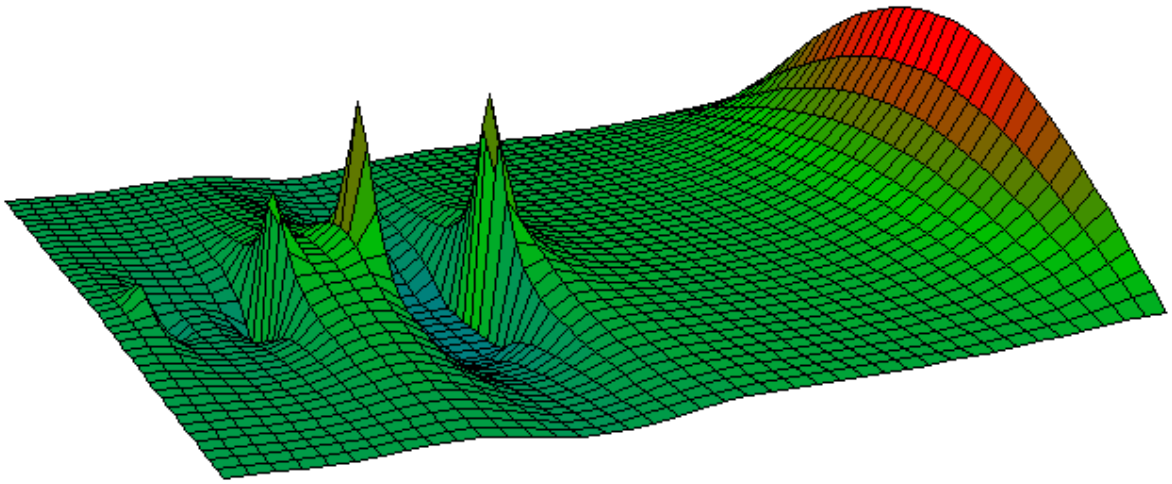
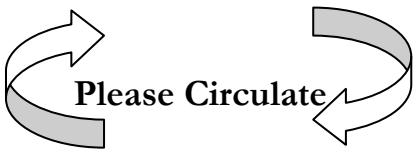
Journal of J

An interdisciplinary journal on J programming language and applications in science





MPM press

A open access Journal

ISSN: 2174-9280

Vol.4, No.2 December 2015

Triangular Fractals by Edge Inflation in J.

R.E. Boss

1. Introduction

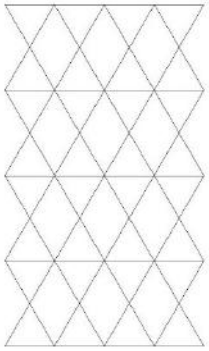


Figure 1

1. We will try to do as much drawing as possible with J, but some complicated drawings we will use Ti*k*Z, the Latex drawing program.

2. Our fractals live on the triangular grid, depicted in Figure 1 as generated by the J-code in J-expression 1, which produces this famous grid.

```
require 'plot'
options=:'tics 0; frame 0; grids 0; color black; labels 0;aspect '
Rl=: ({.,.L:0{:)/.@|.   NB. Lines to the Right
Ll=: ({.,.L:0{:)/.      NB. Lines to the Left
Hl=: ({.,.L:0{:)"1      NB. Horizontal lines
t=:(2%~ 1, %:3) *"_1 >,"2 L:0/(0 2|:>)L:1(Rl(,~<)~ Ll,&< Hl) |:{2#<i.5
(options,":%~/(>./-<./)"1 ,"2 t) plot  ;/t
pd 'save jpg  "F:\JoJdec15\triangular_grid.jpg"'
```

J-expression 1

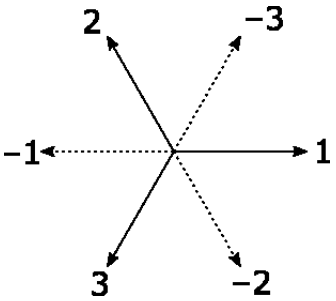


Figure 2

3. From the three main axes we will denote the *positive direction* with 1,2 and 3, as in Figure 2 at the right, and their counter direction with –1, –2 and –3.

4. The fractals we will study will be *walks* on the grid (always triangular) which are directed graphs. But we will consider them as an ordered set of *edges* of which each consecutive pair of edges share a vertex. And such an edge is determined by its direction, so a walk is a *sequence of edges*, e.g. by $\langle 1,-3,2,-1,3,-2\rangle$ to denote a hexagon, as you can check yourself.

5. The first vertex or start of a walk, mostly the origin, but often unknown, is called the *entry*, the last vertex the *exit*. A walk can visit a vertex or an edge zero or more times. A walk is called *Hamiltonian* if a vertex is visited at most once and *Eulerian* if an edge is visited at most once. A Hamiltonian walk, or a *path*, is also Eulerian; a vertex in an Eulerian walk (on a triangular grid) can be visited at most three times.

2. Some tools

6. An important tool which is easy to implement in J is the so called *signed permutation*. It is a permutation σ on a, mostly finite, set of consecutive numbers $\{1,2,\cdots$, called the *digiset* (to distinguish it from *alphabet*) and its negatives, in J to be generated by the J-expression 2 at the right. If $\sigma(-x)=-\sigma(x)$, then we say it is a *signed* permutation. As one can immediately see, we have $n!*2^n$ signed permutations on *n* positive numbers which are determined by their images on those positive numbers. So we will σ denote by $[\sigma(1),\sigma(2),\cdots$.

```
   ,(,.-)>:i.5
1 _1 2 _2 3 _3 4 _4 5 _5
```

J-expression 2

7. On the triangular grid, signed permutation come in real handy. E.g. $[-3,-1,-2]$ is the rotation over 60° as one can see from Figure 2 immediately. But there is one caveat, since the three directions 1,2 and 3 are not independent, not all combinations of numbers are allowed to represent geometrical transformation. We cannot have $[-1,3,2]$ for example.

8. Signed permutations of course can be multiplied and inverses can be taken. Here are the J-verbs.

```
NB. In J, sgndprm applies a (signed) permutation to a sequence, invprm deter-
mines the inverse

sgndprm=: ({ 0,(,-@|.))~"1
      NB. x is the signed perm, y is the sequence to be transformed

   7 6 5 4 3 2 1 sgndprm 7 _2 5 _4 3 _6 1
1 _6 3 _4 5 _2 7
      NB. applying x after y produces a new signed perm, the product of x and y

invprm=: ([:/:~/ [:(*"1 *@{.)  (,: >:@i.@#))"1
      NB. y is the perm to be inverted
   (sgndprm invprm) 7 _3 6 4 2 _5 1
1 2 3 4 5 6 7
```

J-expression 3

9. Another tool we will need is the *substitution*. We will be able to construct fractal walks by substituting one edge for some other edges, increasing the length of the walk. By definition, given a substitution S and a digiset D, we assume there is an $x \in D$ such that $\lim_{n\to\infty}|S^n(x)| = \infty$, or in words, the length of the walk generated by S on x is unbounded. A walk F is called a *fractal* if $S(F) = F$. If $\lim_{n\to\infty} S^n(x) = F$ is a fractal for $x \in D$, the we call the sequences $S^n(x); n = 0,1,\cdots$ the *approximants* of F.

```
 0  0  0  0
 1 _3 _2  1
 2 _1 _3  2
 3 _2 _1  3
_3  2  1 _3
_2  1  3 _2
_1  3  2 _1
```

J-expression 4

10. A substitution is given by *rules* $S(x)$ for each $x \in D$. In J-expression 4 is a table given for the rules of a substitution S [A] on the triangular grid. In J-expression 5 at the right is how to apply a substitution, given its rules.

```
   S=:0,(,-@|.)(, 2 3 1&sgndprm^:(1 2)) 1 _3 _2 1
   S (,@:{~) _2
_2 1 3 _2
   S (,@:{~)^:(2) 1
1 _3 _2 1 _3 2 1 _3 _2 1 3 _2 1 _3 _2 1
```

J-expression 5

3. Koch curve

11. One of the best known triangular fractals is the Koch curve. Let we investigate how to construct it. In

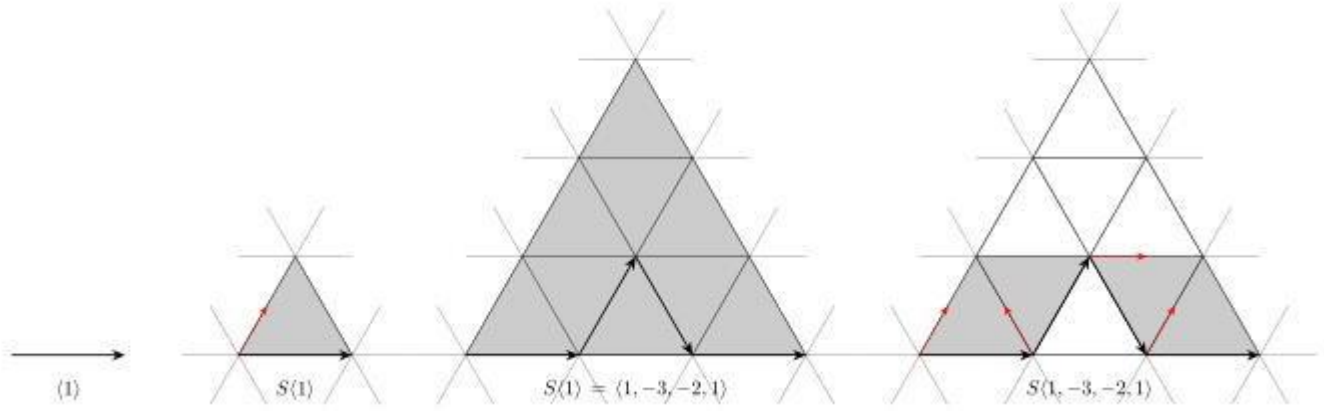


Figure 3

Figure 3 (drawn in Ti*k*Z) one sees the stages to substitute the first approximant, which is an edge, to the next approximant, which has $\langle 1,-3,-2,1\rangle$ as representation. In the second picture a gray triangle is indicated by the edge and a small, red arrow, this is the triangle to which the edge will iterate. The third picture is enlarged by a factor 3 and the new edges of the next generation are drawn. In the last picture the triangles to

[A] This table was constructed in J by `S=:0,(,-@|.)(, 2 3 1&sgndprm^:(1 2)) 1 _3 _2 1`

which these edge will iterate are indicated as well. Notice that this is the same substitution as in paragraphs 9 and 10.

12. I will provide a simple plot function *frctl* with which I produce most of my graphs.[B]

```
frctl=: 3 : 0
 pd 'reset'
 pd 'tics 0; type line; frame 0; color black; grids 0; labels 0'
 pd <"1 |: y
 pd 'aspect ',":%~/(>./-<./)"1 |: y
 pd 'show'
)

  s=: 0,(,-@|.)(2%~ 1, %:3) *"1 [_1 _1,~ _1 1,:~ 2,0
NB. the base vectors for the triangular grid

   S=:0,(,-@|.)(, 2 3 1&sgndprm^:(1 2)) 1 _3 _2 1   NB. substitution rules
   frctl +/\s{~ 0, S (,@:{~)^:(2) 1                 NB. the 3rd approximant, cf Figure 4.
```

J-expression 6

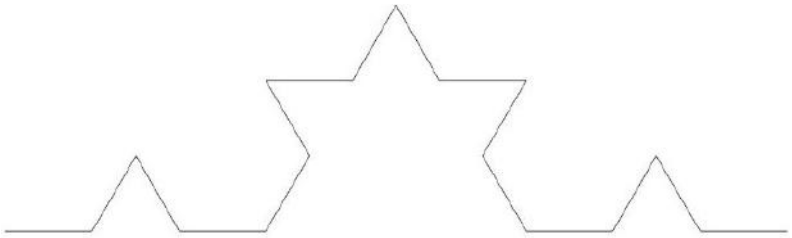


Figure 4

13. So the reader can now experiment with some other approximants by altering the power of the verb.

4. Constructing simple ones

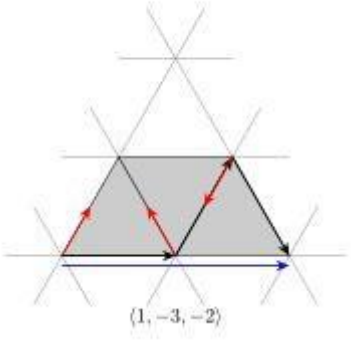


Figure 5

14. We can easily simplify constructions compared to the Koch curve which is on a large triangle, subdivided in nine smaller ones. Let's look at a triangle which is divided in 4 smaller ones, like the one in Figure 5. Here the original edge was $\langle 1 \rangle$, as usual, and is enlarged by a factor 2 and became blue. The triangle it grows in is the large triangle, subdivided in 4 smaller ones. The original edge is substituted by the edges $\langle 1, -3, -2 \rangle$, each of which grows into the small gray triangle indicated by the red arrow which shares its entry with the corresponding edge. So the first substitution rule $S\langle 1 \rangle = \langle 1, -3, -2 \rangle$

15. But we have a bit of a problem. With the Koch curve, all edges iterated to triangles at the left, as can be seen from Figure 3. But now, in Figure 5, one edge, the last one, iterates to the *right*. For that reason and to make it not too complicated, we create alternate edges, 4 will correspond with 1 but will iterate to the right and likewise 5 to 2 and 6 to 3. So the first substitution becomes $S\langle 1 \rangle = \langle 1, -3, -5 \rangle$ since the last one iterates to the right. And the substitution to the right will be the reflection of the one to the left, so $S\langle 4 \rangle = \langle 4, -5, -3 \rangle$ So first we have to generate the 12 substitution rules in *S* which is done in J as follows. And after that we have to convert the 4,5,6 edges back to the 1,2,3 ones to make the final drawing, a structure loaded with

```
   S=: 0,(,-@|.),/(, 2 3 1 5 6 4 sgndprm^:(1 2) ])"1[ 1 _3 _5,:4 _5 _3
   frctl +/\s{~ 0, 1 2 3 1 2 3 sgndprm S (,@:{~)^:(5) 1
```

J-expression 7

[B] Actually, I do have one which is a bit more sophisticated since that uses the plot verb *arrow*, but I will not introduce that now.

what look like circular saw blades, but not depicted here.

16. But there are other possibilities on the 4-cell triangle, essentially only two. See the pictures in Figure 6. They are left as an exercise for the reader, who only has to determine the first substitution rule.

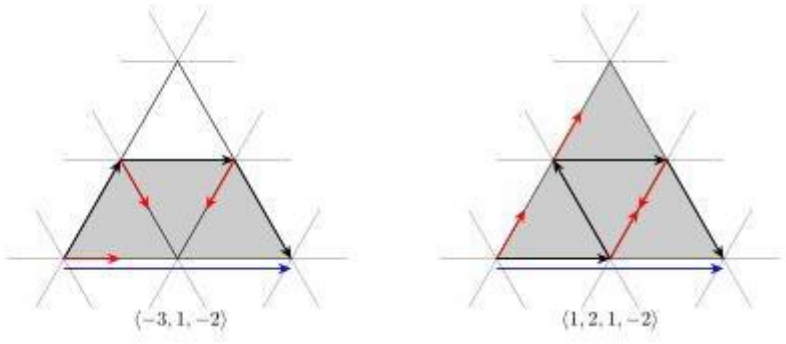


Figure 6

5. Fractal dimension

17. One could easily write a book on this subject, which is done already a number of times, so I will keep it short and simple. As you can see with the Koch curve, only 4 small triangles of the big one are used for further iteration, so one can conclude that only $4/9$ of the original triangle will be occupied. But this holds for the small triangles as well, so the surface of 5^{th} approximant of the Koch curve will be less then $\left(4/9\right)^5$ and finally this will tend to 0.

18. But the fractal *box counting dimension* can be calculated by $\log(N_n)/\log(r^n)$ where N_n is the number of boxes, triangles here, and *r* the multiplication factor applied on the original edge. Since $N_n = 4^n$ and $r^n = 3^n$ we get for the fractal dimension of the Koch curve `4 %&^. 3 = 1.2618595` .

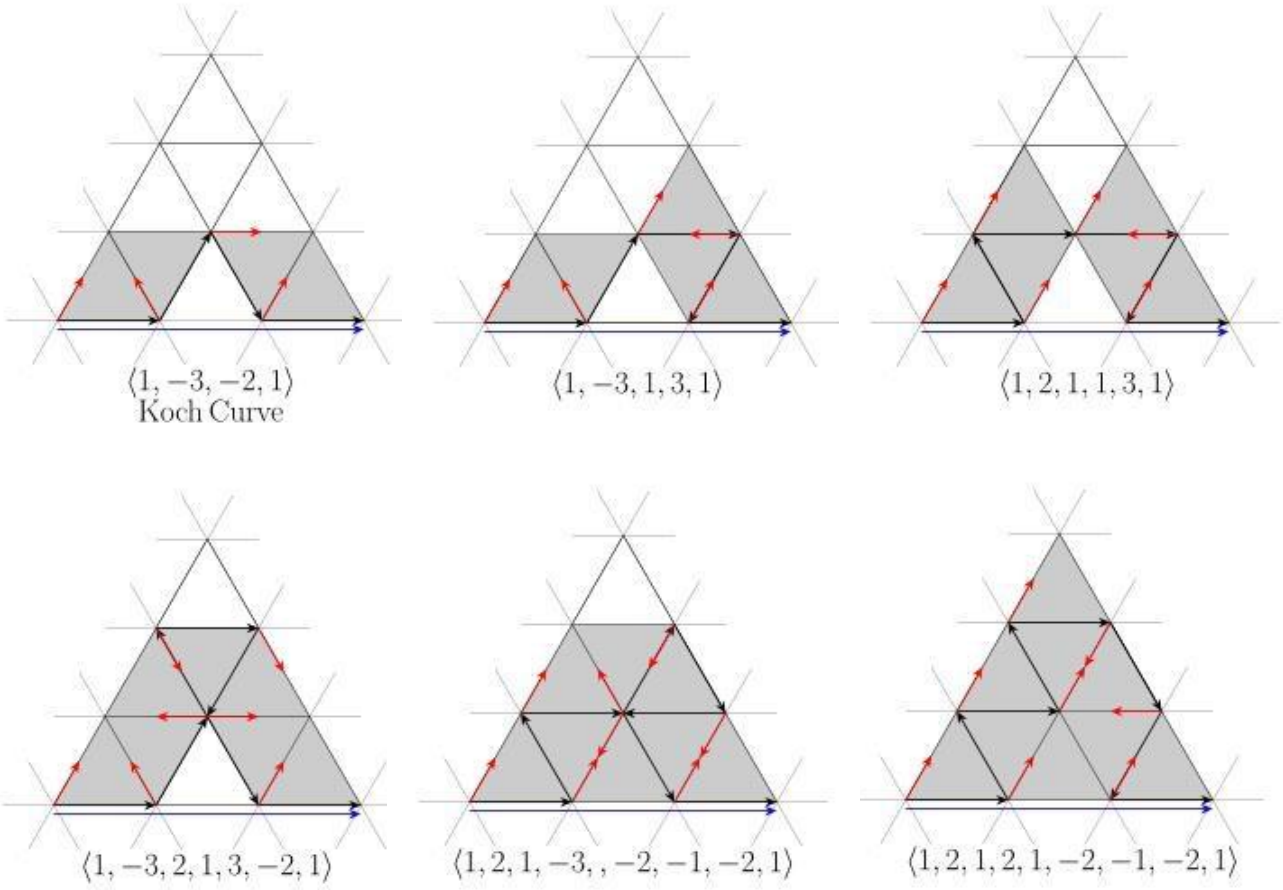


Figure 7

6. Koch curve's nephews

19. In Figure 7 the six fractals are indicated with fractal dimension equal to $\log(k)/\log(3)$ for $k = 4,5,6,7,8,9$. With the red arrows as indicated one can easily derive the first substitution rule and with the first line from J-expression 7 construct the other ones.

7. How to generate triangular curves

20. Now we have seen a few fractals on the triangular grid, it is easy to formulate a general rule and produce any triangular fractal you want. First draw an arbitrary line, like the blue ones in Figure 8. Second, make that in an equilateral triangle, with the red edges. Third, construct a walk from the entry of the first edge to its exit, where it is allowed to visit vertices and edges more than once.

21. Now you have the generator, make the substitution rule, choosing to which side each new edge will iterate. Produce the other rules and generate each approximant you like.

22. If I work out approximant 5 of the larger triangle from Figure 8, I get Figure 9, be it rotated back in the same position as the original edge. The code is to be seen in J-expression 8.

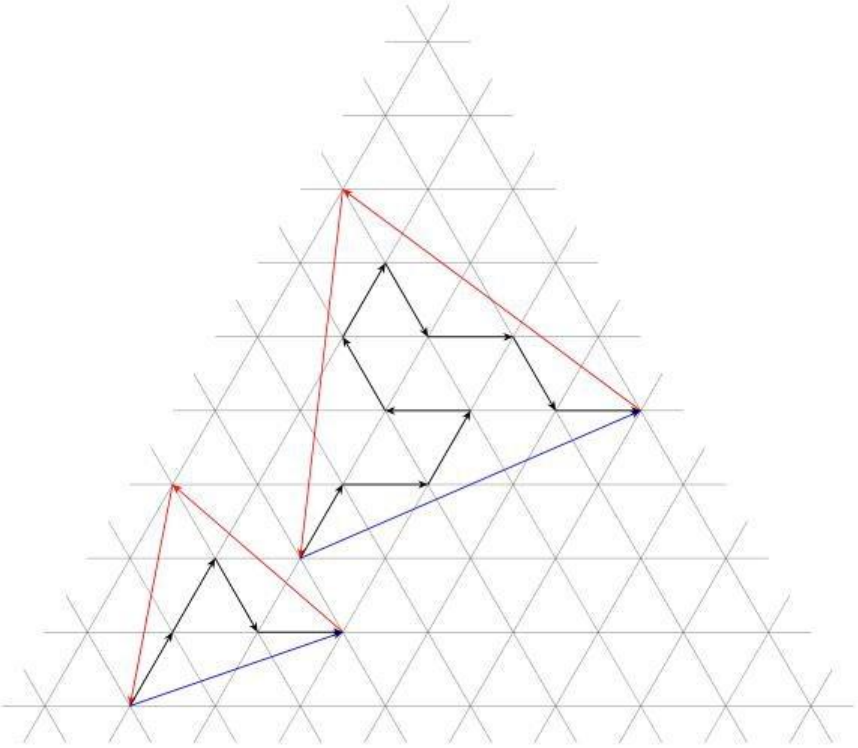


Figure 8

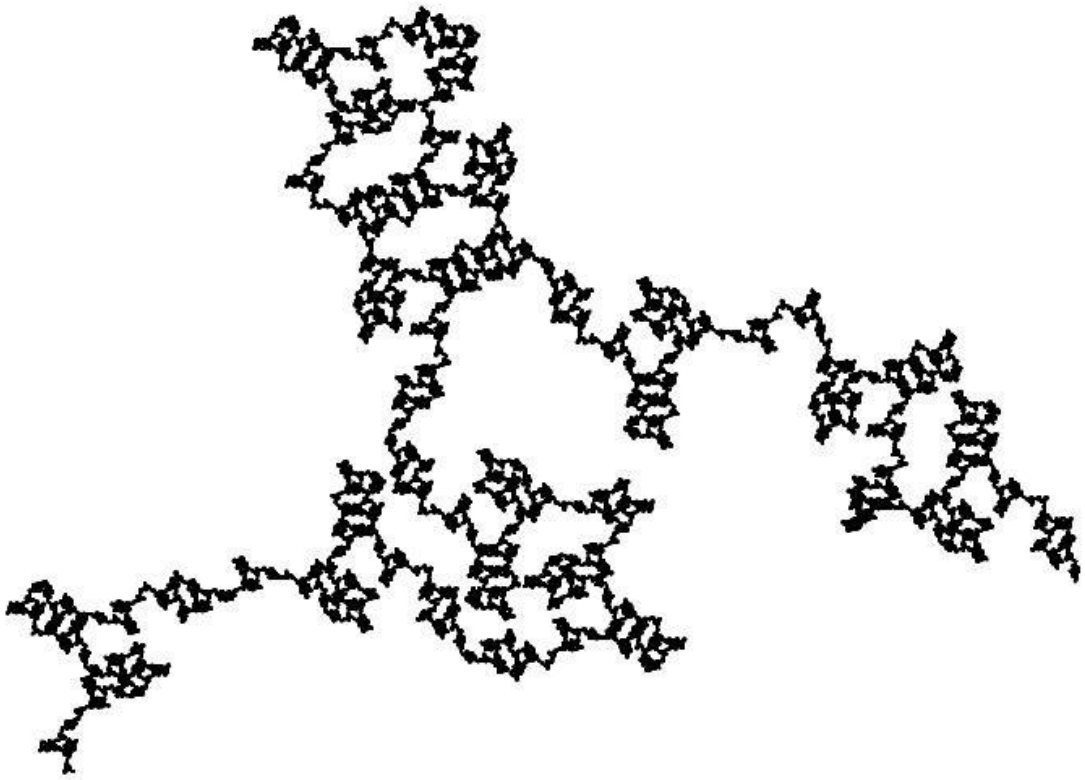


Figure 9

```
   S=:0,(,-@|.),/(, 2 3 1 5 6 4 sgndprm^:(1 2) ])"1 [(,: 4 5 6 1 2 3 sgndprm |.) _3 1 _6 _1 5 _3 _5 4
_2 1          NB. wraparound!

   frctl +.(1 r._4* _7 o. 4 (%~%:)3) * j./"1 +/\s{~ 0, 1 2 3 1 2 3 sgndprm S (,@:{~)^:(5) 1
      NB. the part (+.(1 r._4* _7 o. 4 (%~%:)3) * j./"1)  is used to rotate the 5th approximation back
to the original edge position
```

J-expression 8

8. Conclusion

23. We told you some over triangular grids and how to generate them by edge inflation and draw them with J.

24. Next time we continue with other fractals, also square and cubic ones, some of which are constructed in quite a different way.

25. A bonus here is the picture of the Gosper flowsnake, as first given by Martin Gardner, in Figure 10. The J expressions to generate it also comes next time.

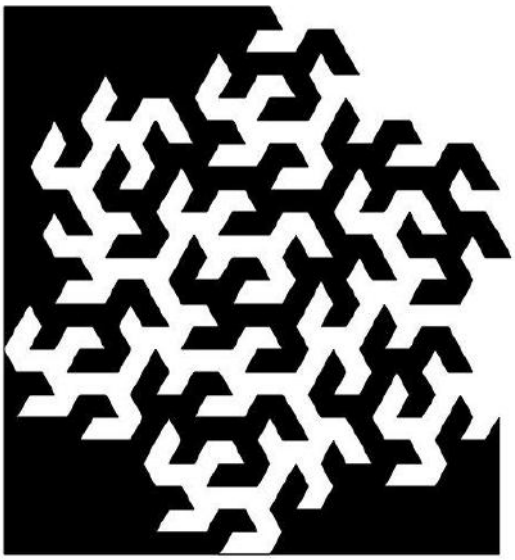


Figure 10

```
NB.Sergey Kamenev
NB.Command line program to calculate weight if you know optimal weight index.

#!/usr/bin/env jc

NB. First arg is height (meters), second - sex (1 - male, 0 - female),
thirt - current weight, forth - goal index
NB. ./ves.ijs 1.925 1 104.3 27

NB. Height
rost =: ". > 2 { ARGV_j_
NB. 1 - male, 0 - female
sex =: ". > 3 { ARGV_j_
NB. weight
ves =: ". > 4 { ARGV_j_
NB. need index
need_index =: ". > 5 { ARGV_j_

NB. y - weight
imt =: %&(*:rost)
imt_big =: [: *&1.1 %&(*:rost)
imt_small =: [: *&0.9 %&(*:rost)

bBig =: ((sex = 1) *. rost > 1.88) +. (sex = 0) *. rost > 1.74
bSmall =: ((sex = 1) *. rost < 1.68) +. (sex = 0) *. rost < 1.54
bNormal =:  *./ -. bSmall, bBig

NB. Smart select of gerundy
eimt =: imt_small`imt`imt_big@.(2 ^. #. bBig, bNormal, bSmall)

echo 'Current index ', ": eimt ves

echo 'Weight (goal): ' , (": eimt ^: _1 need_index), ' kg.'

exit ''
```

Assembly/C types are integer, character, float and so on.

J (and other dynamic languages) understands all of these types. The (3!:0) verb will return the type of any noun. Whether a language is statically or dynamically typed, all functions tend to have a limited range of valid input parameters.

Statically typed languages (explicit types provided in function Signatures) Documents names and types of parameters. Whole program anaylsis can catch errors at compile time when incorrectly typed calls are made. Disadvantage: wordiness, and if sticking to assembler compatible types, lack of flexibility and expressiveness.

Inferred typed languages (implicitly fabricate the types in function signatures) Eliminates the wordiness, but then also the documentation advantage, of explicit typing.

Dynamically typed langages (No types just parameter names) Best practice is to validate at the start of functions. The concept of coercion is to take missing and variable length parameters and make sense of them. There is also flexibility to take what would be wrongly typed parameters in strongly typed languages and fix the errors before processing.

Comment type systems (use structured comments to describe and possibly through file processing code do more) Often used in J's addon libraries for documentation and explanation. Can be a source of problems when comments are not updated with code changes.

And then there is J (No parameter names, unlimited number of parameter counts and dimensions) The above 2 methods for ensuring input types have historically been used in J. But, adverbs/conjunctions provide a way to bind type information in the function header, and through a descriptive DSL can provide a type system more complete and flexible than in any other known language. A type system in a dynamic language doesn't need to be limited to providing helpful errors for wrong use. An even more powerful use of types is to coerce inputs into the "correct" expected values, and this happens to be the power of adhoc validation in dynamic languages.

A brief and technical overview of the type system

http://code.jsoftware.com/wiki/User:Pascal_Jasmin/record_and_type_system

A type in J

has 4 fields to define it: type name: (int str ....) an identifier that applies the type code to an input coercion code: function that is executed if validation guard (3rd field) is false validation guard: boolean function meant to test if input is of this type error text: text to display if validation or coercion fails.

A type processor

validator (v) and coercer (c) are the 2 main useful type processors. They are naturally conjunctions. I've implemented them with the concept of a double adverb (a conjunction where both parameters are to the right. The advantages in this context: Returning an adverb from an adverb means compiling away any type lookup code and other potential optimizations, though this benefit has not been applied by me extensively yet). The github readme spends a lot of space discussing all of the alternative type processors, and they are intended to be useful, but are all mere variations of coercion and validation.

the core type coercer processor

```
    0 ".^:(1 4 8 16 64 128 e.~ 3!:0)
```

is the common J guard pattern of doing nothing (other than returning y) if the condition is false or coercing/transforming and returning that result if true.

This is the numeric type. 0&". is the coercion code, (1 4 8 16 64 128 e.~ 3!:0) the validator test. The type system dsl chains these types of simple guard expressions based on the type processor, and using more english/standard type interface names.

the core simple (paramaterless) types some types have parameters to them (next section). Simpler types do not.

```
   num  0&".                                1 4 8 16 64 128 e.~ 3!:0     Must be n
   int  intify                              1 4 64  e.~ 3!:0             Must be i
   intx [: x:@:intify (,&'x')^:(2 = 3!:0)           64  e.~ 3!:0         Must be e
   intR roundify                                    1 4 64  e.~ 3!:0             M
   str  ":                                  2 = 3!:0                     Must be s
   box  <"_1                                0<L.
        Must be boxed
   byteVals    a.&i.                         0 255&(inrange :: 0:) *.
1 4 e.~ 3!:0   Must be convertable to byte list
   ascii       ('unconvertable' raiseErr ])`utf8@.(1 4 64 131072 e.~
3!:0)          2 = 3!:0       Must be convertable to ascii/utf8
   text utf8`uucp@.(0 255&inrange)                  2 131072 e.~ 3!:0           M
   short       fucp                   0:`(0 65536&inrange)@.(1 4 64
e.~ 3!:0)      Must be number in range of 0 to 65536. Unicode and
negative numbers will be converted.
   no1dim      linearize                                  [: -
.@:notfalse 1 e.~ $    Must not include any shapes of 1
   items       'itemless' raiseErr ]                0 < #                   Must not
   evals       3 : 'y eval'                         (2 ~: 3!:0)             Must be e
   cuts cut                                 0<L.
        Must be boxed or string will be cut on spaces
   words       ;:                                   0<L.                    Must be b
   dltb [: dltb leaf ('str') cV leaf ]      2 32  -.@e.~ 3!:0            Will trim
   localized   3 : 'y locs'                         '_'&e. *. _2 <
4!:0@:< Name (str) must be valid and localized
   uucp uucp                                131072 = 3!:0
        Must be unicode
   any  ]                                   1:"_
        Test will always pass. Any param.
```

each line is a type. Its 4 fields TAB delimited.

some types only make sense as coercers. They can (and much of the above do) use defined J functions. For coercion, only certain types of "invalid" inputs can be coerced. A "wrong" num value must be string for example. There are obvious items mission (floats for example), and I will add them one day I need to use them, but you can add system types as well by just creating an extra line in the typesys.ijs file. Many examples also show how to add private types to a locale.

paramatered types These types allow one (word) parameter to them. A word is defined by a token from ;: So a list of numbers is one word, and arbitrarily complex parameters are possible by encoding them as a list of numbers (theory is that everything in the universe has such a parseable encoding)

```
   count       maybenum {.Fxhy ]              maybenum =Fxhy #                    c
   mthan       maybenum {.Fxhy ]              maybenum <Fxhy #                    c
   fthan       maybenum {.Fxhy ]              maybenum >Fxhy #                    c
   gthan       maybenum@[ >. ]        maybenum@:[ *./@:<: ]
        Must be greater or equal than %s
   lthan       maybenum@[ <. ]        maybenum@:[ *./@:>: ]
        Must be lesser or equal than %s
   inrange     'unconvertable' raiseErr ]    maybenum@:[ inrange  ]       Must be w
   unboxed     <@:[ cV every ]        0 = L.@:]                            Must be c
   each <@:[ cV each ]         [: */@:; <@:[ vbV each ]
        Must be coerceable to parameter %s (type) then  leaves boxed
   every       <@:[ cV every ]        [ vV"_ >@:]
        Must be parameter type %s leaves boxed (use each instead
normally)
   d    maybenum@[             0 < #@]                       Default
value is %s
   dv   4 : '(x eval)"_ y'             0 < #@]
        Default value is verb(named if compound)  %s (applied to null)
or value of  %s  within locale if not specified
   cut  maybenum@:[ multicut ] 0<L.@:]                       Must be
boxed or string will be repeatedly cut by %s (if list must be
numberic. if numeric, numbers are ascii values)
   copies      maybenum@:[ copylist ] maybenum@:[ copylistV ]              Must have
   evalto      'unconvertable' raiseErr ]    4 : 'a=. y eval
label_.(maybenum x) = 4!:0 <a' Str expression must eval to name
class %s. 0 noun, 1 adverb, 2 conjunction, 3 verb
   level       <@:]^:(maybenum@:[ - L.@:])   maybenum@:[ = L.@:]                b
```

parametered types are entered as param&type (& is special parse for parameter), and compound types are possible. For example

```
    ] 'int 5&lthan' c
```

will coerce the input first to int, and then such that it is less than 5.

the maybenum function allows support for both numeric and string parameters which are all strings within the dsl. It is also a powerful data structure used in my todo example database app in letting data be typed like what it looks like, providing a fundamentally higher level of dynamism.

the parameter of a type is the x argument to both the validation test and the coercer.

Some of the more useful types defined are default values, and (multi)cut which is a recursive version of cut that turns text input into multinested boxes.

towards records multi parameter functions are common in other languages, and the type system is designed to simplify this in J

```
3 : ' b [ a b c =. y'("1) 1 ; 2 ; 'three'
```

is the typical way of writing and calling a multiparameter function in J. Each parameter can be of different types. The ("1) doesn't do much above but it allows the function to process a list of records in typical J style.

The record system is more removed than the type system from the functions that use it. A record is a dataless object. Its functions are field coercers and optional validators (after coercion), a preparsing (whole record) "coercer", post processing function (can do anything at all but whole record validation is most likely), and field accessor verb names. These are all lighter weight than you might fear.

towards a record form system

The primary design for the record system was to allow highly complex forms to be defined into single line statements. This includes both form parameters including data names, help text, tooltips, control representation, and dealing with coercing data into form controls (usually string) and out of form controls (coerced from string to types) for the record processing function that is a callback to a completed form. The record validators also allow a form to prevalidate input before letting it access the callback. So it allows both simple single line form creation, and simple form result processing (input validated prior to hitting function). The callback functions can also be called directly without the form

Another essay will describe the form internals, but as an overview. Though the current design is tied to jqt (works almost on jandroid, but works on jqt android) the backend is entirely self contained and replaceable. Form addons include an attached console (designed with some flexibility for formatting htext/html/text type, though current setup is htext with just dead code for the other options) and extra function (callback) buttons.

A J Solution to
"A Superior Mathematical Puzzle"

(from H P. Dinesman)

John C McInturff
(10 21-2015)

Word Statement

Baker, Cooper, Fletcher, Miller and Smith live on different floors of an apartment house that contains only five floors. Given the following statements,S, determine on which each person lives.

S1: Baker does not live on the top floor . (f4).
S2: Cooper does not live on the bottom floor, (f0).
S3: Fletcher does not live on either the top or the bottom floor.
S4: Miller lives on a higher floor than Cooper.
S5: Smith does not live on a floor adjacent to Fletcher.
S6: Fletcher does not live on a floor adjacent to Cooper.

Terminology

```
   ]Names=. ;:'Baker Cooper Fletcher Miller Smith'
```

Baker	Cooper	Fletcher	Miller	Smith

```
   p=. i.@!A.i.
   'f0 f1 f2 f3 f4'=. A=. |:a=.  p 5
   'bot top'=. f0;f4
   'b c f m s'=. n=. i.5
   on=. e.~
   ip=. +/ . *
   k=. (n ip A e.]) (&>) n
   'B C F M S'=. k
   adj2=. 1=[: | -
```

Logic Statement

```
   s1=. -.b on top
   s2=. -. c on bot
   s3=. (f on bot) +: (f on top)
   s4=. M > C
   s5=. -.(S adj2 F)
   s6=. -.( F adj2 C)
```

Solution

```
   z=. >(s1;s2;s3;s4;s5;s6)
   Z=. */z
   +/Z
1
   Z # i. 120
102

   (Z # a);'';'BCFMS'
```

4 1 0 2 3		BCFMS

```
   (Z # a) { 'BCFMS'
SCBFM

   (Z # a) { Names
```

Smith	Cooper	Baker	Fletcher	Miller

Paralleling of XYZ coordinate-triplets by user-defined unicodes

István Kádár

1. We used the available in **MS Windows XP** and **Windows 7** *eudcedit.exe* and *charmap.exe* character-editor for our experinents. We obtain 64x64 pixel grids from *start* and *run* fields for using mouse editing. The saved characters found after starting the *charmap.exe* and copied out ***.doc***, ***.ppt*** or ***.txt*** files. (*„**EUDCEDIT**” = **E**nd-**U**ser-**D**efined **C**haracter **EDITOR***). Thanks for Microsoft!

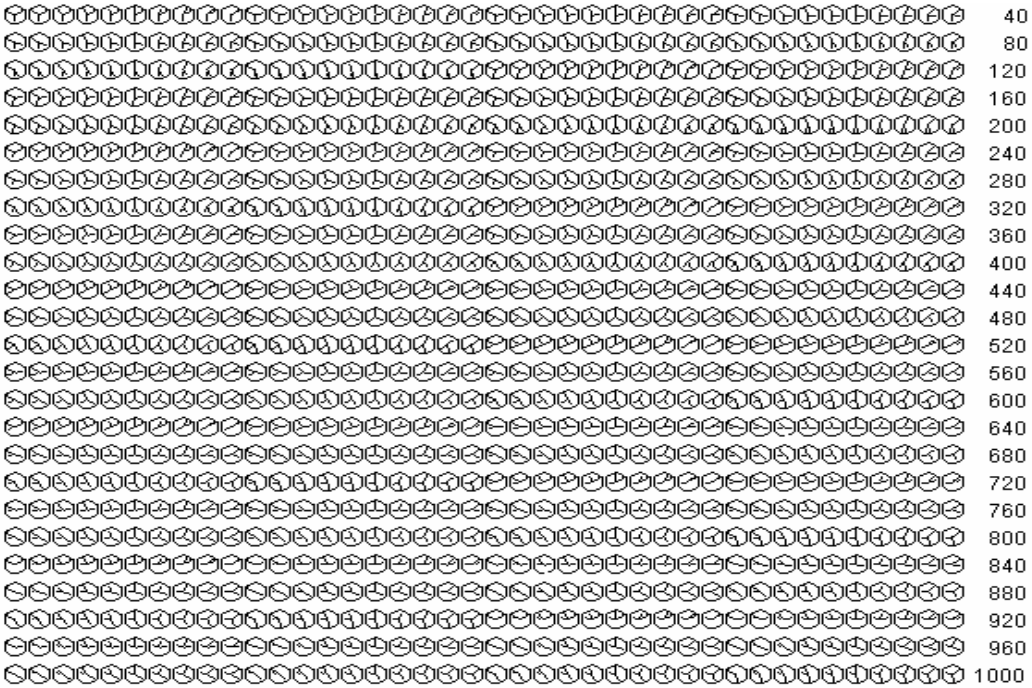


Fig.1 *Self-made spatial decimal digit set (000-999)*

```
  ]XYZ=:(10#10)#:x: 1000 * 4239593.146  1350516.190  4554628.330
4 2 3 9 5 9 3 1 4 6   X
1 3 5 0 5 1 6 1 9 0   Y       from COORDINATE-organization      ( 3 coordinates)
4 5 5 4 6 2 8 3 3 0   Z
                              After transpose
  |:(10#10)#:XYZ
4 1 4  Mm
2 3 5   …                     PLACE VALUE-organization          ( 10 diordinate )
3 5 5   …
9 0 4  km                     from COORDINATES to DIORDINATES
5 5 6  hm
9 1 2   …                     CONVERGENT - DIVERGENT model of opposites
3 6 8    m                       Triple group than  VOLUME calculation,
1 1 3  dm                        Dual group than AREA calculation.
4 9 3  cm                        Tetra group in SPACETIME
6 0 0  mm
```

```
                                        X              Y              Z
 ]DIGI=:   10#.|:(10#10)#:x:1000 * 4239593.146  1350516.190  4554628.330
414  235  355  904  556  912  368  .  113  493 600
```

Fig.2a*. DIGITAL spatial number from 30 digits*

]ANAL=:hms{~DIGI NB. hms is the character list on Fig. 1 from 1000 elements

Fig.2b *ANALOG spatial number from* 10 *digits*

As you can see magnified, each with its own index recorded 120 degrees latitude, so unlike the usual hour indicators do not beat around and not overlap, so their lengths can be the same. It has also interesting property that is reduced when you have the additional features dividing point lines do not stand out, it is still clear to the people reading. **Quite simple to decide which one vertex is "closer" or "very close" or "on" indicator and therefore are looking for decimal numbers after a practice has been "familiar." Such a decimal space clock could be the subject of patents, even if we take into account also, that instead of hexagon for two index (2D) peak of a square and for four index (4D) is also the peak set used in place of a regular octagon can be used.**

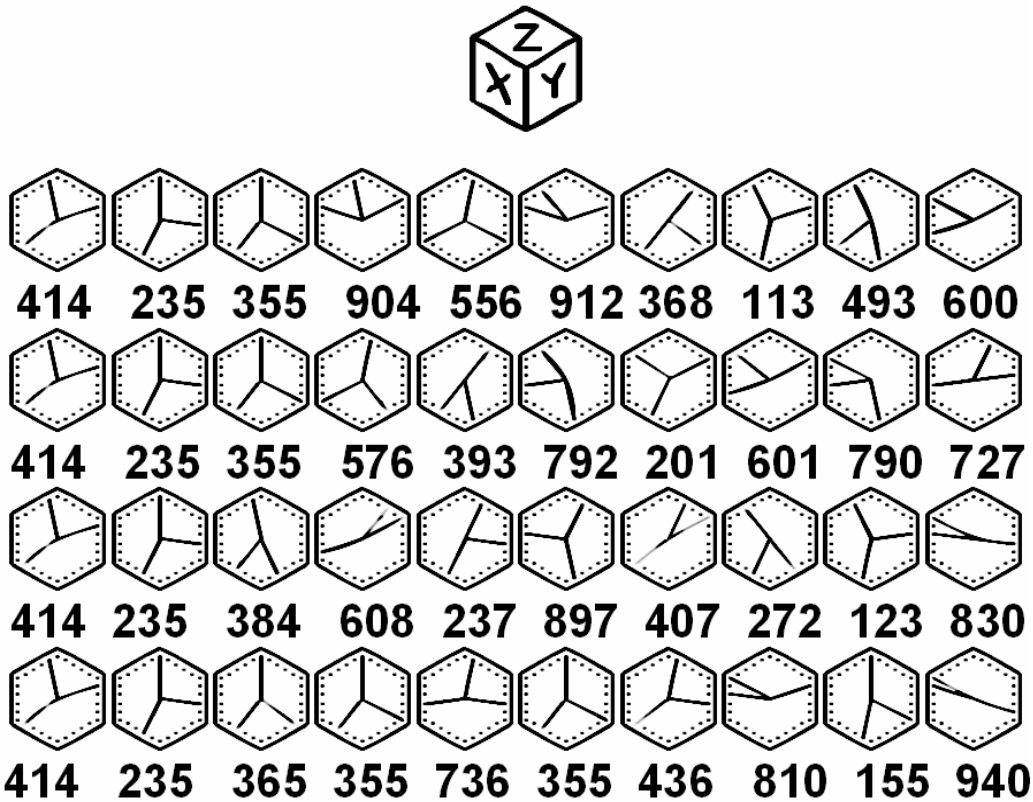


Fig.3a *Reading exercise examples.*

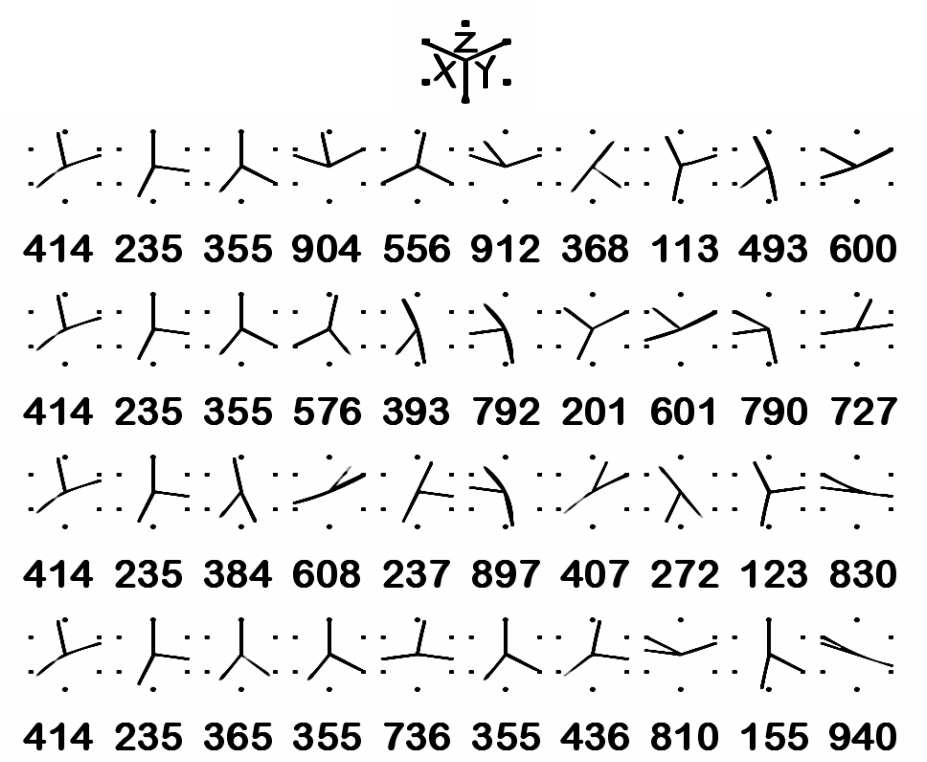


Fig.3b *Examples of the simplified version.*

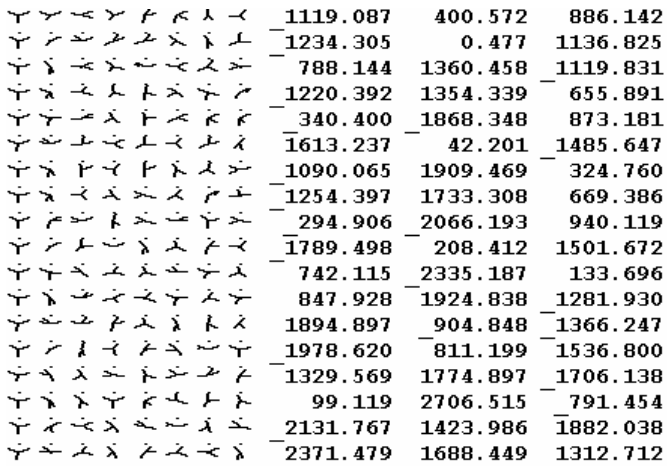
Fig.4 *Another, simpler version of hms*

2.1 Diordinate list from coordinate list

2.1.1 In case of minimal spanning tree

```
dj18=:hms{~>,&.>10#.&.>|:&.>-&.>(8#_10)&#:&.>;/kj18=:18{.XYZe
```

```
dj18,.kj18=:10j3":1000%~kj18
```



```
_1119.087   400.572   886.142
_1234.305     0.477  1136.825
 788.144  1360.458 _1119.831
_1220.392  1354.339   655.891
 _340.400 _1868.348   873.181
1613.237    42.201 _1485.647
_1090.065  1909.469   324.760
_1254.397  1733.308   669.386
 _294.906 _2066.193   940.119
_1789.498   208.412  1501.672
 742.115 _2335.187   133.696
 847.928 _1924.838 _1281.930
1894.897  _904.848 _1366.247
_1978.620   811.199  1536.800
1329.569  1774.897 _1706.138
  99.119  2706.515  _791.454
_2131.767  1423.986  1882.038
_2371.479  1688.449  1312.712
```

2.2 Coordinate list from diordnate list (comparison)

```
kj18-:10j3":1000%~->(8#_10)&#.&.>|:&.>(3#10)&#:&.>;/hms i.dj18
1  NB. 1: match, 0: not match )
```

2.3 Insights into the details

```
-&.>(8#_10)&#:&.>1000*&.>;/kj18   NB. Conversion to _10 based NEGA_decimal system
```

0 2 9 2 1 1 2 7	0 2 8 4 6 3 0 5	0 0 8 2 9 9 5 6	0 2 8 2 0 4 1 2	0 1 7 4 0 4 0 0	1 9 7 9 4 8 4 3	0 1 0 9 0 1
0 0 4 0 1 5 8 8	0 0 0 0 1 6 8 3	1 9 4 4 1 6 6 2	1 9 4 5 5 7 4 1	0 2 2 7 2 4 6 8	0 0 1 6 3 8 1 9	1 8 0 9 0 6
0 0 9 2 7 9 5 8	1 9 2 7 7 2 3 5	0 2 9 2 1 9 7 1	0 0 7 5 6 1 0 9	0 0 9 3 4 9 9 9	0 2 6 9 5 7 6 7	0 0 4 8 5 3

```
|:&.>-&.>(8#_10)&#:&.>;/kj18                NB.  Read by rows:  COORDINATES --- read by columns:  DIORDINATES
```

0 0 0	0 0 1	0 1 0	0 1 0	0 0 0	1 0 0	0 1 0	0 1 0	0 0 1	0 0 1	0 0 0	0 1 0	1 0 0	0 0 1	1 1 0	0 1 0	0 1 1	1 0 0
2 0 0	2 0 9	0 9 2	2 9 0	1 2 0	9 0 2	1 8 0	2 9 0	1 2 9	2 0 9	0 3 0	0 8 2	9 1 2	2 0 9	9 9 2	0 8 1	3 9 9	8 2 2
9 4 9	8 0 2	8 4 9	8 4 7	7 2 9	7 1 6	0 0 4	8 8 7	7 0 0	3 2 5	8 7 2	9 0 8	9 1 7	1 9 6	4 8 3	1 7 2	9 5 9	4 4 7
2 0 2	4 0 7	2 4 2	2 5 5	4 7 3	9 6 9	9 9 8	6 7 3	0 7 6	9 0 0	6 4 7	6 8 9	1 1 7	8 9 7	7 3 1	0 0 0	4 8 2	3 9 2
1 1 7	6 1 7	9 1 1	0 5 6	0 2 4	4 3 5	0 0 5	6 4 0	6 4 1	1 9 2	3 5 4	8 5 8	5 6 4	2 2 7	0 5 4	0 7 9	9 4 2	2 2 8
1 5 9	3 6 2	9 6 9	4 7 1	4 4 9	8 8 7	1 6 3	4 7 7	9 2 9	5 6 4	9 2 3	1 2 0	1 9 3	7 8 2	5 1 2	9 5 5	8 1 0	6 5 8
2 8 5	0 8 3	5 6 7	1 4 0	0 6 9	4 1 6	4 7 6	1 1 9	0 1 2	1 2 8	2 2 0	3 4 7	0 6 6	8 0 0	7 0 7	2 2 5	4 9 4	8 6 9
7 8 8	5 3 5	6 2 1	2 1 9	0 8 9	3 9 7	5 1 0	7 2 4	6 3 1	8 8 8	5 7 4	2 2 0	3 8 7	0 1 0	1 3 8	1 5 4	7 4 2	1 9 2

```
10#.&.>|:&.>-&.>(8#_10)&#:&.>;/kj18        NB.  Read by rows: DIORDINATES --- read by columns:  COORDINATES
```

0 200 949 202 117 159 285 788	1 209 802 407 617 362 83 535	10 92 849 242 911 969 567 621	10 290 847 255 56 4

Paralleling of XYZ coordinate-triplets

3. Comparisons

3.1 In case of minimal spanning tree (each point has own starting point)

```
dj18,.(":hms i.dj18),.(18 3$' '),.kj18
```

diordinates (spatial digits serial numbers) coordinates

(NEGA_decimal)	...	km	hm	...	m	dm	cm	mm	X	Y	Z
	000	200	949	202	117	159	285	788	_1119.087	400.572	886.142
	001	209	802	407	617	362	083	535	_1234.305	0.477	1136.825
	010	092	849	242	911	969	567	621	788.144	1360.458	_1119.831
	010	290	847	255	056	471	140	219	_1220.392	1354.339	655.891
	000	120	729	473	024	449	069	089	_340.400	_1868.348	873.181
	100	902	716	969	435	887	416	397	1613.237	42.201	_1485.647
	010	180	004	998	005	163	476	510	_1090.065	1909.469	324.760
	010	290	887	673	640	477	119	724	_1254.397	1733.308	669.386
	001	129	700	076	641	929	012	631	_294.906	_2066.193	940.119
	001	209	325	900	192	564	128	888	_1789.498	208.412	1501.672
	000	030	872	647	354	923	220	574	742.115	_2335.187	133.696
	010	082	908	689	858	120	347	220	847.928	1924.838	_1281.930
	100	912	917	117	564	193	066	387	1894.897	_904.848	_1366.247
	001	209	196	897	227	782	800	010	_1978.620	811.199	1536.800
	110	992	483	731	054	512	707	138	1329.569	1774.897	_1706.138
	010	081	172	000	079	955	225	154	99.119	2706.515	_791.454
	011	399	959	482	942	810	494	742	_2131.767	1423.986	1882.038
	100	822	447	392	228	658	869	192	2371.479	_1688.449	_1312.712

3.2 In case of coordinates relative to the same origo

diordinates (spatial digits serial numbers) coordinates

Mm	...	...	km	hm	...	m	dm	cm	mm	X	Y	Z
414	235	355	904	556	912	368	113	493	600	4239593.146	1350516.190	4554628.330
414	235	355	576	393	792	201	601	790	727	4235372.677	1357990.092	4556321.107
414	235	384	608	237	897	407	272	123	830	4236284.218	1380390.723	4548777.230
414	235	365	355	736	355	436	810	155	940	4233734.819	1365353.154	4555656.050
414	235	275	937	422	940	243	372	364	352	4229492.333	1373244.765	4557203.242
414	235	365	930	548	642	453	472	358	888	4239564.438	1363445.758	4550823.288
414	235	375	531	871	934	134	793	419	815	4235891.748	1373733.911	4551144.395
414	235	295	406	944	087	207	728	816	612	4224902.786	1390480.211	4556477.862
414	235	295	822	632	323	591	920	748	834	4228635.978	1392329.243	4552231.084
414	235	285	716	069	779	955	386	406	886	4227079.348	1381675.808	4556995.666
414	235	195	978	739	714	357	119	650	041	4219773.160	1397315.154	4558947.901
414	235	385	151	763	467	817	906	139	848	4231748.918	1385661.034	4551377.698
414	235	295	790	758	970	330	907	448	519	4227793.945	1399573.041	4550800.789
414	245	205	065	606	752	531	691	395	874	4220675.638	1406053.997	4555621.154
414	245	115	863	689	493	738	578	477	137	4218647.541	1416893.773	4553938.877
414	245	215	400	615	141	000	629	748	311	4224610.673	1410140.241	4550510.981
314	957	505	149	663	462	107	717	093	547	3951641.705	1504660.194	4759327.737
314	947	595	876	181	244	963	357	305	731	3958129.337	1497846.503	4756143.751
314	957	425	925	585	380	456	004	181	556	3949534.015	1522885.085	4755506.416
314	957	435	436	205	980	464	083	532	427	3944294.054	1533086.832	4756504.327
314	957	435	395	322	775	164	573	456	277	3943371.542	1539276.757	4755254.367
314	957	455	033	083	322	751	754	478	090	3940037.740	1553825.579	4753321.480
314	957	345	838	323	066	168	027	785	919	3938301.079	1543266.281	4758368.759
414	235	217	656	567	469	914	526	344	432	4226549.534	1315661.243	4576794.642
414	235	227	266	988	667	912	480	304	900	4222969.439	1326861.800	4576872.040
414	235	317	123	094	364	008	497	668	469	4231030.464	1312960.966	4573448.789
414	235	227	951	027	295	489	039	365	512	4229024.035	1325298.361	4571759.952
414	235	326	594	792	432	328	112	967	920	4235743.199	1329932.162	4564228.270
414	235	316	696	554	337	382	177	042	034	4236533.100	1319538.743	4566472.724
414	235	247	073	046	132	319	900	553	485	4220013.954	1347431.058	4573629.035
414	235	147	509	835	345	519	808	495	850	4215835.848	1340341.095	4579559.850
414	235	237	244	798	230	720	128	920	963	4222727.199	1334932.226	4574800.803

Contribution with *Erik Papp*