

The Quantum Pyramid

Source: <http://sci.tech-archive.net/Archive/sci.physics/2006-10/msg01768.html>

- *From:* CoreyWhite@xxxxxxxxxx
 - *Date:* 27 Oct 2006 21:26:53 -0700
-

I spent quite a bit of work trying to get this to format on google groups, so let me know if it needs explanation. Basicly the tree sorts randomly, and can start from any number on the pyramid. It has a 50/50 chance of moving down either tree. If it continues to the next number down the tree it advances. Otherwise it retreats back up the tree on the adjacenet path. When it reaches 0 or 1, it can switch back over, and start moving in the opposite direction

```
^
/\
/\
1 0
/\ / \
2 | \ | -1
/\ / \
3 | | -2
/\ / \
4 | | -3
/\ / \
5 | | -4
/\ / \
6 | | -5
/\ / \
7 | | -6
\ | /
\ | /
\ | /
^ ^
```

Here is how the pyramid looks if you have a universal font installed in your usenet or e-mail client.

```
^
/v
1 0
/^ /\
```

The Quantum Pyramid

2 | \ | -1
/ \ | \
3 | ^ ^ | -2
/ \ | \
4 | ^ ^ | -3
/ \ | \
5 | ^ ^ | -4
/ \ | \
6 | ^ ^ | -5
/ \ | \
7 | ^ ^ | -6
| |
^ ^

In quantum physics everything functions on random reactions, and we see this in traditional physics too, within chaotic systems. But when looking at the quantum pyramid there are two different ways you can see it. You can let a particle fall through the pyramid completely at random, or you can stop and *observe* the ball falling through the pyramid at any given point.

If we just let the particle randomly start at either 1 or 0, it could wind up at -6 or 7 next we look. But if we have a particular goal in mind and we decide to observe the particle and allow it to start at 1 instead of 0. Then the expectation the particle will get to 7, instead of ending at -6 are incredibly higher. And if we decide to observe the ball at 2 or 3 instead of one of the other numbers, then the odds increase even more.

What I am trying to say is that because we are observing the pyramid, and by that I mean we are looking at our particle at a given spot on the pyramid. We can actually control where the particle will be in the future. Otherwise, when we aren't observing the particle, its future is entirely random, and it could be literally anywhere on the pyramid.

But once we observe the pyramid, we know where the particle will be in the future, even though it is moving in a completely random pattern.
8)

And by observe I mean we actually create, literally. Because we can start the particle off at a point in the pyramid ourselves, and let it operate within the random chaos sphere.

Now here is the math:

To calculate the average number of moves before you advance n steps forward. The equation $k(n-k)$ will work according to martingale probability theory. k is equal to the

The Quantum Pyramid

absolute value of the number you start at and n is equal to your goal. So if we start at 4, and only try to advance to 6, the average number of moves it will take before we get there is $4(6-4)$, which equals 8 moves.

To show the final proof of the quantum pyramid you only have to move from the probability of winning the first game, and multiply it by the probabilities of winning the following games. For example, if we are at 3 and only try to advance one step, the odds are $3/4$. And once that penny is collected there is now a $4/5$ th chance of winning another penny.

You calculate this easily by counting all the possible ways you can advance a step, to get the numerator, and adding 1 to it to get the denominator. Because there is only one way you can lose. To count all the possible ways you can win, just count how many smaller pyramids there are along the path, and add those to the direct route.

Here is the source code that proves the quantum pyramid's results:

```
.. #include <stdio.h>
.. #include <stdlib.h>
..
..
.. main ()
.. {
.. double r;
.. long int M;
.. double x;
.. int y;
.. int z;
.. int count;
..
..
.. int seed = 10000;
.. srand (seed);
.. M = 2;
..
..
.. int score = 0;
..
.. //Score keeps track of the number of beans won every game
..
..
.. int games = 0;
..
.. // games keeps track of the number of games we have played before
.. //losing all of the beans, which is equal to score.
..
..
```

The Quantum Pyramid

```
.. int beans1 = 0;
..
.. // Initial value set to zero and defined within the loop
..
.. int wins = 0;
.. int lost = 0;
.. int quit = 0;
.. int init = 0;
..
.. printf ("Initial Beans: ");
.. scanf ("%d", &init);
.. printf ("Stop after winning X number of beans: ");
.. scanf ("%d", &quit);
..
.. for (int cnt = 0; cnt < 10000; cnt++)
.. {
.. // We play 10,000 rounds
..
..
.. int count = 0;
.. beans1 = init + score;
..
.. // Beans gets defined here, as starting with 3 beans
.. // and having a 0 bonus score (It changes as you
.. // win more beans per round)
..
..
.. int beans2 = 1;
..
.. // The program attempts to win just one
.. // bean for every game.
..
..
.. while (beans1 != 0 && beans2 != 0)
..
.. // The battle begins
..
..
.. {
.. r = ((double) rand () / ((double) (RAND_MAX) + (double)
(1)));
..
..
.. x = (r * M);
.. y = (int) x;
..
.. z = y + 1;
..
.. // A coin is flipped and is either 1 or 2 in value
..
.. if (z == 1)
```

The Quantum Pyramid

```
.. {
.. // Heads wins.
..
.. beans1++;
..
.. // Beans1 gains one bean from Beans2
..
.. beans2--;
.. }
.. if (z == 2)
.. {
.. // Tails loses
..
.. beans1--;
..
.. // Beans2 gains one bean from Beans1
..
.. beans2++;
.. }
..
.. count++;
..
.. // We keep track of the number of rounds in the battle
..
.. }
..
..
.. if (beans1 > score + init)
.. {
.. // If beans1 is greater than the initial value
.. // of beans plus the total number of beans
.. // that have been won so far in this game, then
.. // the score goes up, and we go on to the next
.. // game. We check this at the end of every game.
..
.. score++;
..
.. games++;
.. }
..
.. if (beans1 <= 0)
.. {
.. //If beans1 has lost the game and doesn't
.. //have anymore beans then we know the
.. //game is over, so we reset score, and reset
.. //games.
..
..
..
.. printf ("Lost at: %d beans , %d games.\n", score + init,
games);
..
..
```

The Quantum Pyramid

```
.. // And we print out the total number of
.. // games played on this trial and show the
.. // total score plus the initial value of beans.
..
.. lost++;
.. score = 0;
.. games = 0;
..
.. }
..
.. if (score >= quit)
.. {
.. wins++;
..
.. printf ("Won at: %d beans , %d games.\n", score + init,
games);
..
.. beans1 == 0;
.. score = 0;
.. games = 0;
..
.. }
..
.. }
..
.. printf ("Total Won: %d/%d\n", wins, wins + lost);
..
.. }
.
```