stop the BS Documentation

Release 0.1.0

julien tayon

January 22, 2013

CONTENTS

A friend once told me **a solution without a problem is just plain bullshit**

So, let's explain why we need an exact matching operator in programming language instead of a precise one, and **especially**, what problems it would solve for real humans.

Then, let's explain in further details how we have the hints for a solution yet, and why we have already difficulties solving it.

We will then try to reprogram your brain with some back to basics idea, and we will finally propose an incomplete coded solution.

Warning: All typos, orthographs mistakes, unsustained level of language are just a patented ingenious method of watermarking. © 2012 JT.

CHAPTER

ONE

PRECISE AND EXACTS RESULTS ARE NOT THE SAME

A precise result of π is *3.141592653589793*, an exact result is *3.14 +- .005*, the truth is both these results are not safe for use (but I keep the development for a later chapter).

Precision is impressive, but in the real world, no results are perfectly measured. An exact result is something that is safe to work with because it gives you a range of confidence. A precise result is a good enough result, but it may bit you later.

If you have a value of π and you would check it in a programming language you would not write:

```
if pi == 3.141592653589793:
    print("yes this is pi")
```

but as with any floats you'd write:

```
epsilon=10**-10
if abs(pi - 3.1415926535) < epsilon:
    print("I am sufficiently close enough to pi")
```

1.1 Let's think of more concrete problems

1.1.1 The «almost equal» a priori version

Imagine you are a policeman searching a criminal or a marketer fitting a profile. Your database is so large, and -humans changing their biometrics during life- you know that your very nice 10 digit precision metrics are false.

You already know that even though you have the good size, weigth, eye colour, and other metrics, exact matches done with the equal operator will fail.

What you desire is something like:

```
profile=Human(
    # in IS units
    sex= 42, #
    size=2.1,
    weight=69.12343234,
    eye_color=rgb(00,200,20),
    ...
)
accetable_variation=Human(
    sex=0,
    size=-.1, # the criminal is more than 20 years old, (s)he cannot grow
    weigth= interval(+10, -20),
    eye_color=0.01,
    ...
).incertitude()


for suspect in database:
    distance = abs(suspect - profile)
    if distance > accetable_variation:
        continue
    # abs is the norm for the complex so why not
    # try to be consistent?
    print(
        "%f%% match %r found " % (
            100*(1.0-( abs(suspect)/abs(profile) )),
            suspect
        )
    )
```

An alternate notation might be:

```
for suspect in dabatabe
    if not suspect ~= profile given accetable_variation:
        # where ~= would be the almost equal operator
        # given would be a reserved key word
        continue
    print "%r is a %f far from profile" % (
        suspect,
        abs(suspect - profile)
    )
```

Warning: yes I totally assume my devilish non sensical confusing choice of ~= close to =~ that is regexp match operator in Perl.
But it is far less horrible than my initial choice of [ ] operator for denoting the interval of confidence (now **given**).
And, yes I agree no people in their right mind would choose this notation.
However, I already know I won't need this notation. But I do take the right to have fun.

1.1.2 Computer lie!

Just ask a computer what he thinks of::

```
>>> .1 * .1
0.010000000000000002
```

Well, it is no big deal. As long a you don't chain operations, where errors will propagate. It is cool as long as your problem is not sensitive to initial conditions.

Just remember banking, trading, commercial fees are computed with this flaw.

Sometimes you wish you have the exact results when a human being's life relies on it.

1.1.3 Conclusion

As I know how to do it, and since it is possible I propose to think of an almost equal operator

CHAPTER

TWO

GOOD NEWS, IT PARTIALLY ALREADY EXISTS

I had this idea while @pyconfr listening to Aymeric Augustin presenting the pits in datetime

2.1 Debugging the developer's brain

Date time handling whatever the langage is, reveals one of the biggest weakness in human brain: **not being able to see the difference between a position and an interval**

Of course datetimes are interval of time elapsed since an arbitrary origin.

It is just the representations that are differents.

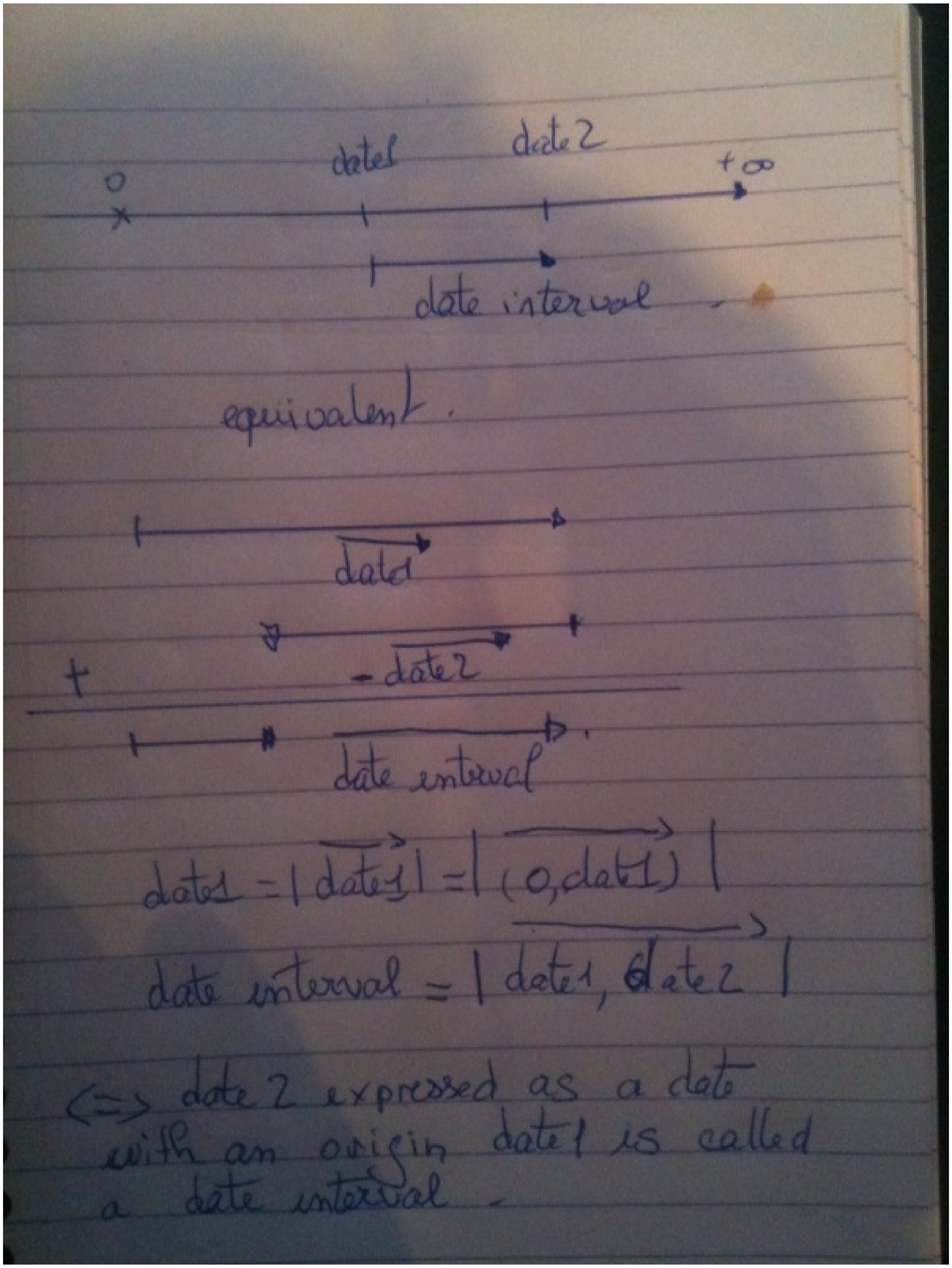
Times and dates are full of caveats:

- leaping hours in CEST;
- absolute vs local time;
- Time Zone;
- calendar time vs usual time (1 day duration, vs an event occuring the next day);
- and a lot of confusing stuffs ...

For dates implemented as seconds elapsed since 1970 in UTC you can figure that the unit of the date is seconds, and the duration is seconds. The only difference is the origin.

Why did we need to abstract datetime so much that we had to introduce the datetime interval?

A datetime interval is just a datetime



$\text{date1} = |\overrightarrow{\text{date1}}| = |\overrightarrow{(0, \text{date1})}|$

$\text{date interval} = |\overrightarrow{\text{date1}, \text{date2}}|$

<=> date2 expressed as a date with an origin date1 is called a date interval.

2.2 Because there is a bug in the perception in the brain of developers

I don’t know where it is, I only know that datetime is sufficiently well designed that it lowers the bugs in date handling. They achieve this by making believe that datetime and intervals are somewhat different and raising exception when you try to add a date to a date and an interval to an interval.

I just hate soft science, so I leave to pseudo scientific from a psychology/pedagogy department the care to develop this uninteresting part.

But, I see these datetime problems not as a bug in computers but an awfull big bug in education. It is probably located near the bug that causes the the strings/bytes array/unicode/utf8 confusion.

If I tell you I am not sensitive to this bug, you know I lie. I am just a little less sensitive to this confusion than average. Still I sometimes too make mistakes on this topic.

2.3 Conclusion

Since I think there is a bug in the concepts involved my answer will not be coding.

It is first understanding what works and how we could shatter this barrier.

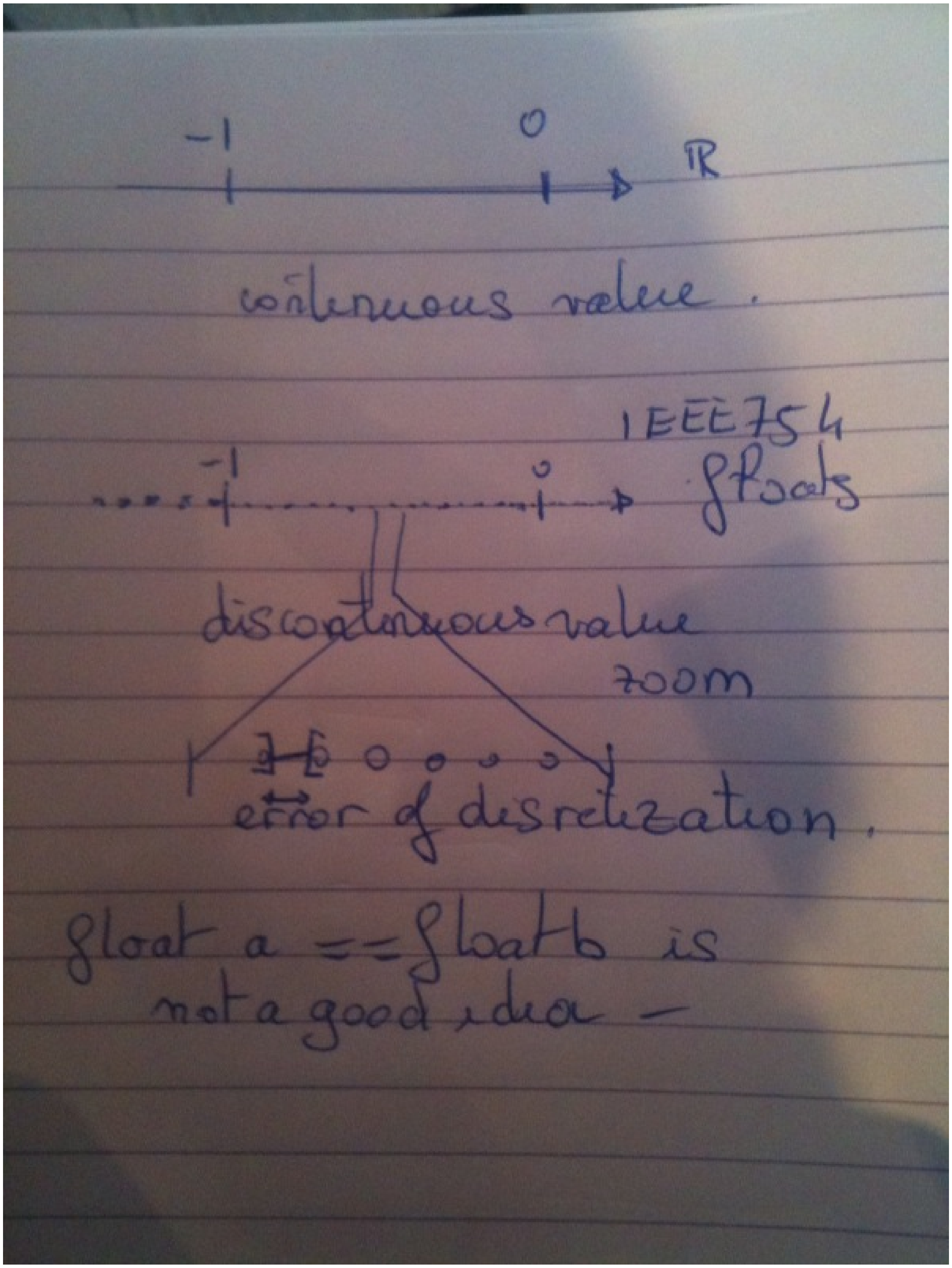
CHAPTER

THREE

ARE THEY NATURAL APIS?

3.1 Float and exact matching

Well our intuition of number is the one from math. Computer numbers are not the same. I know this is a trivia, but I see all the time developers ignoring this:



Since computer IEEE754 is only exact an precise for sums of power of 1/2, thus exact matching is made this way:

```
>>> good_enough=10e-5
>>> wanted = 0.01
>>> .1 * .1
>>> 0.010000000000000002
>>> abs( ( .1*.1 ) - wanted ) < good_enough
True
```

Okay nice. This is our first example.

3.2 Complex or the class 2D Point

So, in stdlib documentation for named tuple we propose writing a useless class: the 2D point.

While in python we a base type that having all the needed feature: the complex type.

Given you call *x* the *real part*, and *y* the *imaginary part*, then you have a 2D point.

Let's restate the obvious::

```
    >>> a=complex(1,2)
```

This is both a point declaration and a vector whose origin is complex(0,0).

Now, let's shrink the vector by an homothetia a 100 scale::

```
    >>> a=.1*(.1*a)
    >>> a
    (0.010000000000000002+0.020000000000000004j)
    >>> wanted=.01+.02j
    >>> tolerable_margin=10e-5
    >>> abs(wanted-a)<tolerable_margin
    True
```

It works!

By the way, what is **abs**?

3.2.1 Norm of a vector

What does abs in python?:

```
Return the absolute value of a number. The argument may be a plain or long integer or a floating poi
```

Well it is basically the distance from the origin 0.

So basically the preivous python code is

is the distance between a and wanted comprised in a 10e-5 radius.

3.2.2 And what does a - wanted means?

So with the datetime example we now understand minus (__sub__) is in fact an **addition** of the **reversed** value of *wanted* to *a*.

wanted and *a* are de facto vectors. Thus, **addition is put simply a translation**.

3.2.3 What is a multiplication?

A multiplication is an homothetia from a vector.

3.3 Summarizing up

For the following types:

- complex;
- floats;
- datetime we have a natural API.

That is based on:

- by combining two vectors, we always a vector;
- interval are vectors;
- abs transforms a vector into a scalar called norm or distance.

Thus ::

```
>>> abs( a-b ) < margin
## is the same as
>>> a == b ± margin
```

And since computer makes mistakes in some cases (by rounding) it is therefore the closest answer we have to the mathematical truth called equality.

Next question is does it works for everything?

CHAPTER

FOUR

DESIGNING METRIC ORIENTED OBJECTS

4.1 Premisce

I make the following axioms:

- an object has properties;
- properties are axis;
- the purpose of object metrics is to know if another object is close to another one or;
- to know if an object is oriented in the same direction.

Since an attribute itself is an object, we have to consider the most simple case:

4.2 An object made of type of base

4.2.1 Integers, float, complex

The L1 Euclidian distance suffices from value to value.

Let's see if we have the interval with orientation this way?

Given a Point A and B with properties x,y,z can we easily find the Euclidean results?::

```
>>> from archery.bow import Daikyu
>>> class Point(Daikyu):
>>>     def __abs__(self):
>>>         # L2 distance
>>>         return sum(map(lambda s:s*s,self.value()))**.5
```

CHAPTER

FIVE

KNOWN LIMITATIONS, WHY IT DOES NOT ALWAYS WORKS?

Imagine you have a speed and you add speed because you want to know the speed of a bullet fired at **.5*c** in a space ship travelling at **.9*c**.

What happens?

Well, you have a problem by finding a speed greater than the speed of light in a fixed referential.

The big question is: **is the bug located in the composition of the vectors or located in the norm?**.

We have a very fast answer::

```
>>> ### let's compose the speed and see
>>> bullet.speed= .5 # expressed in ratio of c
>>> spaceship.speed = .9 # expressed in ratio of c
>>> spaceship.speed + bullet.speed
```

The problem should be solved at composition level. We have left euclidian geometry, and we know work in a space were transaltion of vector are bound by Lorentz transformation.

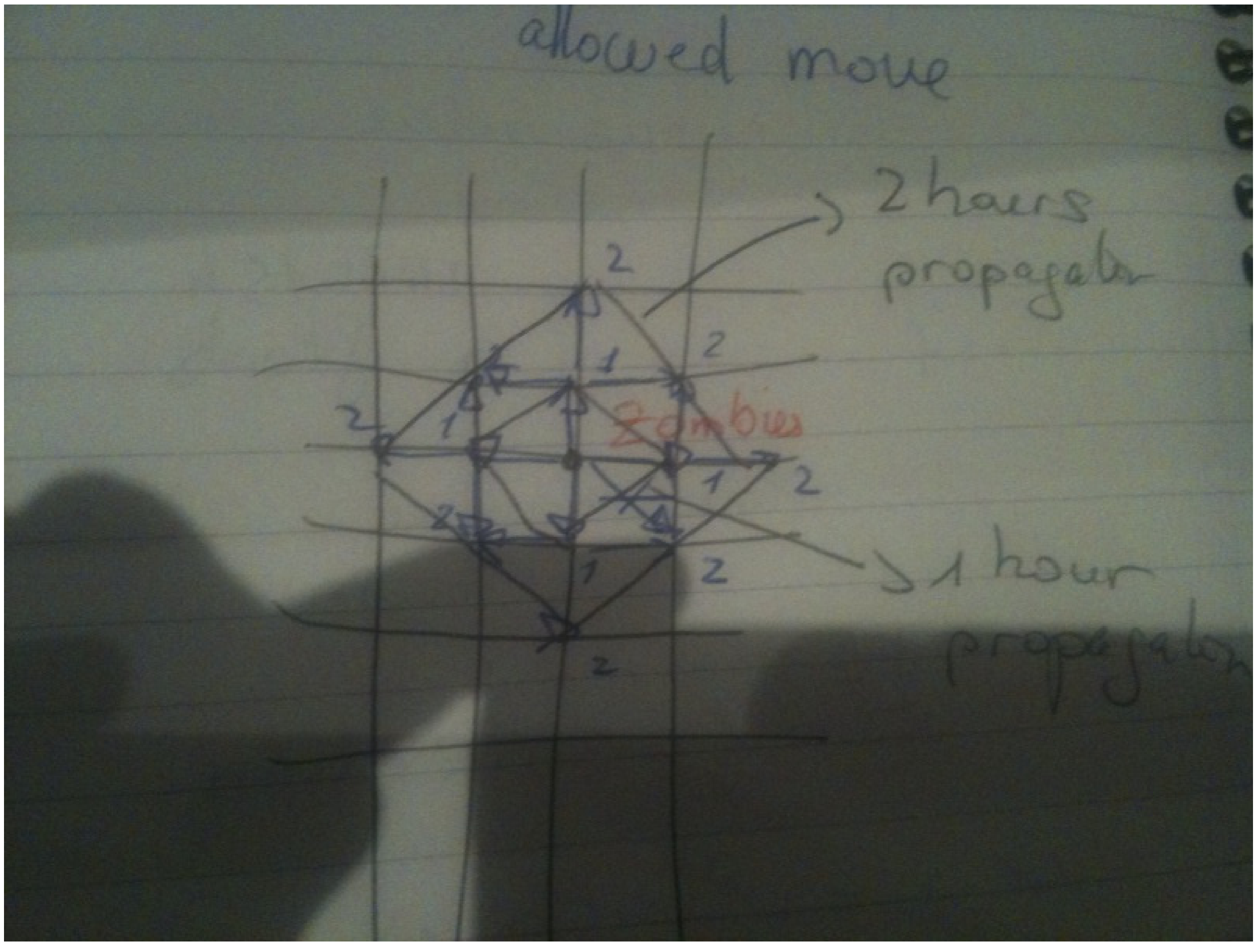
5.1 So, the limitation is in the substraction/addition?

Well not.

Well, this example not as funny as measuring the distance at which you place the army in case of a zombie invasion in a square city (looking like a matrix).

Given a speed of 1 block per hour (zombies are slow), we want to where to post our valiant anti-zombie task force. We want to be able to test if a given human will be in zombie range in n hours.

Given that I am lazy person, I only make a nice draw to illustrate the problem:



And I give the solution. In this case: you change **abs** for any given vector.

Instead of::

```
>>> abs = lambda x,y: ( x**2 + y**2 ) ** .5
```

You write::

```
>>> abs = lambda x,y:abs(x) + abs(y)
```

5.2 I am really speaking of zombies only?

Well, no.

If you push hard enough the precision of your computer, your reachable values are discretized, thus, it is also how your 2D vector will look.

Have you noticed that the stuff I draw are circles.

These are **squares**.

Thus the value of **Pi** is changed (when a circle is a square there are odds that pi is not an irrational number anymore), so is the distance logic.

Most time, you don't care. I guess only 1 out of 100 000 developers will be bitten by this problem in their whole life.

So cool down, it is just an illustration of computers limitation.

CHAPTER

SIX

WHAT IS A SYMMETRY AND WHY DOES IT MATTERS?

The point is you should never believe a teacher :)

The question asked by symmetry is: given a peculiar way of moving in space constricted by geometrical operations, if I do operation A then operation B on a given place (topos) then will the result be the same?

Ex: if I take a sphere in Euclidian geometry and then rotate it by its center of an angle alpha will the new topos map the other one?

Yes, then a sphere accetps a symmetry of a rotation of any angle in any direction.

Is it true?

Well, let's imagine I have a 2D clock in a 2D space. Now I do a mirror symmetry on any diameter, the mirrored clock seems to be the same? Let's watch the clock ticking. Buh it is ticking backwards.

Just also try saying that an aquarium opened on one the top can be rotated of 1 quarter towards the ground :)

The first example works only because another implicit symmetry is added, the place of the points don't matter. It is a permutation symmetry.

However let's get back to business. So let me assert my own definition of symmetry.

Note: A figure is invariant by its symmetries if it overlaps itself fully given any application of any 2 symmetries in any given order.

There can be no less of 2 symmetries on a system.

A complete set of symmetries on a problem is called a base.

There are as much symmetries in a system there are degrees of liberty.

The measure of symmetries is called a commutator denoted [ ] Saying that A and B commute is the same as saying

- [A, B] = 0
- or A(B(topos))=B(A(topos))
- measure of symmetry is [A, B] = partial(A,B) - partial(B,A)

A and B operates on place, and [A, B] is also an operator.

A singularity is a place S such as you always have [A,B](S)=S

6.1 Hey lad, you are nice, but why should I care of symmetry in computer science?

Saying that a place Ta is symmetric to a place Tb is the same as saying they are to deduce Ta from Tb.

CHAPTER

SEVEN

ALL YOUR BASES ARE BELONG TO US

A base is a set on uncorrelated dimensions.

What are uncorrelated dimensions?

We ain't doing SiFi so let's begin with the first question:

7.1 What is an axis?

An axis is a right a on which you have corrdinates. Every points on the axis can be compared and have therefore a distance to the origin and between themselves.

7.1.1 Weired Axis: incomplete knowledge and order

It may seems odd, but in computer all day's life, we have axis on which we have no idea where to place our «points» but for which we can compute the distance.

The best example that comes in mind are **strings**.

We can have two strings *s1* and *s2* which we cannot plot on axis, we can't always tell if *s1* is greater than *s2*, but we can tell the distance between *s1* and *s2*.

How is it possible?

if *s1* is a subset of *s2* we can say *s1 < s2* however if *len(s1) == len(s2)* and s1 and s2 are made of different characters we are stuck. That's were we use Levenshtein distance

So to sum up there exists problem for which in computer science we :

- cannot say where a point is located;
- cannot say the direction of a segment on to point of the axis;
- can measure the norm between two points on the axis.

So given 3 strings A, B, C on the axis of the string we cannot say if *A > B > C*, **but** we can say if **|AB| > |AC| or |AC| > |BC|**

Of course, strings are not really points, they are sequences of points, they are segments. But what I say, is given we want to measure the distance between two points: **we don't care it behaves just the same**.

Well, let's call it **geometrical duck typing** : if you can get the absolute distance between two entities of same type, we have far enough for the purpose of comparing. I hear some people jump off their chair: I have the norm, I don't have the direction.

Héhé... I know, but I don't need it.

7.2 Are there any real infinite dimension spaces useful in real life out there?

Well, google is mastering the art of projection in multi-dimensional space, so I dare say yes.

7.2.1 Text indexation is a natural indefinite dimension problem

On this one, I'll work on a simplified problem of the textual indexation geometry.

What is Google's job:

- the world is full of documents;
- you want all documents relative to a word **Q**.

So in terms of geometry you say that there exist a relationship between a documents made of N words Wn to one word Q. Since Wn and Q are of the same type we call this a projection. Since it can be more or less revelant, we see there is a logic of measure.

7.2.2 Why is the word counter problem the most used problem for parallelism (map/reduce)?

Well, since working on sequence of words might be too hard here is one of the way people represent text:

- you tokenize the text in plain string separated by their boundaries;
- you transform each words in its invariant natural form (go, gone, going becomes go(verb));
- you get rid of useless words (articles and maybe preposition);
- you count each invariant form.

So you have built an incomplete vector of word. Each invariant form is a dimension; Each count is a scalar aka a measure/distance/coordinate on the axis of the invariant form.

Why is the vector incomplete? Unless you are a dictionnary you use less word in your text than the whole available words in the language.

7.2.3 Building the canonical text corresponding to the keyword Q

This a secret of fabrication, but it usually begins with human beings tagging various text the more accurately possible with keywords. Than afterwords, with a statisticall analysis, for each invariant form of a word, you'll decide if the presence of the word is correlated to the keyword.

Once you have the list of the Words, you know just do a frequential analysis, and for a given keyword **Q** you have a canonical word counter of invariant form **Vq**.

A keyword Q is now associated with an incomplete vector Vq pointing in a certain direction

7.2.4 Searching the text the closest to your query

For each **Vti** which represents the Vector of a text **t** expressed as a word counter of its invariant form, you:

- you normalize each **Vti**;
- you project each **Vti** on **Vq** with a classical dot product*;

- you give every results of the **Vti** that are the closest to **Vq** given a confidence margin.

Note: **dot product on words**: pseudo code:

```
let dot be zero
let normV be zero
let normQ be zero
for each wordsi frequency Fv,Fq present in Vti and Vq:
   multiply frequence of Fv and Fq
   add it to dot
normV is absolute value of distance from 0 of Vti
normQ is absolute value of distance from 0 of Vq

return dot / (normV * normQ)
```

Now, since Vti and Vq are normalized you have a scalar *Confidence* called **c** which varies from 0 to 1 which says how much *Vti* is pointing in direction *Vq*. It is called a cosine similarity.

The dot product of Vq and Vti is in fact a projection of Vti on Vq. If the text Vt triggers the keyword Vq then it means the vector of the text (Vt) points in the same direction as Vq.

So Vq dot Vt is how much times Vt «speaks» of Vq. By dividing by the norm of Vt we make an homothetia that answers another question: if we Vq and Vt were the same size, what is the ratio of **Vt** represents compared to **Vq**.

The magic in textual indexation is a 60 words can be as relevant as 6000 words text. A definition of *serendipity* (60 words) is has meaningfull has a thesis on serendipity. But a text of 10 000 words long having once the keyword vector triggered is useless since S=-K.ln(Omega) you lose informations.

The problem with distance and measures, is that you can use a lot of other metrics. For instance you can weight the words with its presence in anything that semantically denotes a stress (title, emphasis, URL).

The norm is usually calculated with L2 = sqrt(Sum((Xi)^2)). But is flawed.

CHAPTER

EIGHT

SOME INTERESTING PROPERTIES OF MANHATTAN GEOMETRY

Manhattan geometry, which is a discrete geometry whose axioms are:

- the universe is discretized in quantum of distance;
- 2 points A & B defines more than one segments;
- a circle is twitesd square, but a square is still a square;
- something about the sum of the angle in a triangle;
- there is an infinite number of rights coming through a point, but it is an infinity less than in euclidean geometry;
- parallels may share an infinity of subsegment that are periodics;
- the generalized distance is L1 for whatever dimension you use;

Note: Given:

- dimension varying from 0 to i;
- X,Y denoting of coordinate Union(| xi >) Union(| yi >)

Then distance is : *D(x,y)=Sum(0,i)(abs(yi-xi))*

For the following let us denote C(O,r,x) a circle C of center O(irigin) with a radius r, with a level of discretization x. Default being 1.

The level of discretization indicates how many division I accept in the unit vector.

8.1 Circonference of a circle

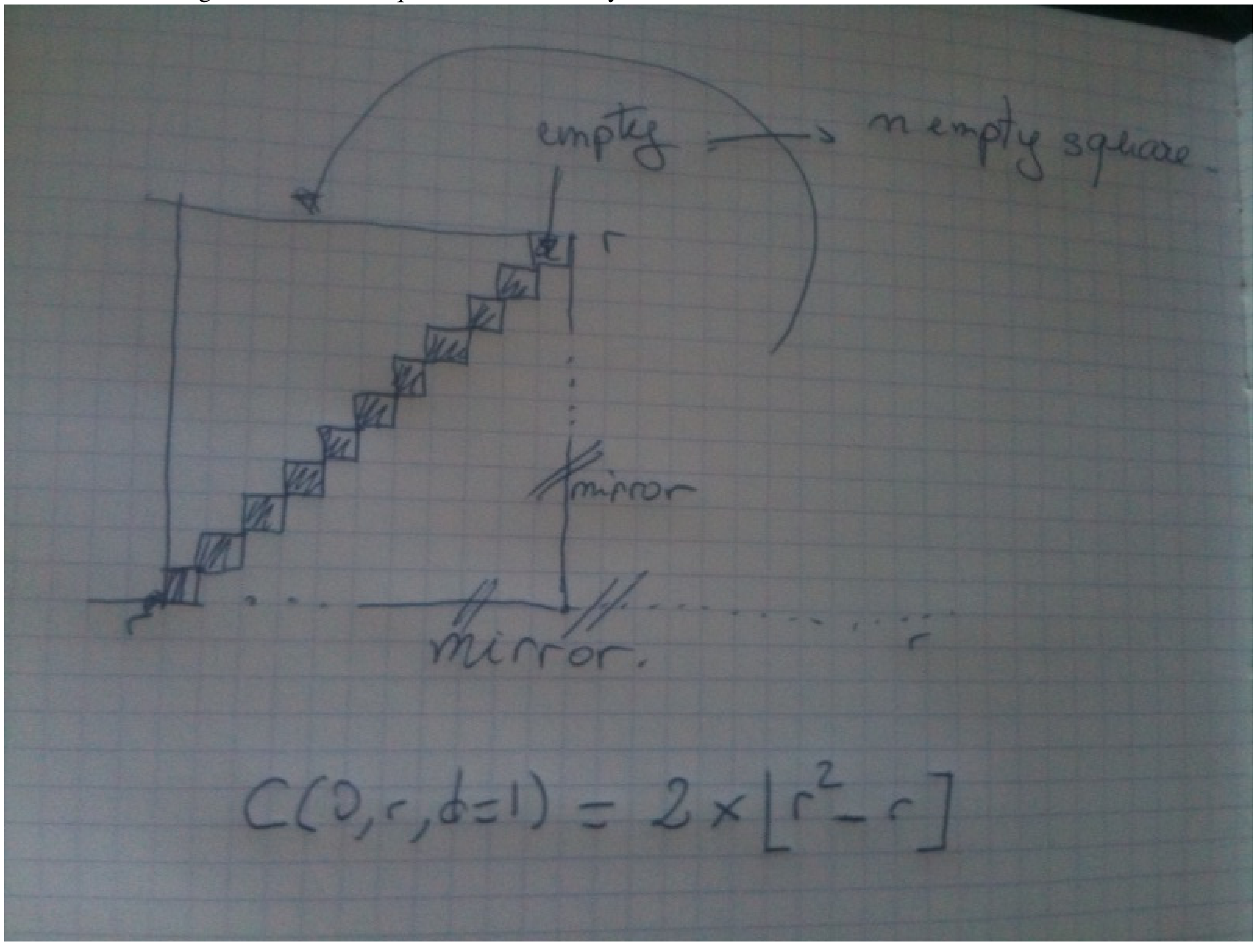
Note: Circonference(C(O,r,d)=8 x r

Whatever the discretization level is Circonference will always be the same o/

8.2 Area of circle

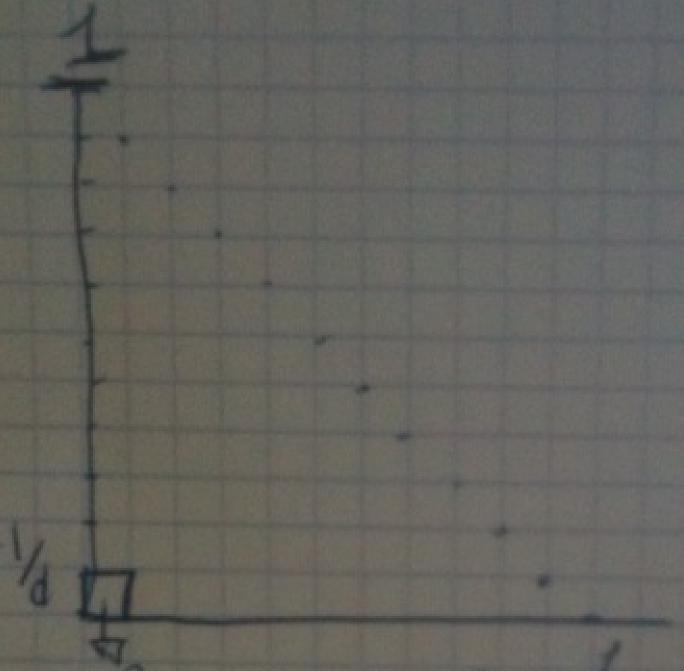
This one is quite funnier, so let'es draw some pictures :)

The area is measured by the number of square contained within the envelope of a Circle C(O,r,d) such as any edge is a vertex which length is inferior or equal to r measured by L1 from O.



Which strangealy enough tells us that the area of C(0,1,1) is 0

Now let's study the case where the discretization level is not 1 for unit Circle: C(O,1,d=x).



a_x — area of $\frac{1}{d} \cdot \frac{1}{d} = \frac{1}{d^2}$.

$$S\, C(0,1,d) = 2 \times d(d-1) \cdot a_x$$
$$= 2 \cdot d(d-1) \cdot \frac{1}{d^2}$$
$$= 2 - \frac{2}{d} \qquad \lim_{d \to \infty} \frac{2}{d} = 0.$$
$$= 2.$$

Note: The surface of the unit circle will tend towards 2. **A(0,1,d)=2(1-1/d)**

It is trivial to demonstrate that **A(0,r,d=D)=r.(A(0,1d=D)** thus

Note: A(0,r,d)=r.2.(1-1/d)

A(0,r,d=>inf)=2xr

8.3 Nth dimensional probem of the volume

This one is boring easy so let's just state the results:

Note: given n dimension, the volume V of dimemsion n is

Vn(0,r,d=>inf)=2^(n-1) . r

Plus if the Area of dimension n is denoted A

An(0,r,d) > Vn(0nr,d) whatever the dimension is

So in three dimensions for instance, the surface of the sphere will have a measure in measure^n inferior to its surface in measure^(n-1), thus the ratio V/A < 1. In a Manhattan driven geometry big mammals would not exists, since the reason to be of large mass is diminishing this ratio so that you radiate less heat.

Since computers are Manhattan driven geometry, thus some problem might give false results.

8.4 Conclusion

Geometry are abstract yet intuitive constructions. What may looks like an intellectual wanking is interesting: Manhattan geometry is a geometry fitted for problems where distance between to points can be expressed as quantum of differences. Plus since it is pretty intuitive, it is easier to handle than nth dimension generalized euclidian geometry.

Plus since is does not rely on real numbers but on on natural integers, measures are peculiarly well suited for computers. Since we have order relations it permits what we love to do with computers:

- comparing;
- sorting.

CHAPTER

NINE

THE OBVIOUS

9.1 Geometry

In Greek, *geo* means the place, and *metrein* means measuring.

A geometry is a set of axioms (accepted truth) and transformations that describes the legal movement possible in a place.

Any changes in the axioms changes the geometry. For example change one or two axioms of euclidian geometry gives you the geometry used by Einstein.

By changing a place in another you can compare them, by comparing them you measure.

9.2 Transformations, projections and symmetries

There are no projections in one dimension In order to get dimensions you have to give your geometry new rules that's called adding dimensions. Normally this operation is recursive. So once you add a dimension you can add as much dimension that you want.

Transformations and projections are operations that transform a place in antoher place. If it there is a reciprocal non deformative operation it is a transformation if you lose information in the process it is a projection.

Projection are destructive operations. Transformations are lossless operations in terms of operations. Projections can reduce the dimensions of the studied place (topos in greek).

Symmetries are transformations that given a set of rules leaves certain properties as-is.

9.3 Measure

Measuring is the magic that transforms geometrical rules in number.

It comes with three items that must consistently be defined :

- an algebrical distance;
- constrictor (which limits the place);
- a comparison operator (reductor or projection):

The algebrical distance must be:

- defined (non infinite for any topos);

- normed (there must be a unite distance);
- positive;

The reductor is the geometrical counterpart of the distance and define how you scale from a unit distance to a given topos. It answers to the question: given my multi-informational space how do I obtain a 1D problem? It may be a projection.

A projection is an operation yielding linear results, whereas a reductor yields a non linear result.

A constrictor is tightly coupled with the two items it defines the accepted transformation. From this will come the distance you use.

Example

A Manhathan geometry implies you move in a squarish universe (constrictor). Then the reductor is counting all the vertices you have to walk in this universe to go from place (topos) A to B using the «shortest path», and finally the distance is Sum(abs(vertices)). Defining the shortest path is equivalent to define two places known as :

- a unit circle;
- a unit square;

CHAPTER

TEN

LESS OBVIOUS

10.1 Non Euclidian geometries

A non euclidian geometry appears every time you play with axioms, accepted transformations or measures.

10.2 Everything is relative

Since numbers, thus distance boils down to comparing to unit places, a measure is **always** a comparison with unit measure. There are no absolute numbers. That might be the reason I hate numerologists and astrologists. People taking for certain the outcome of numbers when numbers are only ratio to a unit place is like someone telling me: 8 is a great number after I have been beaten to the pulp once and telling me to get beaten 7 times more (I know chinese takes 8 for a lucky number).

10.3 Corrolary: Point don't exists

A point is always a segment (the smallest place possible) given an *arbitrary* origin.

Example: the date of today is the number elpased since the arbitrary so called birth of Christ. Well, he was born sooner since he add is christmas gift on a 25th of december just very near to the proto christian religion begining of the year (21st of december) when the earth begins its trip back from the furtherst point to the sun resulting in the lenghtening of the day in the northen part of the earth.

Dear christians if earth is flat explain to me why Australian babes are in bikini when we great the new year in polar clothes in Montréal? To explain this come up with a good geometry that bits Terry Pratchett most hallucinated novel, it will be entertaining.

10.4 Craving for a dynametric

I believe time should be included in geometry as a side effect of projections. You have time once you cannot go back. Thus any operation that given a place has more than one predecessor defines a time (irreversilbility).

A time line is a serie of projections or reductor that don't let the invariant of the last projections be the invariant of the next projection. The serie of projection can be continuous. It could be called an expander or a cruncher. Pick your name.

10.5 Reality and virtuality

A geometry is virtual and reality is boring. However, what is described in the virtual world might come true given either an awful lot of good premisces and hard work to keep things consistent with observations or a short sight. Most people prefer short sight by lazyness.