“Reality is nothing but a mathematical structure, literally”.

In This letter I shall derive the laws of nature from a simple mathematical system that forces itself upon us because it is the only system that lead to a dynamical universe. Such an idea has long been suspected to varying degrees by different scientists and philosophers from past to present. As evidence mounted stronger, advocates albeit few have argued for the cause. Max Tegmark of MIT has been the leading proponent of such hypothesis.

Because our understanding of nature has grown tremendously in the past hundred years or so, it was the scientists in the field who got to consider that nature looks like it has more than this casual relation with mathematics. It was not just the suggestion of that casual relation but also the deeper understanding of how nature seems to be constructed. While we don’t understand a lot of things about nature, it was this comprehendible thing about it that made many scientists make that connection.

The quote of Wigner’s “Unreasonable Effectiveness of Mathematics in the Natural Sciences” is very well known and pointed to as one of the first hints. Another hint you can see in the classic textbook by Wheeler , Misner and Thorne GRAVITATION where the first attempts were made to drive the law of physics by logic which they called pre-calculus. As our knowledge increased more people got to consider it like Wolfram in New Kind of Science, Conaway’s game of life, all kinds of automata ideas, Fractals and not the least as we got hints from how computers generate virtual realities. But the grand slam belonged to Dr. Tegmark with his MUH. So this idea did not happen in one go but in a continuous fashion. But the man who put that in words that I think is most beautiful is wheeler.

Behind it all is surely an idea so simple, so beautiful, that when we grasp it - in a decade, a century, or a millennium - we will all say to each other, how could it have been otherwise? How could we have been so stupid?

The website is not fully developed yet, but consists of the following

1. How I arrived at the idea.
2. Basic results that shows how QM arises, written in BASIC program.
3. Description of two particles interacting and explaining the program in C++.
4. Showing the results for Bohr atom hydrogen 1s simulation.
5. 1/r law and the running phase
6. The amazing formulas deduced from the system.
7. How spin arises from 2D simulation.
8. The appearance of the mass of the electron through simulation.
9. How gravity arises.
10. Other results and discussion.

1. How I arrived at the idea.

Reality exists hence we say it is true. But what is really true besides that more than anything else which we can really trust, it is mathematical facts. So, to my mind I connect both since both seem to be a statement of truth. So I took a guess that reality is something akin to a circle (truth). The relations between the points give you a mathematical structure whereby you get PI which defines the structure of the circle.

So I was thinking the relation(s) between what entity(s) could give a rise to a universe (truth). To come up with a structure with some entities, the easiest way was to see if I could draw two entities and define some kind a rule for their interaction. At that time I was familiar with fractals and vaguely heard of Conway's idea, but I did not know about either Dr. Tegmark or Wolfram's- New Kind Of Science-. I said let me see maybe I will be smarter than Conaway and get some really fancy rule between some triangle or circles or lines or whatever. But as soon as I put a blank sheet in front of me, for a short while I thought to myself this sounds very enigmatic, first by what criteria I am going to choose my entity, and which characteristic of that entity I was going to interrelate them and what expression. Choosing by trial and error was not very natural.

My intuition was telling me I needed something more natural. Being an engineer and a programmer we learn to be efficient in our designs. So I opted first for the simplest configuration and that was point and to start simple and not to draw points all over the paper, I restricted myself to a line. Now, if I iterate on an artificial formula I will just get fractals which has already been tried which gives you beautiful suggestive pictures but that's all. Also the different formulas I could use were most unnatural. So I thought the only way out is to throw random numbers on the line and see what happens. Off course, after a bit more than few seconds it was obvious I am going to get a uniformly distributed points on the line, I don't have to tell you that I was sad at that point(although I should have been happy as hell, you will see why). How I was to get out of this conundrum, other than mangling that paper, throwing it in the garbage can and go to a party.

The only other thing to do was to throw random lines that did not exceed an original line of length L. One more choice was necessary is to choose where those lines started, the obvious choice was random position on that line L.

After that thought analysis I went to my pc and downloaded a simple BASIC program and started coding the idea thinking I was going to get some fractal like universe or something useless. Creating the random length lines and their random positions was straightforward, but now I had to decide on what logic/constraint to use to eliminate the lines which were going outside of the original line. I tried few of them with not too complicated expressions and the output, the random positions, looked jittery but when approximated looked like some kind of a trigonometric function. So I superimposed a sine, cosine, sin^2, cos^2.

Using the simplest expression for the constraint I was in a shock, it matched perfectly sin^2, and that was the solution for Schrödinger's equation for a particle in a 1D box, with the probability of the particle position directly (no complex wave). Just from that I knew at that time that I was onto something big. Reality was nothing but random numbers (representing lines and their positions) just like what I have suspected.

The next natural thing to do is to generalize to 2D and 3D, I was in a shock again, it was so simple, just repeated the code for 1D and labeled appropriately, and plotted, a perfect probability wave for first energy level in 2D and 3D. The amplitude came out also perfectly once I normalized the probability positions to the number of throws, 2/L, L being the original line length. That was natural because probabilities had to add up to 1.

The next question that presented itself was how to calculate the energy for the particle in a box and what does it take to get the higher energy levels. It turned out, just like it should, that both questions are linked. The only obvious choice for calculating the energy was to somehow add up the lengths of the random lines. It did take some testing to see the correct expressions, because, you could take several expressions like...

but only one matched the behavior that we get from solving Schrödinger eq. to get the energy level and that is adding up all the line lengths for each point. Calculating energy that way (after normalizing) it was quadrupling when the distance of confinement was halved in a perfect match to the higher energy levels.

For a moment at that time the situation looked hopeless, how can I get any interaction going, even simpler, how to add some potential? I was having some doubts about the model, on one hand, it was reproducing sh. eq. on the other I noticed early on that I could reproduce any function in math by randomly throwing lines and applying certain constraints more complicated than the one which reproduced sin^2. I asked myself could it be that what I have discovered was just a math trick.

That situation did not last long because this model is amazing in more than one way. In a way creating these function was a blessing, because now I have a choice, very few actually, to create an interaction between the particle in a box and a potential like function ,say 1/r or exp(x) or a step function, for example. The reason why I said amazing, is that the only choice I had for interaction is to set a relation between the random lines for the different entities, mostly, if two lines crossed or not. Then I only keep the probability position for the lines that did not cross. Long and behold, you get probability waves similar to solving Schrödinger equation with a potential. The model was pushing me to do the only available choice which led to the right results.

And that led me to the next step, setting up a particle in a potential well. I could use a step function or just another particle in a smaller range within the original particle space. And the result was perfect. I got the exponential decay for the parts which were overlapping and the tunneling with continuity automatically satisfied. There was no

turning back at that point.

Next I will describe two particles interacting.

2. Basic results that shows how QM arises, written in BASIC program.

Pick a line of size "l" throw a random number denoting a position ("p") on the line and associate a line whose length ("li") is also chosen randomly but cannot exceed the length "l". Set a constraint with a particular relation between "l", "p", and "li" such that if this relation holds ignore the outcome, otherwise register the position. ***Boom, the outcome is the solution for Schrödinger equation for a particle in an infinite potential well (probability wave sin^2) (Fig. 1).*** My line of thought was: if nature is made of math then the best place to start is with a line. And what could be happening on this line. Nothing much, really, a point and a piece of line over and over. For the next energy level divide the line using natural numbers (how convenient). Simple line and simple rules lead to this gigantic dance of reality. It has been a dream of many that we owe our existence to some kind of automata. Well, this is it, *maybe*.

The program is listed here. It is written in liberty basic and you can download it here. And if that is not enough I simulated two particles where the second acts like a potential. The two particles interact according to some basic rule (see the program). When you make the potential narrow i.e. the energy high, the first particle gets constrained and its probability goes to zero at the potential (fig. 2). When the second particle has a wide base then it very much acts like a square well and the probability looks exponential inside it (fig. 3). And tunneling phenomenon is clear, continuity automatically satisfied. In other words Schrödinger equation popped out of some geometry with rules. Crazy enough? No? Here is some more: I can make the particles interact without their wave functions overlapping just by not restricting the size of the associated length "li". That is spooky action at distance. Left as an exercise.

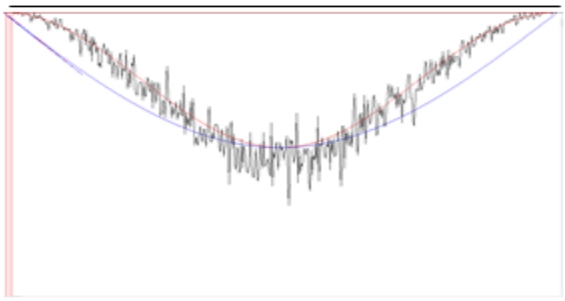


Figure 1: red line is sin^2, generated function simulates the same. In the program. In the program sc1 = 4, sc2 = 1

d1=3, st1=4 (second particle .omitted)

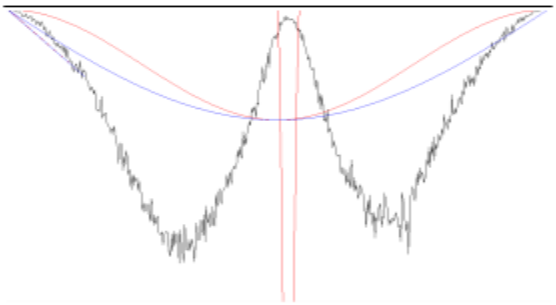


Figure 2: red line is sin^2, two particles , red particle simulating a steep potential (notice behavior as infinite potential). In the program sc1 = 4, sc2 = 1, d1=40, st1=250.

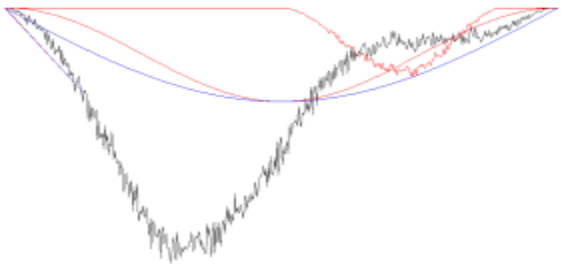
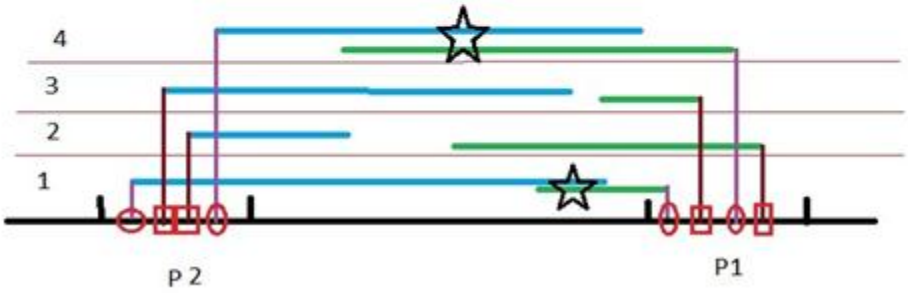


Figure 2: red line is sin^2, two particles , red particle simulating a potential (notice exponential decay & tunneling). In the program. In the program sc1 = 4, sc2 = 1, d1=180, st1=300.

3. Description of two particles interacting and explaining the program in C++.



the thumbnail shows 1D implementation. 1,2,3,4,.... are the number of loops. in each loop I throw two numbers for each particle denoting their position and length. if the lines cross (star) I ignore I don't register the position(the round marks) or don't do anything with the lines. But if they don't cross then I have a counter that updates the number of times a hit happened in the particular position (the squared marks). then for each particle I have a counter that simply adds the lengths of this line to the previous total for each particle.

I do that (loops) a million, sometimes a 100 trillion times. then I normalize to the number of throws. the totals of the lines (normalized) are the energy. the numbers of hits for each positions is operated on to get the expectation values. normalized position hits are the probabilities that are similar to the ones we get from the "squaring" of the wavefunction. Without interaction the expectation value is the midpoint of the particle. But when interaction happens the expectation value moves. lets say to left in the left particle and right in the right particle. That denotes a repulsion. you can also get attraction with different logic. But more on the logic part later.

then the particles are moved to a different distance and the operation is repeated.

Now I explain the code in more detail. see here.

The code that you see is the cleaned up version of the one in the website.

1. define variables/types

2. set the particle widths (d0,d1) , which I interpret as the Compton wavelength, I assume lambda= h/mc the model shows (I will show why) that h=c , so lambda =1/m ,then I choose m to be in au hence if m=.0005485 then lambda=1822.8885 units of length on the axis/line . more on scale later.

3. set the interval (intr), that is used as a quantity to increase the distance between the particles after the calculation finished for certain distance.

4. start the mk loop that will increase the distance between the particle after each iteration.

5. based on mk value set the positions of the particles, zero out some of the variables need be. f1 is the number of hits for crossing f for not crossing. Zero out the arrays (S[],Sy[]),that hold the hits for each position on the axis/line.

6. next is the j loop the heart of the program, it iterates on the random throws

7. don't worry about these lines, not important

```
long r= rand();
double rndm=(double)r/((double)RAND_MAX);
```

8. calculate the start of the lines from inside of the particles and the length of the lines shooting to the other particl all based on random numbers.

9. use if (st1+p1 + li1 > st0+ p - li) to check if lines crossed or not.

10. if not crossed update the position hit by incrementing the counter S[] for that position. add the random line to an accumulation counter (en). I do that for one of the particles only. the other will be similar.

While I said I don't do anything when lines crossed but in this program I do the same using Sy[], en1 just for information. I will talk more about it later.

11.go to 6

12. when done with j loop normalize the energy en to the numbers of throws accepted frf = (double)f/en; //energy of the particle

13. calculate the expectation value for the position array S[] -over the width of the particles.

```
edx = edx + (( n) * S[n]);
```

calculate how much expectation is offset from center of the particle

```
ex[mk] = (double)edx / ((double)f)- (0.5 * int(w*d1))+.5 ;
```

14. update all data in file for that separation.

15 . go to mk loop for new separation distance

16. Done

4. Showing the results for Bohr atom hydrogen 1s simulation.

Here is the most important first result from the three results that I will show. The results confirm that the classical Bohr Model falls out from QSA model which encompasses QM and QFT.

Please always refer to these wiki

Bohr radius - Wikipedia, the free encyclopedia
Bohr model - Wikipedia, the free encyclopedia

this is the result of simulating two particles with a width of 1823 which is close to 1822.8885 for electron compton wavelength (just simplification)interacting at a seperation of around Bohr radius which is
1/(m*alpha)=1/(.00054858*.007297352569) = 249801.3

the raw data is below from the program with int=50. also make this change in the program to get these results
for (mk = 2475; mk <= 500000000; mk++)
also
d0 =1823; // Particle 1 size
d1 = 1823; // Particle 2 size

long long kj =20000000000; // # of random throws (approx 30 min for each distance)
but next I give the important data that we will discuss
[CODE]
distance energy (P.E.) charge^2(e^2) Expectation value(Ex)
249323 0.0000120326 3.000003457 2.219640631
249423 0.0000120278 2.999998876 2.219325817
249523 0.0000120229 3.000000804 2.217591031
249623 0.0000120181 3.000000809 2.217731633
249723 0.0000120133 2.999998829 2.215744702
249823 0.0000120085 3.000006682 2.215434488
249923 0.0000120037 2.999998356 2.214921159
[/CODE]

because I have the 1/r law I interpret the energy as e^2/r , e=charge
so if you multiply distance *energy= e^2=3 as shown, the average of above e^2= 3.000001, but we will take 3 to simplify.
then because we know alpha I deduce that (from alpha=e ^2/(h*c))
h*c=e^2/alpha= 3/ .007297352569= 411.108
from other arguments I have h=c= sqrt(411.108)= 20.2758
Now, the important part which Expectation value(Ex for short)
after inspection I find it to be related to the classical bohr model variables
Ex=v^2/(2*m*e^4) ---------- eq 1
solving for v^2=(2*m*e^4)*Ex ---------- eq 2
from above simulation the average of Ex= 2.2172 almost
hence v^2= (2*.0005485*9)*2.2172= 0.0218936
v= sqrt(0.0218936)= 0.14797
now we compute v/c=0.14797/20.2758= 0.0072976

v/c should be alpha we have a very good match with some error mostly because of Ex which we can simulate with higher j thows to get more accuracy and also due to the approxomation of 1823 and 1822.8885

Great we proved that Ex is what it is and h=c

next

from eq 1 we can compute the kinetic energy

K.E.= (m^2*e^4)*Ex=(1/2)*m*v^2=.5*.00054858*0.0218936

= 0.000006005195544

2*K.E.= 0.000012010

That is Bohr Model P.E.= 2*K.E.

So the energy has the interpretation of potential energy and Ex is related to K.E. , that makes perfect sense

also if we take 1/(2*Ex)=1/(2*2.2172)=0.22551 almost m*c^2

m*c^2=.00054858*411.108= 0.225526

errors should be taken into account as mentioned earlier

Q.E.D

[CODE]2475 249323 1.2032598102434993e-005 5.9941919763095141e-006 3.0000034566933995 1.823 2.2196406306904919 -2.1562643940237649

2476 249423 1.2027755561853412e-005 5.9917972737823617e-006 2.9999988755041636 1.823 2.2193258174578432 -2.1564566940604664

2477 249523 1.202294299276971e-005 5.9894015680537819e-006 3.0000008043848765 1.823 2.2175910306876858 -2.1541575969249607

2478 249623 1.2018126569310113e-005 5.9870183764102081e-006 3.0000008086108982 1.823 2.2177316327685048 -2.1541184268552342

2479 249723 1.2013306059579073e-005 5.9846252913626688e-006 2.9999988291162647 1.823 2.2157447018220182 -2.1534291553698495

2480 249823 1.2008528765272037e-005 5.9822392351484673e-006 3.000006681726556 1.823 2.2154344880241297 -2.1517840085696207

2481 249923 1.2003690562073311e-005 5.9798542464554e-006 2.9999983563450483 1.823 2.2149211586316824 -2.1521569239379232

1/r law and the running phase

here, I show (thumbnail) the 1/r law generated by the system. I just compute few more points in the same simulation for the first result. And then plot distance vs energy(P.E.) and do curve fitting.

Next graph concerns the running phase of the charge until its relative stabilization. The interaction starts when the particles are very close to each other. the interpretation of the results in quite bit more involved, but shows the basic feature.

```
#include <math.h>


#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <ctime>
#include <iomanip>
#include <fstream>
using namespace std;
        // Global arrays
            long long    S[18230000];
            long long    Sy[18230000];
          double ex[90000];
        double ex1[90000];
            double fr[90000];
int main() {
        srand(time(0));
            long long    i=0;
            long long    g=0;
          double frf;
             double dist;
            double l;
            double d1;
            double st1;
```

```
        double d0;
      double st0;
      long long    f;
      long long    f1;
      long long edx;
      long long edx1;
      long long m;

   double p;
      double li;
        double p1;
        double li1;
long long w =10000;
        double en;
        double en1;
    double alpha = 0.0;
    double a1=0.0;
    double a2=0.0;
    double a3=0.0;

     double avg=0;
    double cn =0;
    double ent =0;
    double intr;
    double b1=0.0;
            l = 1000000;            // Universe size
            d0 =1822.8885;          // Particle 1 size
            d1 =1822.8885;          // Particle 2 size

    long long kj =1000000000;// # of random throws
            double d0div = d0 ;
            intr=.05;
```

```
cout << intr <<"\n";

long mk;
                for (mk = 2479778; mk <= 500000000; mk++)

                        {
                                dist = (mk) * 2 * intr+(d0);

                                st1 =(l/2)- (((double)mk ) * intr);

                                st0 = (l/2)+ d1 + (((double)mk ) * intr);

                                f = 0;      f1 = 0;     edx = 0;  edx1 = 0;             en = 0.0;     en1=0;     ent=0;

                                cout << l/2<<"\n";

            long long kk;
              for (kk = 0; kk <= 20000000; kk++) {
                    S[kk] = 0;
                    Sy[kk] = 0;

              } // Next kk

//number of throws
                  long long j;
              for (j = 0; j <= kj; j++) {

                                        long r= rand();
                                        double rndm=(double)r/((double)RAND_MAX);
                                        long r1= rand();
                                        double rndm1=(double)r1/((double)RAND_MAX);
```

```
                    long r2= rand();
                    double rndm2=(double)r2/((double)RAND_MAX);

                    long r3= rand();
                    double rndm3=(double)r3/((double)RAND_MAX);

                    long rx= rand();
                    long ry= rand();
                    double rndmx=(double)rx/((double)RAND_MAX);
                    double rndmy=(double)ry/((double)RAND_MAX);

p =    (d0 *(rndm));

li = ((d0+dist)    *rndm2);

p1 =    (rndm3 * d1);

li1 = (rndm1 * (dist+d0));

   if ( st1+p1 + li1 > st0+ p - li)

  {
       Sy[int(w*p)]=Sy[int(w*p)]+1;

    en1=en1+li;

       f1 = f1 + 1;

  }

  else
```

```
                    {

                        S[int(w*p)]=S[int(w*p)]+1;

                      en=en+li;

                        f = f + 1;

                    }

                } // Next j
                   frf = (double)f/en;      //energy of the particle


                en1 = (double)f1/en1;
            long long n;
                   for (n = 0; n <= int(w*d1); n++) {

// CALCULATE EXPECTATION VALUE AFTER INTERACTION HAS TAKEN PLACE
                        edx = edx + (( n) * S[n]);
                edx1 = edx1 + (( n) * Sy[n]);


                   }


ex1[mk] = (double)edx1 / ((double)f1)- (0.5 * int(w*d1))+.5 ;

ex[mk] = (double)edx / ((double)f)- (0.5 * int(w*d1))+.5 ;

cout << std::setprecision(17)<< dist<<"        "<<frf << " "<<en1<<" " <<f1<<"      " <<ex[mk]<<"      "
<<ex1[mk]<<"\n";
```

```
        a2= (dist*frf);
        a3= d0div/1000;

                        ofstream fo;
                        fo.open ("yem.txt", ios::out | ios::app);
                        fo << mk << "      " << std::setprecision(17) <<dist<<"      "<< frf << " " <<en1<< "    "<< a2
<< "      "<< a3 << "          "<<ex[mk]<<" "<< ex1[mk] <<"\n";
                        fo.close();
                  i = 0;
                    g = 0;
                 } // Next mk
          system("pause");
          return 0;
}
```

6. The amazing formulas deduced from the system.

check out these formulas that I deduce from my theory, these are NOT a guessed formulas.

fine structure constant ,alpha=α = 7.297352568 × 10-3

let

1/m_e= 27*(1/(2*alpha) -(.5*alpha) -1) + (alpha/(2*PI))

1/m_e=1822.888474(approx).....

the term (alpha/(2*PI)) it is related to spin, this term is not deduced from my theory(just a guess but it is small)
....

Also first term 27*(1/(2*alpha))=1849.98599...
Two terms 27*(1/(2*alpha) -1)= 1822.98599
average (1849.98599+1822.98599)/2 -1/3= 1836.15266 electron-proton ratio !!!

Actually 27=3^3, 3 is equal to e^2(charge squared!!) in my system, the long term not the running phase.

Also reversing the relation I find

1/alpha= (2/3)(1/m_e*e^4) +2 +(3/2)(m_e*e^4) Approx.(m_e=1/1822.8885) (e^2=3), e^4=9

beautiful symmetry. all these equation can also be written in a form that uses the golden and silver ratios (google)

also

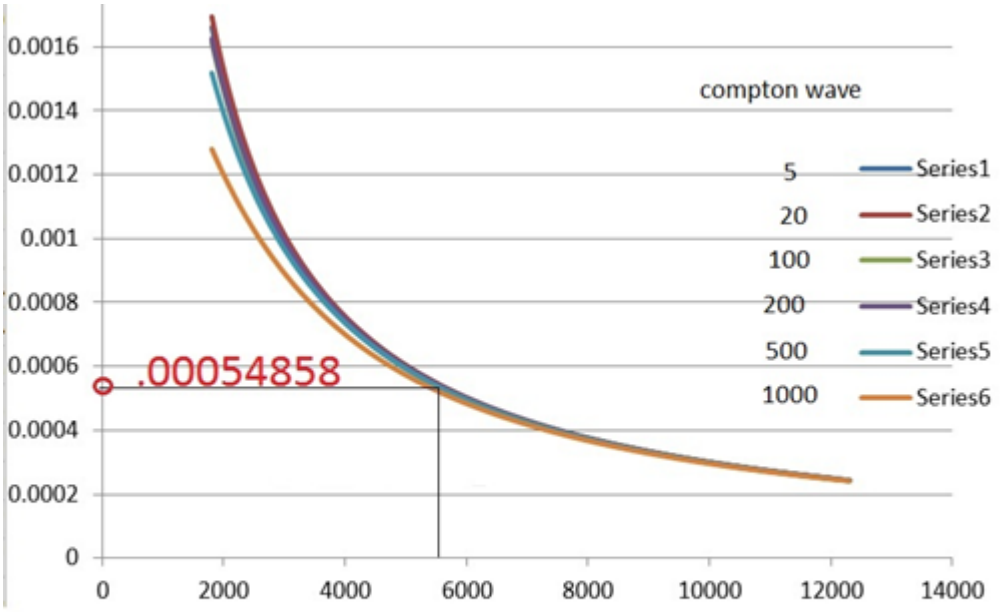
electron g-factor=(4m_e/3eh)*(2/(3*m_e*alpha) - 2*e^2 -1)

=2.00231934...

.. e=3(charge square),h actually h_bar=(e/alpha)^.5=20.2758.. m_e=.00054858

8. The appearance of the mass of the electron through simulation.

Here it is, what I call the most beautiful graph ever, I simulate two particles interacting, with different Compton wave lengths for each run. they all converge on the .00054858 the mass of the electron.



Ok, here is the electron mass and spin relation. if you look at the attachment you will see as the particle size approaches 1823,1822,1821 then at 1820 you get a **flip** in those notches and the frequency becomes minimum. I interpret that there is a relation such as 1820+3(charge square)=1823 which is almost electron mass.

also the same behavior appears at 1820*5, 1820*3,1820*2,1820*2,1820*1,1820/2,1820/3,1820/4 ...

attached mass.txt has the code just plug the particle size (1823 ....) into d0 and d1 in the code. take the data from "y5a.txt" and put it in excel sheet and plot. Please erase "y5a.txt". before you run again for different particle number.

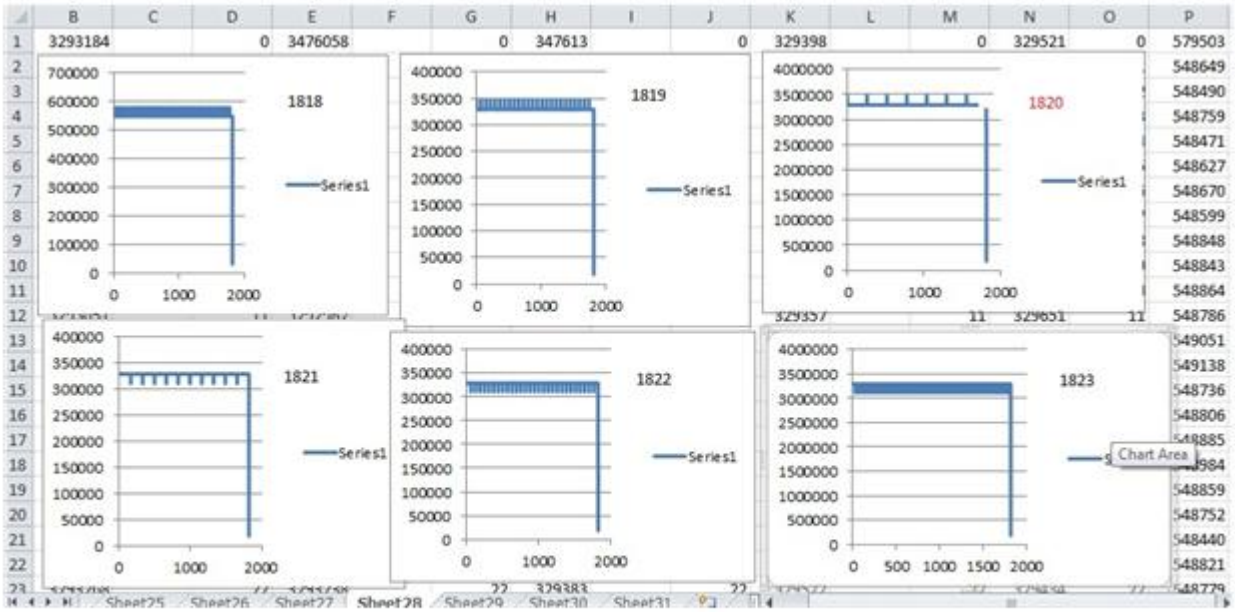
change code to prevent from repeated runs
from

```
for (mk = 1; mk <= 100; mk++)
```

to

for (mk = 1; mk <= 1; mk++)

Electron mass and spin relation



Appendex A

```
l = 500

t= 0

dim S(1000)


dim L(1000)

dim S1(1000)

'open "c:\list1.txt" for output as #1

dim L1(1000)

z=.5


'open "b" for graphics as #dra

'print #dra, "home ; up ; north ; turn 180 ; go 400 ; down ; north ; turn 90"

open "Draw" for graphics as #draw

for n=1 to 10

for j = 1 to 1000000

d1=40

st1=300

p= int(l * rnd(0))

li = int(l * rnd(0))

p1= st1+int(d1 * rnd(0))

li1 = int(d1 * rnd(0))


''if p1=p2 goto [q]

''if p1<75+n or p1>90+n goto [q1]

'if p<30+n or p>45+n goto [q2]

if p1+li1>p-li and p1+li1<p+li goto [q]

if p+li>p1-li1 and p+li<p1+li1 goto [q]

if (st1+d1-p1-li1)/d1 < rnd(0) then goto [qc]
```

```
if (-1*st1+p1-li1)/d1 < rnd(0) then goto [qc]

L1(p1)=L1(p1)+li1


S1(p1) = S1(p1) + 1

[qc]


if (l-p-li)/l < rnd(0) then goto [q]

if (p-li)/l < rnd(0) then goto [q]


L(p)=L(p)+li


S(p) = S(p) + 1



goto [q]


[q1]

'S1(p1)=0

goto [q]

[q2]

'S(p)=0

[q]

next j

'print #dra, "down ; place 0 0 ; go 1000     "


'for j = 1 to 50

'print #dra, "up ; goto " ; 100 + Po(j) ; " " ; j * 10

'print #dra, "down ; go " ; Lo(j)
```

```
'print #dra, "flush"
'next j
'rem wait

for k = 1 to l
print k , S(k),S(k)^.5,L(k)/(S(k)+1)
tx=tx+S(k)
tn=tn+L(k)/(S(k)+1)

print sin((k*3.14)/l)*sin((k*3.14)/l)*y

next k
y=7.5*tx/k
print tx,tn/l


  print #draw, "home ; down "

  for r = 1 to l
  'print #1, S(r)
  print #draw,"goto "; r * 2 ; " "; S(r)*5
  'print #draw, "flush"
  next r
  print #draw, "up ; home ;color red ; down"
    print #draw, "home ; down "

  for r = 1 to l
  'print #1, S1(r)
  print #draw,"goto "; r * 2 ; " "; S1(r)*1
```

```
  'print #draw, "flush"
  next r

  print #draw, "up ; home ;color red ; down"
  for e = 1 to l
  print #draw, "goto "; e* 2 ; " "; sin((e*3.14)/l)*sin((e*3.14)/l)*y
  'print #draw, "flush"
  next e
    print #draw, "up ; home ;color blue ; down"
  for e = 1 to l
  print #draw, "goto "; e* 2 ; " "; sin((e*3.14)/l)*y
  print #draw, "flush"
  next e
' close #1
print n
input g
print #draw, "cls"
next n
  wait
  'print #draw, "home ; up ; north ; turn 180 ; go 100"
  'print #draw, "down ; north ; turn 90 ; go 100 ; turn 180 ; go 100"
  'for r = 1 to 100
  'print #draw, "down ; north ; go ";     S(r)
  'print #draw, "up ; north ; turn 180 ; go "; S(r)
  'print #draw, "turn 90 ; go 1 ; flush"
  'next r
  'wait

  rem open "Draw" for graphics as #draw
```

```
rem     print #draw, " home ; north ; down"

rem     for r =           1 to 100

rem     print #draw, " home ; north ; goto     1,1"

    rem print #draw,     "turn 90 ; go " ; r

rem print #draw, "flush"

rem next r

rem wait
```