0 \end{bmatrix} \cdot Rot(\theta)$$

L(10, 25, 5, 20)



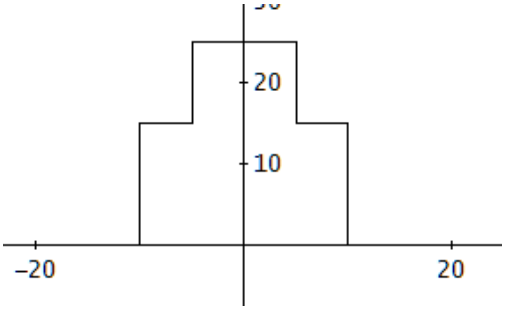
• Function T

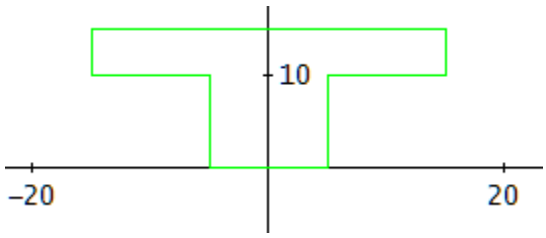
Generates polygons in form of a T.

Syntax: T(a,b,c,h,θ)
a: top side; b: base side; c: upper altitode; h: total altitude
T is symmetric wrt y-axis with abse on the x-axis.

```
T(a, b, c, h, θ := 0, out_) :=
  Prog
    out_ := [a/2, h; a/2, h - c; b/2, h - c; b/2, 0]
    out_ := APPEND(out_, REVERSE(out_·[-1, 0; 0, 1]))
    out_ := APPEND(out_, [FIRST(out_)])
    out_ := out_·Rot(θ)
    out_
```

```
[T(10, 20, 10, 25), T(30, 10, 5, 15)]
```





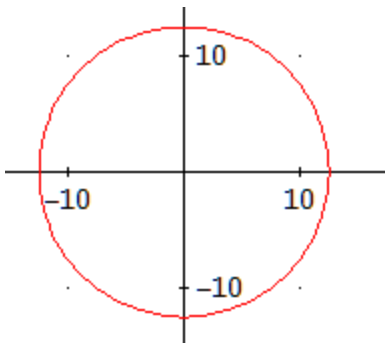
• Function circle

Generates circles with center in origin.
(needs function pgon)

Syntax: circle(d)
d: diameter

$$\text{circle}(D) := \text{pgon}\left(100, \frac{D}{2}, 0\right)$$

```
circle(25)
```



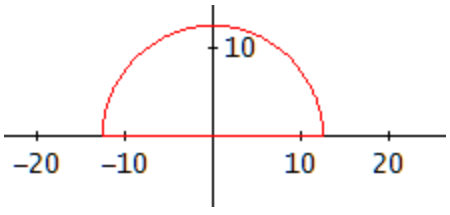
• Function semicircle

Generates semicircles & diameter.

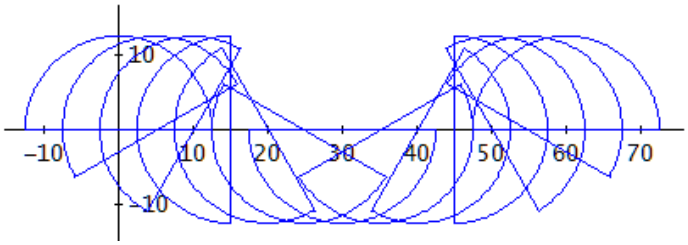
Syntax: semicircle(d,θ)
d: diameter of the semicircle

```
semicircle(d, θ := 0, out_, pts_) :=
  Prog
    pts_ := 100
    out_ := VECTOR([d/2·COS(π·k/pts_), d/2·SIN(π·k/pts_)], k, 0, pts_)
    out_ := APPEND(out_, [FIRST(out_)])
    out_ := out_·Rot(θ)
    out_
```

```
semicircle(25, 0)
```



$$\text{VECTOR}\left(\text{Translate}\left(\text{semicircle}\left(25,\ \frac{k\cdot\pi}{6}\right),\ 5\cdot k,\ 0\right),\ k,\ 0,\ 12\right)$$



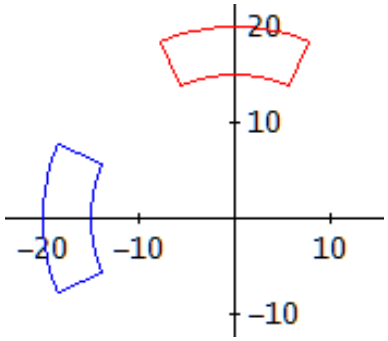
• Function **arch**

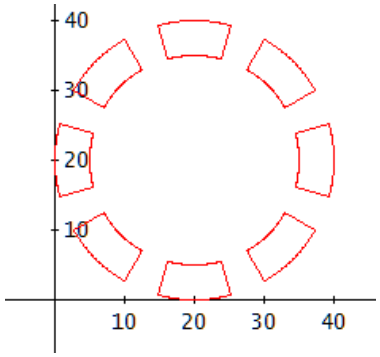
Generates arches of a circular ring.

Syntax: arch(re,ri, ω,θ)
re: outer radius; ri: inner radius; ω: aperture angle

$$\left[\text{arch}(20,\ 15,\ 45°),\ \text{arch}\left(20,\ 15,\ 45°,\ \frac{\pi}{2}\right)\right]$$

$$\text{VECTOR}\left(\text{Translate}\left(\text{arch}\left(20,\ 15,\ 30°,\ \frac{\pi\cdot k}{4}\right),\ 20,\ 20\right),\ k,\ 0,\ 7\right)$$





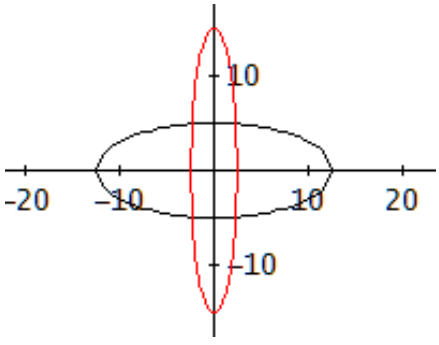
• Function **ellipse**

Generates ellipses (center in origin, axes || x- and y-axis)

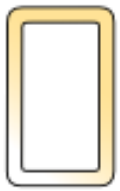
Syntax: ellipse(p,q,θ)
p: horizontal axis; q: vertical axis

```
ellipse(p, q, θ := 0, pts_, k, a_, b_, out_) :=
  Prog
    [a_ := p/2, b_ := q/2, out_ := [], pts_ := 40, k := 0]
    Loop
      out_ := INSERT([p/pts_·k - a_, b_·√(1 - ((p/pts_·k - a_)/a_)^2)], out_, k + 1)
      If k = pts_
         exit
         k :+ 1
    out_ := APPEND(out_, REVERSE(out_·[1, 0; 0, -1]))
    out_·Rot(θ)

[ellipse(25, 10), ellipse(5, 30)]
```



• Function HSS_beam



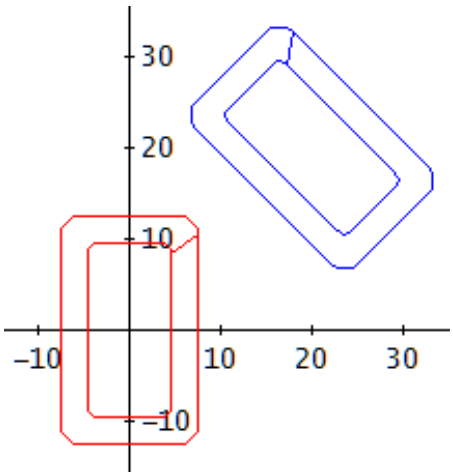
Generates polygons of square and rectangular cross-section tubes.

Syntax: HSS_beam(h,b,te,re,ri,θ)
h: outside vertical length; b: outside horizontal length;
te: wall thickness; re: outside corner radius (typical = 2te);
ri: inner corner radius (typical = te)

```
HSS_beam(h, b, te, re, ri, θ := 0, out_) :=
  Prog
    out_ := APPEND(roundedRect(h, b, re, 0), REVERSE(roundedRect(h - 2·te, b - 2·te, ri, 0)))
    out_ := APPEND(out_, [FIRST(out_)])·Rot(θ)
    out_
```

```
HSS_beam(25, 15, 3, 2, 1)
```

$$\text{Translate}\left(\text{HSS\_beam}\left(25, 15, 3, 2, 1, \frac{\pi}{4}\right), 20, 20\right)$$



• Function CHS_beam

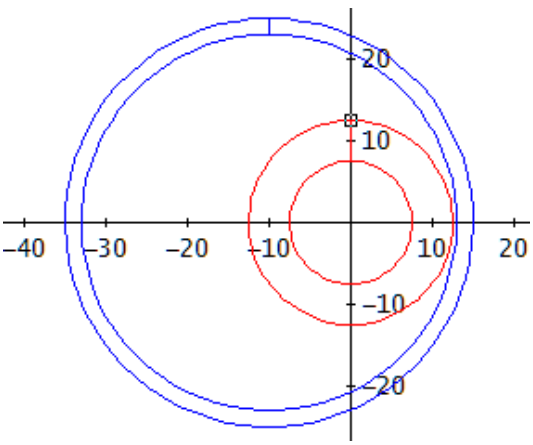
Generates cross sections of circular pipes as polygons.

Syntax: CHS_beam(de,te)
de: outer diameter; te: wall width

$$\text{CHS\_beam}(de, te) := \text{hPgon}\left(75, \frac{de}{2}, te, 0\right)$$

```
CHS_beam(25, 5)

Translate(CHS_beam(50, 2), -10, 0)
```



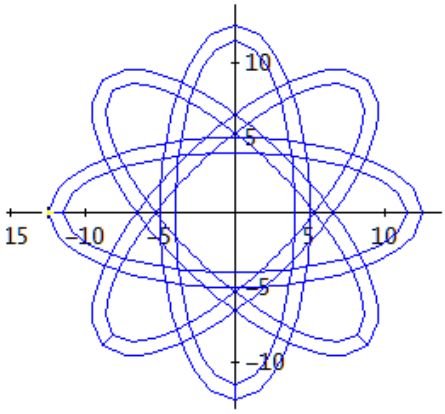
• Function EHS_beam

Generates cross sections of elliptical pipes.

Syntax: EHS_beam(p,q,te,θ)
p: horizontal width of outside ellipse;
q: vertical width of outside ellipse,
te: wall width

```
EHS_beam(p, q, te, θ := 0, out_) :=
  Prog
    out_ := APPEND(ellipse(p, q, 0), REVERSE(ellipse(p - 2·te, q - 2·te, 0)))
    out_ := APPEND(out_, [FIRST(out_)])·Rot(θ)
    out_
```

$$\text{VECTOR}\left(\text{EHS\_beam}\left(25,\ 10,\ 1,\ \frac{k\cdot\pi}{4}\right),\ k,\ 0,\ 3\right)$$



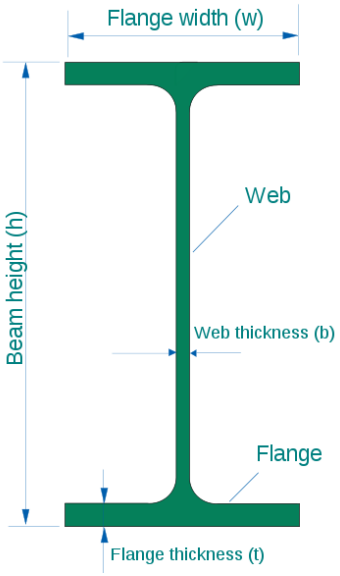
• Function I_beam

Generates European I-profile polygons (**EU**: IPE / HE / HL / HD / HP;
UK: UB / UC / UBP; **US**: W / HP; **RU**: HG; **JP**: H).

Syntax: I_beam(h,b,tw,tf,r,θ)
h: beam height; b: flange width;
tw: web thickness; tf: flange thickness;
r: radius of the web-flange connection

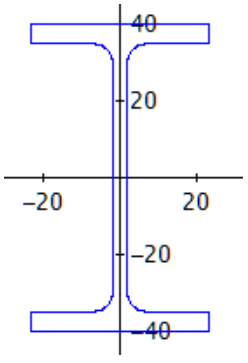
Note: r=0 gives an I-beam with straight edges

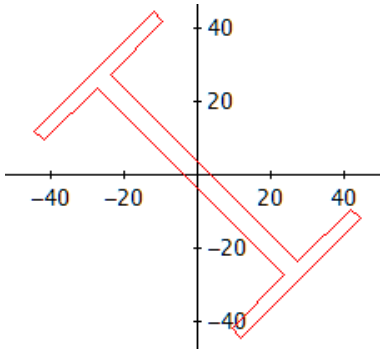
https://commons.wikimedia.org/w/index.php?curid=10305422



I_beam(80, 46, 3.8, 5.2, 5, 0)

$$\text{I\_beam}\left(80,\ 46,\ 5.2,\ 3.8,\ 0,\ \frac{\pi}{4}\right)$$

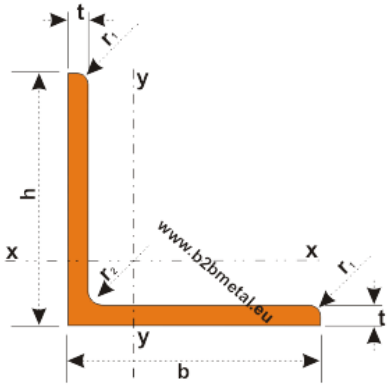




• Function L_beam

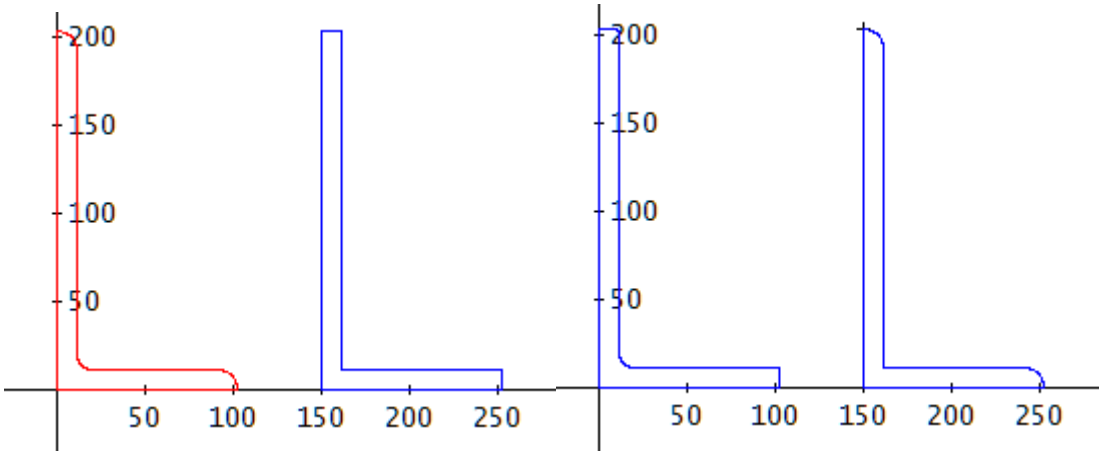
Generates poligons in form of L-shaped angle beams.

Syntax: l_beam(h,b,te,r1,r2,θ)
h: vertical leg; b: horizontal leg;
te: thickness of the legs;
r1: radius at the end of the legs
r2: radius of the connection of the legs



```
[L_beam(203, 102, 11.1, 10, 10), Translate(L_beam(203, 102, 11.1, 0, 0), 150, 0)]
```

```
[L_beam(203, 102, 11.1, 1, 10), Translate(L_beam(203, 102, 11.1, 10, 1), 150, 0)]
```

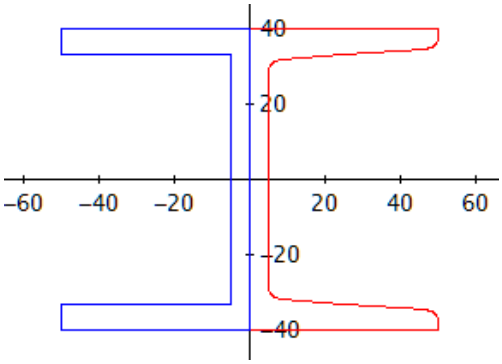


- Function **U_beam**

Generates poligons in form of U-shaped beams.
EU [DIN 1026-1: 2000] (UPN) or parallel flanged channels (UPE / PFC).

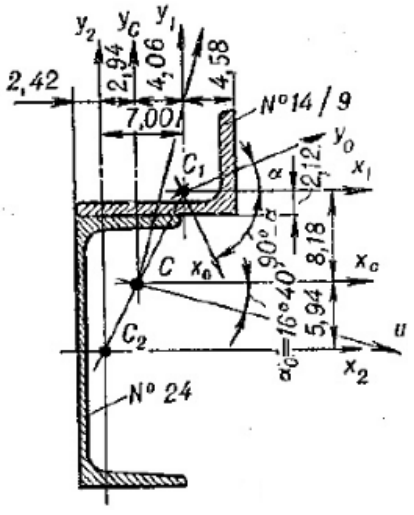
Syntax: U_beam(h,b,tw,tf,r1,r2,θ)
h: height of the cant [mm]; b: length of the legs;
tw: cant thickness; tf: legs' thickness;
r1: radius at the end of the legs (*UPE / PFC when r1 = 0*);
r2: radius of the interior edges;

```
[U_beam(80, 50, 5, 7, 3, 3), U_beam(80, 50, 5, 7, 0, 0, π)]
```



Ejemplo/Example:

Representar las secciones transversales compuesta de un angular de alas desiguales N° 14/90 (GOST 8510-72) y un canal N° 24 (GOST 8240-72).
Represent the composite cross sections of an angular with unequal wings N° 14/90 (GOST 8510-72) and a channel N° 24 (GOST 8240-72) .



Konstruktionszeichnung

- ANGULAR N° 14/90 (GOST 8510-72)

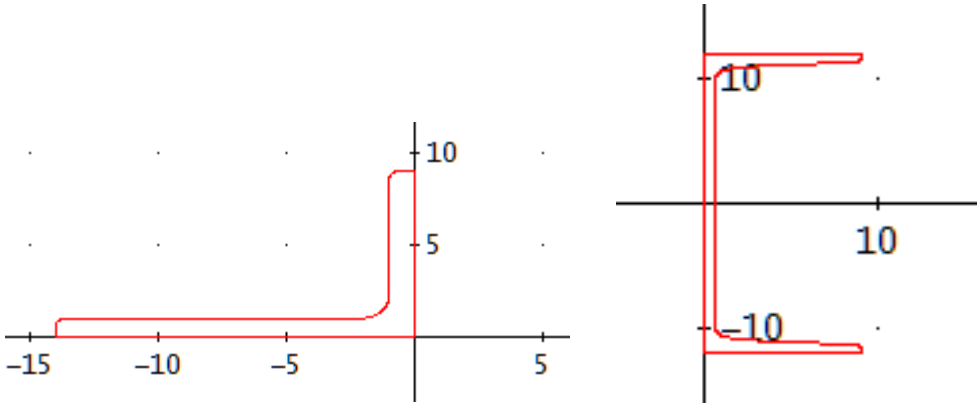
#67: polygon1 := L_beam(140, 90, 10, 12, 4, 90°)·mm_

#68: $\text{polygon1}\cdot\frac{1}{\text{cm\_}}$

- CANAL N° 24 (GOST 8240-72)

#69: polygon2 := U_beam(240, 90, 5.6, 10, 10.5, 4, 0)·mm_

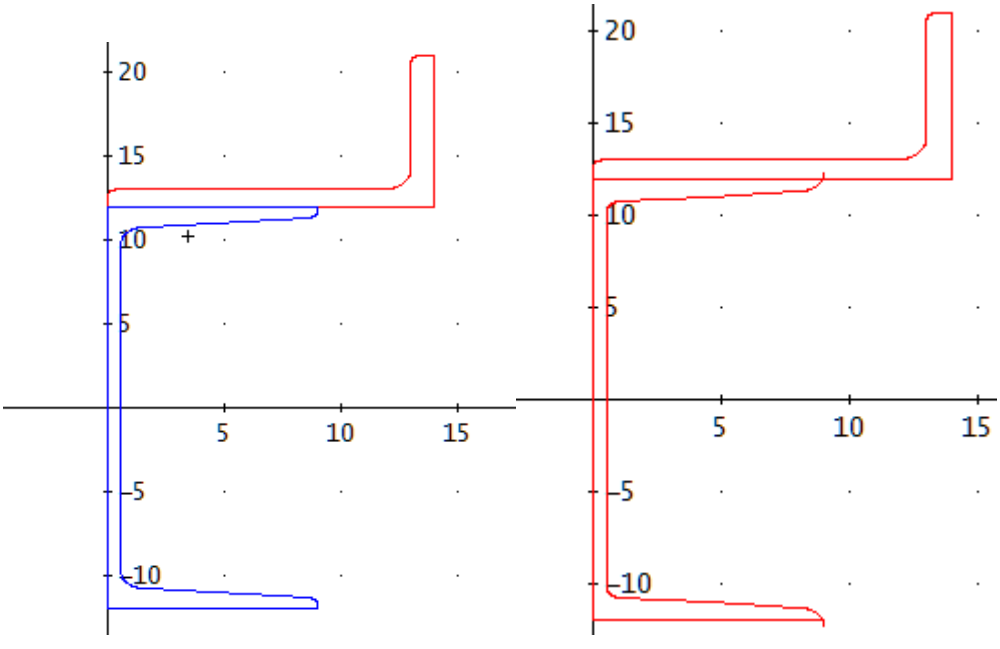
#70: $\text{polygon2}\cdot\frac{1}{\text{cm\_}}$



- SECCIÓN TRANSVERSAL COMPUESTA
- COMPOSITE CROSS SECTION

$$\left[\text{Translate}\left(\text{polygon1}\cdot\frac{1}{\text{cm\_}},\ 14,\ 12\right),\ \text{polygon2}\cdot\frac{1}{\text{cm\_}}\right]$$

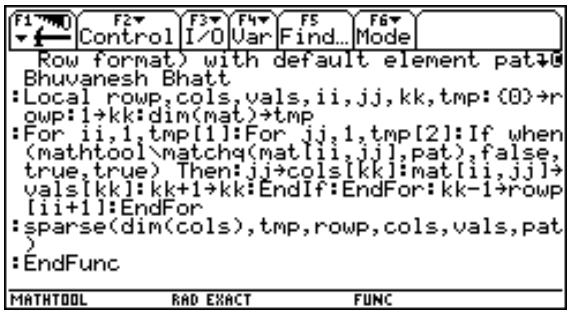
$$\text{APPEND}\left(\text{polygon2}\cdot\frac{1}{\text{cm\_}},\ \text{REVERSE}\left(\text{Translate}\left(\text{polygon1}\cdot\frac{1}{\text{cm\_}},\ 14,\ 12\right)\right)\right)$$



Sparse Matrices (Dünn besetzte Matrizen)

Josef Böhm, inspired by Bhuvanesh Bhatt (and many web sites)

When I worked through Bhuvanesh's *Mathtools* (see earlier DNLs) I came across three functions addressing "*Sparse Matrices*".



I must admit that I didn't have any idea about their purpose.

Explication from the web:

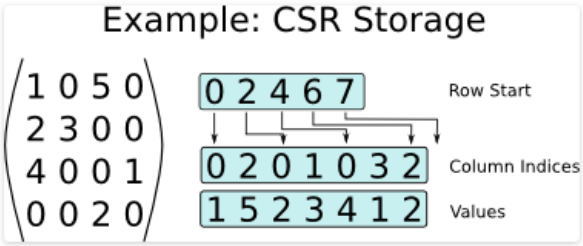
A sparse matrix is a special case of a matrix in which the number of zero elements is much higher than the number of non-zero elements. As a rule of thumb, if 2/3 of the total elements in a matrix are zeros, it can be called a sparse matrix. Using a sparse matrix representation — where only the non-zero values are stored — the space used for representing data and the time for scanning the matrix are reduced significantly.

The compressed sparse row (CSR) or compressed row storage (CRS) or Yale format **represents a matrix M by three (one-dimensional) arrays, that respectively contain nonzero values, the extents of rows, and column indices**. It is similar to COO, but compresses the row indices, hence the name.

So, I tried the first two TI-92-functions referring to an example from one of the many web resources:

CSR format

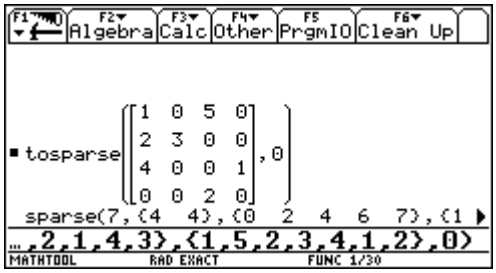
In the CSR format all column indices for each nonzero are stored row after row in a single buffer. Similarly, all nonzero values are stored row after row in a single buffer. The entry points denoting the beginning of each row are stored in a third buffer, where the end of the `i`-th row is implicitly given by the start of the `i+1`-th row.



Example of storing a sparse matrix with 0-based indexing in the CSR format.

Here is a short explication of the CSR-format. You can see that the matrix is described by three arrays: {0,2,4,6,7}, {0,2,0,1,0,3,2}, {1,5,2,3,4,1,2}. Learn more on page 30.

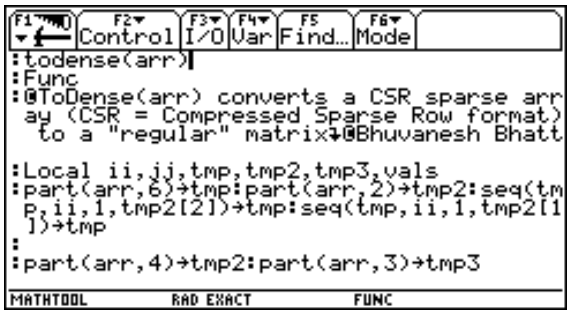
Let's see how `tosparse(mat,pat)` performs.(`pat` is the element filling the "empty" cells of the matrix, which is usually 0).

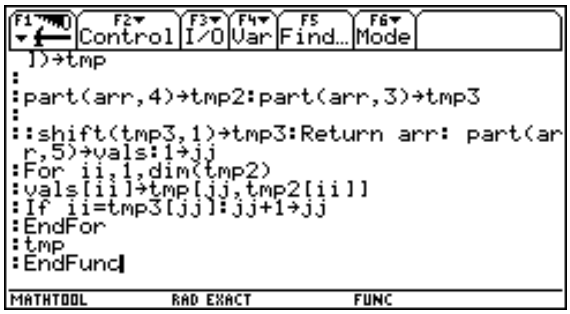


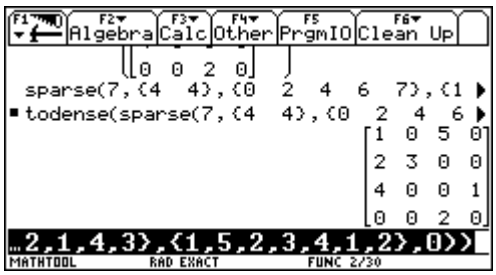
The complete output reads as follows:

`sparse(7,{4,4},{0,2,4,6,7},{1,3,1,2,1,4,3},{1,5,2,3,4,1,2},0).`

We have 7 elements ≠ pat distributed in a 4×4-matrix. RowStart-array and Values-array are the same as given in the example from above. ColumnIndices-array is shifted by 1. This is because Bhuvanesh starts column indexing with 1. I will explain how to interpret the output later. Let's first try the reverse action: how to come back to the original matrix from its CSR-notation:







Seemed to work! I was happy and "translated" both functions into DERIVE- and TI-Nspire-Code.

$$tosparse\left(\begin{bmatrix}2&0&1\\0&0&0\end{bmatrix},0\right) \qquad sparse(2,\{2,3\},\{0,2,2\},\{1,3\},\{2,1\},0)$$

$$todense(sparse(2,\{2,3\},\{0,2,2\},\{1,3\},\{2,1\},0)) \qquad \begin{bmatrix}2&0&1\\0&0&0\end{bmatrix}$$

$$tosparse\left(\begin{bmatrix}1&x&5&x\\2&3&x&x\\4&x&x&1\\x&x&2&x\end{bmatrix},x\right) \qquad sparse(7,\{4,4\},\{0,2,4,6,7\},\{1,3,1,2,1,4,3\},\{1,5,2,3,4,1,2\},x)$$

$$todense(sparse(7,\{4,4\},\{0,2,4,6,7\},\{1,3,1,2,1,4,3\},\{1,5,2,3,4,1,2\},x)) \qquad \begin{bmatrix}1&x&5&x\\2&3&x&x\\4&x&x&1\\x&x&2&x\end{bmatrix}$$

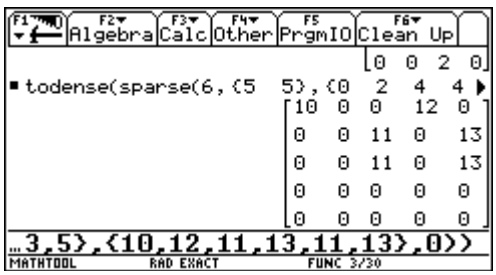
$$tosparse\left(\begin{bmatrix}10&0&0&12&0\\0&0&11&0&13\\0&0&0&0&0\\0&0&11&0&13\\0&0&0&0&0\end{bmatrix},0\right)$$

$$sparse(6,\{5,5\},\{0,2,4,4,6,6\},\{1,4,3,5,3,5\},\{10,12,11,13,11,13\},0)$$

$$todense(sparse(6,\{5,5\},\{0,2,4,4,6,6\},\{1,4,3,5,3,5\},\{10,12,11,13,11,13\},0))$$

$$\begin{bmatrix}10&0&0&12&0\\0&0&11&0&13\\0&0&11&0&13\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix}$$

That was the disappointment and bad surprise. I thought that I had made a mistake in the TI-Nspire-coding and tried the same matrix with Bhuvanesh's tool on the Voyage 200.



Wrong again! Same happened in DERIVE. I found out, that this function works for matrices containing "zero rows" in one direction (`tosparse`), but does not work in reverse direction (`todense`).

I revised my DERIVE code and tried to improve the function. Since there are some other formats to record sparse matrices, I call the functions as `to_csr` and `todense_csr`.

This is my reference matrix from above and its treatment:

```
to_csr [1 0 5 0]
       [2 3 0 0]
       [4 0 0 1]
       [0 0 2 0]

[[4, 4], [0, 2, 4, 6, 7], [0, 2, 0, 1, 0, 3, 2], [1, 5, 2, 3, 4, 1, 2], 0]

todense_csr([4, 4], [0, 2, 4, 6, 7], [0, 2, 0, 1, 0, 3, 2], [1, 5, 2, 3, 4, 1, 2])

[1 0 5 0]
[2 3 0 0]
[4 0 0 1]
[0 0 2 0]
```

Inspecting the input and the output you will find some differences to the TI-results:

- I don't enter the variable `pat` = 0 because it is 0 by default (see next example with another `pat`).
- The number of elements ≠ `pat` does not appear in the output because it is the dimension of the last array.

- Both arrays `rowp` and `cols` start with 0, because almost all references use 0-indexing (Bhuvanesh applies 1-indexing). See also Walter Wegscheider's comment next page.

See now my example applying the improved CSR-function (taking "x" to mark the "empty" cells of the matrix):

```
to_csr([[8, x, 2, x, x],
        [x, x, 5, x, x],
        [x, x, x, x, x],
        [x, x, x, x, x],
        [x, x, 7, 1, 2],
        [x, x, x, x, x],
        [x, x, x, 9, x]], x)
```

```
[[7, 5], [0, 2, 3, 3, 3, 6, 6, 7], [0, 2, 2, 2, 3, 4, 3], [8, 2, 5, 7, 1, 2, 9], x]
```

```
todense_csr([7, 5], [0, 2, 3, 3, 3, 6, 6, 7], [0, 2, 2, 2, 3, 4, 3], [8, 2, 5, 7, 1, 2, 9], x)
```

```
[8, x, 2, x, x]
[x, x, 5, x, x]
[x, x, x, x, x]
[x, x, x, x, x]
[x, x, 7, 1, 2]
[x, x, x, x, x]
[x, x, x, 9, x]
```

I try to explain how to read the output. It might be a bit confusing, because rows and columns have numbers starting with 0 (their indices):
The sparse matrix described by `[[7,5], …,x]` is a 7×5 matrix containing the elements ≠ `pat` (= x) in the last array `[8,2,5,7,1,2,9]`. The first array is the row pointer (`rowp`) and the second one gives the columns (`cols`).

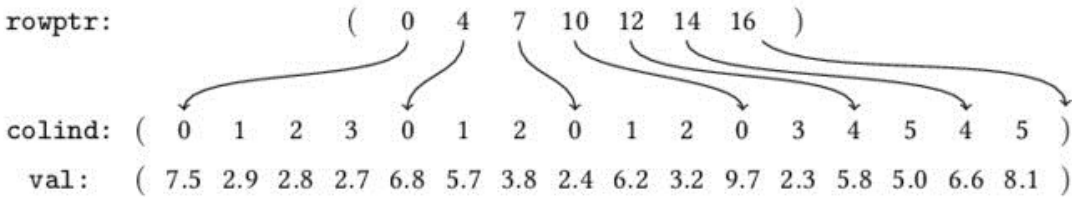
row 1: until 2nd element in `cols` (0,2) which are cols 1 and 3 elements 8 and 2, else x: `[8,x,2,x,x]`
row 2: until 3rd element in `cols` (2), which is col 3 take 5, else x: `[0,0,5,0,0]`
row 3: until 3rd element in `cols` (no new element in this row): `[x,x,x,x,x]`
row 4: again until 3rd element in `cols` (no new element in this row, too): `[x,x,x,x,x]`
row 5: until 6th element in `cols` (2,3,4) which are cols (3,4,5) take the next 3 elements `:[x,x,7,1,2]`
row 6: until 6th element in `cols`, which gives no element in this row: `[x,x,x,x,x]`
row 7: last element (9) is in col 4 (index = 3): `[x,x,x,9,x]`

Now we have reconstructed the matrix. Function `todense_csr` does the job. I must admit that it took me long time and much thinking to find a solution how to improve Bhuvanesh's function. It's a pity but I have to realize that my gray cells are becoming older. This is one of the reasons for the delay of publication of DNL#127.

I am quite sure that my solution of this problem is not the most elegant one. I saw that repeating numbers in the first array indicate empty rows. This helped. It would be great if anybody could provide a tighter code for `todense`!

I found another nice illustration for the CSR-format of a sparse matrix (among the references):

$$A = \begin{pmatrix} 7.5 & 2.9 & 2.8 & 2.7 & 0 & 0 \\ 6.8 & 5.7 & 3.8 & 0 & 0 & 0 \\ 2.4 & 6.2 & 3.2 & 0 & 0 & 0 \\ 9.7 & 0 & 0 & 2.3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5.8 & 5.0 \\ 0 & 0 & 0 & 0 & 6.6 & 8.1 \end{pmatrix}$$



One obtains the CSC-format of a sparse matrix by interchanging rows and columns in the procedure.

See my DERIVE solution:

$$\text{to\_csc}\left(\begin{bmatrix} 8 & x & 2 & x & x \\ x & x & 5 & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & 7 & 1 & 2 \\ x & x & x & x & x \\ x & x & x & 9 & x \end{bmatrix}, x\right)$$

```
[[7, 5], [0, 1, 1, 4, 6, 7], [0, 0, 1, 4, 4, 6, 4], [8, 2, 5, 7, 1, 9, 2], x]
```

```
todense_csc([7, 5], [0, 1, 1, 4, 6, 7], [0, 0, 1, 4, 4, 6, 4], [8, 2, 5, 7, 1, 9, 2], x)
```

$$\begin{bmatrix} 8 & 0 & 2 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 7 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 9 & 0 \end{bmatrix}$$

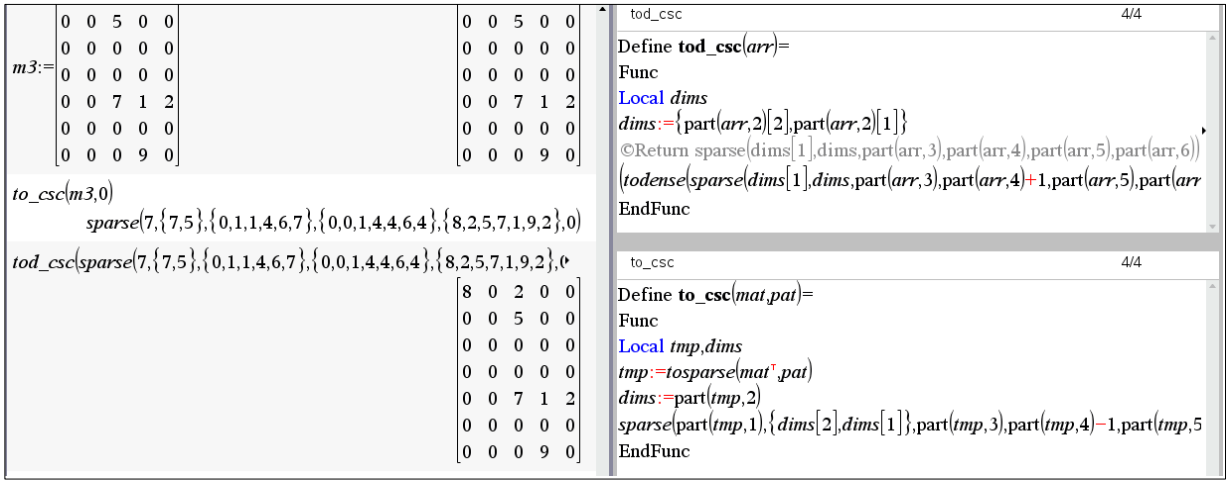
(Walter Wegscheider told me that 0-indexing has been very common for many programming languages (C/C++, Java, ..). Just PASCAL and BASIC were “outliers”. Many thanks Walter

It took me much time again and many failing attempts to convert the DERIVE code of `todense_csr` for TI-Nspire. I missed some helpful DERIVE commands. This is one reason. The other one might be that my gray cells are not in the shape as they were some time ago …

However, but finally – after two weeks vacation in Sicily – the cells worked and I am happy to present todense for TI-Nspire, too.

I must repeat my comment to the DERIVE-function: I am quite sure that there is a tighter code possible for this function. The problem are the empty rows …

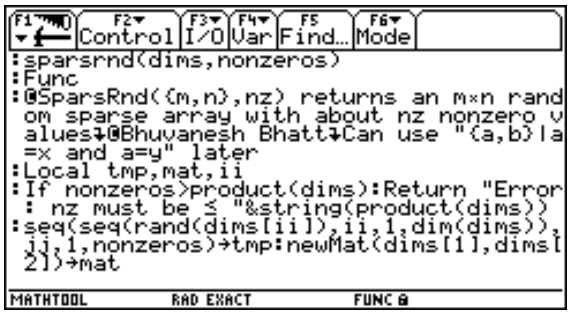
The CSC-format for TI-Nspire:

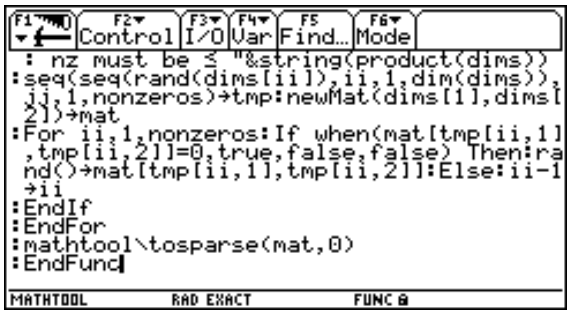


Bhuvanesh provides one more function in connection with sparse matrices: generate a sparse matrix of given dimension with given number of elements differing from the empty cells.

From Bhuvansh's Mathtool-description:

SparsRnd({m,n},nz) returns an m×n random sparse array with nz nonzero values
Needs: ToSparse
Example: RandSeed 0:SparsRnd({2,3},2) ⇒
sparse(2,{2,3},{0,1,2},{2,3},{0.733812311248,0.405809641769},0)





Testing this function, I came across some problems:

- Generating matrices with dimensions a bit higher (really not much) needs too much time – or the system hangs up (even with TI-Nspire).
- The function produces *nz* pairs (i,j) of integer random numbers $[1 \le i \le m,\ 1 \le j \le n]$, which give the rows and columns of the *nz* random numbers $0 \le r \le 1$. But the function does not consider that random generated pairs can appear twice or more often. Then we will not reach the requested number *nz* of nonzero values.
- I asked myself, why variable `pat` is introduced to mark the "zero – positions" in `tosparse` and in `todense`. It is ok, but then it would be consequent to offer this opportunity for the random sparse matrix, too.

The correct `sparsrnd` for DERIVE:

```
sparsrnd_csr(dims, nz, pat := 0, dummy, tmp, mat, ii, outp) :=
  Prog
    dummy := RANDOM(0)
    If nz > Π(dims)
       RETURN APPEND("nz must be ≤ ", STRING(Π(dims)))
    Loop
      tmp := VECTOR(VECTOR(RANDOM(dims↓i) + 1, i, 2), j, nz)
      mat := VECTOR(VECTOR(0, j, dims↓2), i, dims↓1)
      ii := 1
      Loop
        If ii > nz exit
        k := IF(mat↓(tmp↓ii↓1)↓(tmp↓ii↓2) = 0, false, true, true)
        If k = false
           mat↓(tmp↓ii↓1)↓(tmp↓ii↓2) := RANDOM(1)
        ii :+ 1
      If DIM((to_csr(mat))↓4) = nz exit
    outp := to_csr(mat)
    outp↓5 := pat
    outp
```

```
sparsrnd_csr([3, 5], 6, x)
[[3, 5], [0, 2, 4, 6], [1, 4, 2, 4, 1, 2], [0.0351, 0.955, 0.249, 0.824, 0.0731, 0.328], x]
todense_csr([3, 5], [0, 2, 4, 6], [1, 4, 2, 4, 1, 2], [0.0351, 0.955, 0.249, 0.824, 0.0731, 0.328], x)
```

$$\begin{bmatrix} 0.0351 & x & x & 0.955 & x \\ x & 0.249 & x & 0.824 & x \\ 0.0731 & 0.328 & x & x & x \end{bmatrix}$$

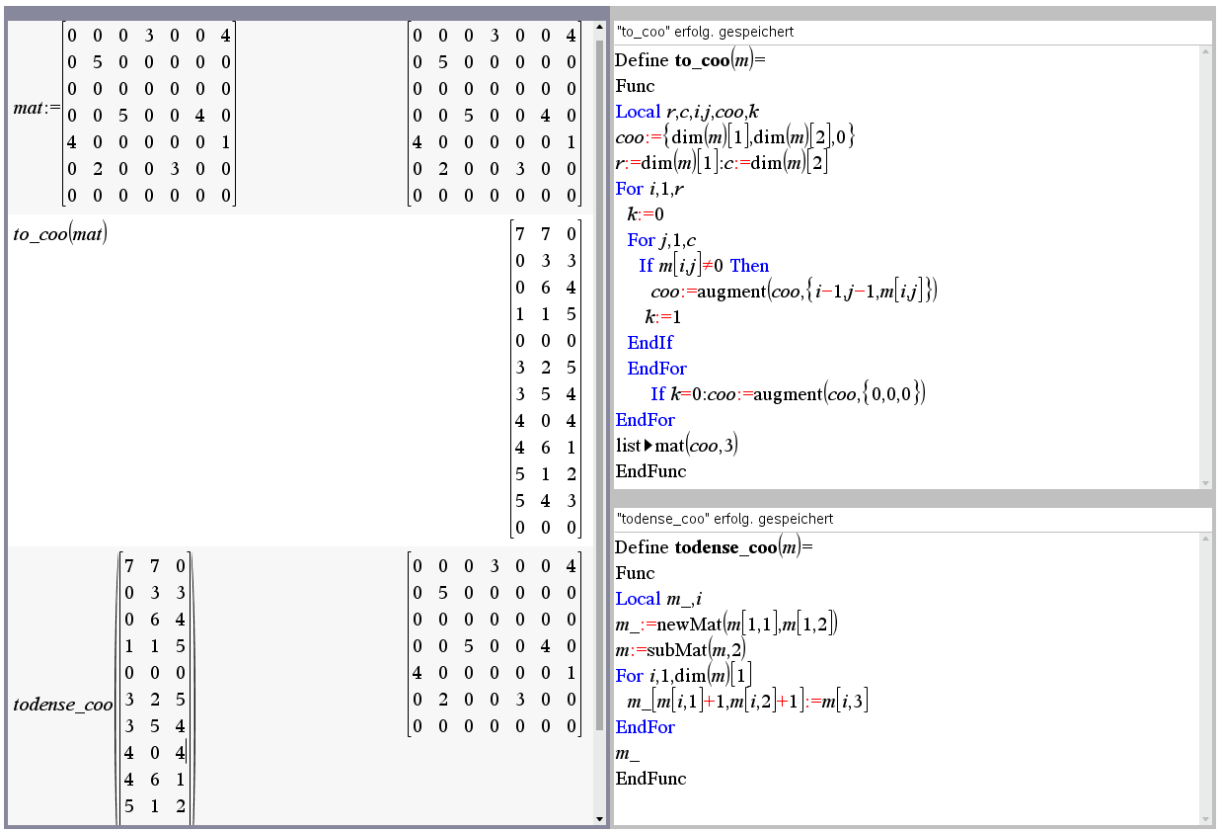
```
sparsrnd_csr([4, 4], 6)
[[4, 4], [0, 2, 3, 4, 6], [1, 3, 2, 3, 1, 4], [0.328, 0.840, 0.357, 0.500, 0.376, 0.162], 0]
todense_csr([4, 4], [0, 2, 3, 4, 6], [1, 3, 2, 3, 1, 4], [0.328, 0.84, 0.357, 0.5, 0.376, 0.162])
```

$$\begin{bmatrix} 0.328 & 0 & 0.84 & 0 \\ 0 & 0.357 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0.376 & 0 & 0 & 0.162 \end{bmatrix}$$

If you would like to generate matrices with random numbers others than elements of (0,1) you had to extend the function or to process the matrix.

When I informed about sparse matrices, I found out that there are more formats for describing them:

The COO-format:



All nonzero elements are noticed as a list (vector) [row-index, col-index, value]. I collect all lists in a matrix, indicating the “zero-rows” by [0,0,0]. The first row gives the dimension of the given matrix. I wonder, why indexing starts with 0? It might be the same reason why indexing of lists in PYTHON also starts with 0.

I had to use the output in form of a matrix, because in TI-Nspire it is not possible to work with lists of lists. In the first row you can find the dimension of the matrix. In my opinion this is necessary because in case of a "zero-col" as last column it would not be able to reconstruct the original matrix.

BTW, there was only one example in all the many websites presenting a sparse matrix with an empty row. I wonder if such matrices really don't appear in applications?

Linked Lists or in Lists of Lists are used in

LIL-Formats:

Again, we face 0-indexing of rows and columns!

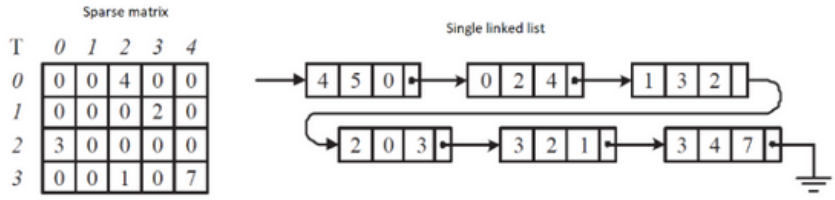
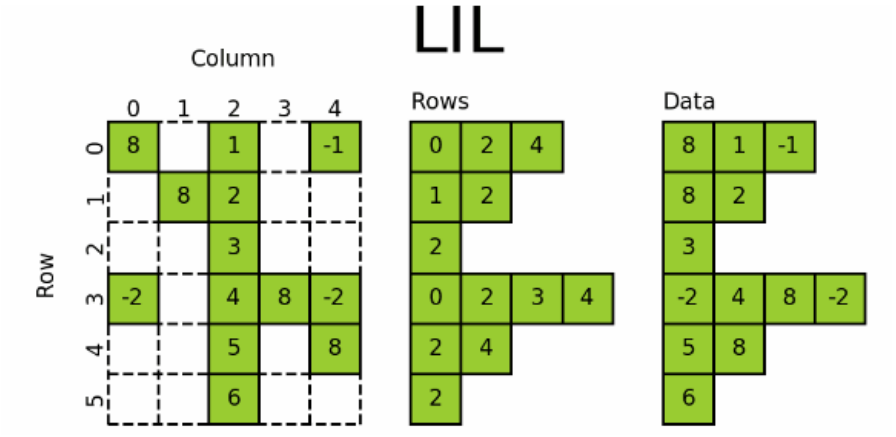


Figure 1: An example of representing a sparse matrix with a single linked list

https://www.chegg.com/homework-help/questions-and-answers/exercise-1-sparse-matrix-singe-linked-list-numerical-analysis-sparse-matrix-sparse-array-n-q37289682




```
In [32]: lil.rows
Out[32]:
array([list([0, 2, 4]), list([1, 2]), list([2]), list([0, 2, 3, 4])
       list([2, 4]), list([2])], dtype=object)

In [33]: lil.data[:, np.newaxis]  # expose jagged structure
Out[33]:
array([[list([8, 1, -1])],
       [list([8, 2])],
       [list([3])],
       [list([-2, 4, 8, -2])],
       [list([5, 8])],
       [list([6])]], dtype=object)
```

https://matteding.github.io/2019/04/25/sparse-matrices/

```
to_lil(mat, lilc, lile, le, lc, i, c, dm) :=
  Prog
    [lile := [], lilc := []]
    dm := [DIM(mat), DIM(mat')]
    i := 1
    Loop
      If i > (DIM(mat))↓1 exit
      le := []
      lc := []
      c := 1
      Loop
        If mat↓i = [] exit
        If FIRST(mat↓i) ≠ 0
           Prog
             le := APPEND(le, [FIRST(mat↓i)])
             lc := APPEND(lc, [c - 1])
        c := c + 1
        mat↓i := REST(mat↓i)
      lile := APPEND(lile, [le])
      lilc := APPEND(lilc, [lc])
      i :+ 1
    [dm; lilc; lile]
```

This is the DERIVE-code for the LIL-notation of a sparse matrix. I used empty vectors [ ] to indicate empty rows in the given matrix in both lists `lilr` (the rows) and `lilv` (the values) as well.

The first row in the result gives the dimension of the matrix. Otherwise we could not recognize a “zero-column” as last column.

$$\text{lilmat} := \begin{bmatrix} 8 & 0 & 1 & 0 & -1 \\ 0 & 8 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ -2 & 0 & 4 & 8 & -2 \\ 0 & 0 & 5 & 0 & 8 \\ 0 & 0 & 6 & 0 & 0 \end{bmatrix}$$

```
to_lil(lilmat)
```

$$\begin{bmatrix} [7, 5] \\ [[0, 2, 4], [1, 2], [2], [], [0, 2, 3, 4], [2, 4], [2]] \\ [[8, 1, -1], [8, 2], [3], [], [-2, 4, 8, -2], [5, 8], [6]] \end{bmatrix}$$

$$\text{todense\_lil}\begin{bmatrix} [7, 5] \\ [[0, 2, 4], [1, 2], [2], [], [0, 2, 3, 4], [2, 4], [2]] \\ [[8, 1, -1], [8, 2], [3], [], [-2, 4, 8, -2], [5, 8], [6]] \end{bmatrix}$$

$$\begin{bmatrix} 8 & 0 & 1 & 0 & -1 \\ 0 & 8 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ -2 & 0 & 4 & 8 & -2 \\ 0 & 0 & 5 & 0 & 8 \\ 0 & 0 & 6 & 0 & 0 \end{bmatrix}$$

Converting back to the common form of the matrix.

When I read my contribution, I found out that the COO-format is missing for DERIVE.

Look, here it is:

$$\text{to\_coo}\begin{bmatrix} 8 & 0 & 1 & 0 & -1 \\ 0 & 8 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ -2 & 0 & 4 & 8 & -2 \\ 0 & 0 & 5 & 0 & 8 \\ 0 & 0 & 6 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 7 & 5 & 0 \\ 0 & 0 & 8 \\ 0 & 2 & 1 \\ 0 & 4 & -1 \\ 1 & 1 & 8 \\ 1 & 2 & 2 \\ 2 & 2 & 3 \\ 0 & 0 & 0 \\ 4 & 0 & -2 \\ 4 & 2 & 4 \\ 4 & 3 & 8 \\ 4 & 4 & -2 \\ 5 & 2 & 5 \\ 5 & 4 & 8 \\ 6 & 2 & 6 \end{bmatrix} \quad \text{todense\_coo} \begin{bmatrix} 7 & 5 & 0 \\ 0 & 0 & 8 \\ 0 & 2 & 1 \\ 0 & 4 & -1 \\ 1 & 1 & 8 \\ 1 & 2 & 2 \\ 2 & 2 & 3 \\ 0 & 0 & 0 \\ 4 & 0 & -2 \\ 4 & 2 & 4 \\ 4 & 3 & 8 \\ 4 & 4 & -2 \\ 5 & 2 & 5 \\ 5 & 4 & 8 \\ 6 & 2 & 6 \end{bmatrix} = \begin{bmatrix} 8 & 0 & 1 & 0 & -1 \\ 0 & 8 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ -2 & 0 & 4 & 8 & -2 \\ 0 & 0 & 5 & 0 & 8 \\ 0 & 0 & 6 & 0 & 0 \end{bmatrix}$$

A Challenge for programmers:

Browsing the many resources addressing sparse matrices, I came across that some of them explain how to operate with them. How to multiply sparse matrices? Here is the answer:

> To **Multiply** the matrices, we first calculate transpose of the second matrix to simplify our comparisons and maintain the sorted order. So, the resultant matrix is obtained by traversing through the entire length of both matrices and summing the appropriate multiplied values.
>
> Any row value equal to x in the first matrix and row value equal to y in the second matrix (transposed one) will contribute towards result[x][y]. This is obtained by multiplying all such elements having col value in both matrices and adding only those with the row as x in first matrix and row as y in the second transposed matrix to get the result[x][y].
>
> https://www.geeksforgeeks.org/operations-sparse-matrices/

Let's try and multiply two matrices following the "recipe" given above:

$$A = \begin{pmatrix} 0 & 10 & 12 \\ 1 & 0 & 2 \end{pmatrix},\ B = \begin{pmatrix} 2 & 5 \\ 0 & 1 \\ 8 & 0 \end{pmatrix}$$

Then we generate the COO-form of matrix A and the transpose of Matrix B and remove the first row containing the dimension of the matrix giving matrices a and b:

$$a=\begin{pmatrix}0&1&10\\0&2&12\\1&0&1\\1&2&2\end{pmatrix},\quad b=\begin{pmatrix}0&0&2\\0&2&8\\1&0&5\\1&1&1\end{pmatrix}$$

Consequently, I use 0-indexing. (If you are uncomfortable with this, then simply add 1 to all values in the first two columns of both matrices.)

We find the elements of the 2×2 product matrix *ab* by summing up the multiplied values as follows:

$$\begin{aligned}ab_{0,0}&=a_{0,2}\cdot b_{0,1}=12\cdot 8=96\\ab_{0,1}&=a_{0,1}\cdot b_{1,1}=10\cdot 1=10\\ab_{1,0}&=a_{1,0}\cdot b_{0,0}+a_{1,2}\cdot b_{0,2}=1\cdot 2+2.8=18\\ab_{1,1}&=a_{1,0}\cdot b_{1,0}=1\cdot 5=5\end{aligned}\quad\rightarrow\quad ab=\begin{pmatrix}96&10\\18&5\end{pmatrix}$$

No more products are possible!

My Challenge for you is, trying matrix multiplication in this way for

$$A=\begin{pmatrix}3&0&2&0\\0&0&1&0\\5&1&0&2\end{pmatrix},\quad B=\begin{pmatrix}1&0&2\\0&3&0\\0&0&6\\4&0&0\end{pmatrix}.$$ Are you able to program the "COO-multiplication"?

It would be great if there are members providing one or the other function in PYTHON – or improved versions of my functions?

Last Question: What is "*Sparsity*"?

This is not too difficult. It is the fraction (number of non-zero elements of a matrix)/(total number of all elements = $n \times m$. Make your own sparsity-function!

Selection of References:

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

https://www.learndatasci.com/glossary/sparse-matrix/

https://www.geeksforgeeks.org/how-to-create-a-sparse-matrix-in-python/

https://www.geeksforgeeks.org/python-program-to-convert-a-matrix-to-sparse-matrix/

https://www.tutorialandexample.com/application-of-2d-array-sparse-matrix

http://www-lmpa.univ-littoral.fr/~jdm/jdm08/talk/wimereux_saad.pdf

https://www.cise.ufl.edu/~sahni/cop3530/slides/lec114.pdf

http://www.ece.northwestern.edu/local-apps/matlabhelp/techdoc/math_anal/sparse4.html

https://rd.yyrcd.com/CUDA/2021-12-25-GPU%20Accelerated%20Computing/15-Sparse.pdf

https://scikit-learn.org/stable/datasets/real_world.html

https://matteding.github.io/2019/04/25/sparse-matrices/

https://scipy-lectures.org/advanced/scipy_sparse/lil_matrix.html

https://phys.libretexts.org/Bookshelves/Mathematical_Physics_and_Pedagogy/Computational_Physics_(Chong)/08%3A_Sparse_Matrices/8.02%3A_Sparse_Matrix_Formats