Can apparent superluminal neutrino speeds be explained as a quantum weak measurement?

M V Berry[1], N Brunner[1], S Popescu[1] & P Shukla[2]

[1]H H Wills Physics Laboratory, Tyndall Avenue, Bristol BS8 1TL, UK

[2] Department of Physics, Indian Institute of Technology, Kharagpur, India

Abstract

Probably not.

Example Problem: Simple Statistics

- Many programs deal with large collections of similar information.
  - Words in a document
  - Students in a course
  - Data from an experiment
  - Customers of a business
  - Graphics objects drawn on the screen
  - Cards in a deck

Sample Problem: Simple Statistics

Review code from chapter 8, what does it do?

```
# average4.py
#    A program to average a set of numbers
#    Illustrates sentinel loop using empty string as sentinel

def main():
    sum = 0.0
    count = 0
    xStr = input("Enter a number (<Enter> to quit) >> ")
    while xStr != "":
        x = eval(xStr)
        sum = sum + x
        count = count + 1
        xStr = input("Enter a number (<Enter> to quit) >> ")
    print("\nThe average of the numbers is", sum / count)

main()
```

Sample Problem: Simple Statistics

Now extend the program to compute not only the mean, but also the median and standard deviation.

Median

- *median* is data value that splits data into equal-sized parts
- For 2, 4, 6, 9, 13, the median is 6, since there are two values greater than 6 and two values that are smaller
- One way to determine median is to store all numbers, sort them, and identify middle value

Standard Deviation

- *standard deviation* is a measure of how spread out the data is relative to the mean
- If the data is tightly clustered around the mean, then the standard deviation is small. If the data is more spread out, the standard deviation is larger
- The standard deviation is a yardstick to measure/ express how exceptional the data is

Std Dev Equations

- The standard deviation is $$s = \sqrt{\frac{\sum\left(\bar{x} - x_i\right)^2}{n-1}}$$

- Here $\bar{x}$ is the mean, x_i represents the *i*th data value and *n* is the number of data values
- The expression $\left(\bar{x} - x_i\right)^2$ is the square of the "deviation" of an individual item from the mean

Std Dev Calculations

- The numerator is the sum of these squared "deviations" across all the data
- Suppose our data was 2, 4, 6, 9, and 13.
  - The mean is 6.8
  - The numerator of the standard deviation is

$$(6.8-2)^2+(6.8-4)^2+(6.8-6)^2+(6.8-9)^2+(6.8-13)^2=149.6$$

$$s=\sqrt{\frac{149.6}{5-1}}=\sqrt{37.4}=6.11$$

Sample Problem: Simple Statistics

- Calculating the standard deviation not only requires the mean, but also each individual data element!
- Need a way to remember values as they are entered
- Combine an entire collection of values into one object?

Lists and Arrays

- Python **lists** are ordered sequences of items. For instance, a sequence of *n* numbers might be called *S*: $S = s_0, s_1, s_2, s_3, \ldots, s_{n-1}$
- Specific values in the sequence can be referenced using subscripts
- By using numbers as subscripts, mathematicians can succinctly summarize computations over items in a sequence using subscript variables

$$\sum_{i=0}^{n-1} s_i$$

Lists and Arrays

- Suppose the sequence is stored in a variable `s`. We could write a loop to calculate the sum of the items in the sequence like this:

```
sum = 0
for i in range(n):
      sum = sum + s[i]
```

- Almost all computer languages have a sequence structure like this, sometimes called an array

Lists and Arrays

- A list or array is a sequence of items that is referred to by a single name (i.e., `s`) and individual items are selected by indexing (i.e., `s[i]`)
- In other programming languages, arrays are generally a fixed size,
  - when you create the array, you must specify how many items it can hold
- Arrays are generally homogeneous,
  - meaning they can hold only one data type

Lists and Arrays

- Python lists are dynamic. They can grow and shrink on demand
- Python lists are also heterogeneous, a single list can hold arbitrary data types
- Python lists are mutable sequences of arbitrary objects

List Operations

Operator	Meaning
<seq> + <seq>	Concatenation
<seq> * <int-expr>	Repetition
<seq>[]	Indexing
len(<seq>)	Length
<seq>[:]	Slicing
for <var> in <seq>:	Iteration
<expr> in <seq>	Membership (Boolean)

List Operations

- Except for the membership check, we've used these operations before on strings
- The membership operation can be used to see if a certain value appears anywhere in a list

```
>>> lst = [1,2,3,4]
>>> 3 in lst
True
```

List Operations

- The summing example from earlier can be written like this:

```
sum = 0
for x in s:
    sum = sum + x
```

- Unlike strings, lists are mutable:

```
>>> lst = [1,2,3,4]
>>> lst[3]
4
>>> lst[3] = "Hello“
>>> lst
[1, 2, 3, 'Hello']
>>> lst[2] = 7
>>> lst
[1, 2, 7, 'Hello']
```

List Operations

- A list of identical items can be created using the repetition operator. This command produces a list containing 50 zeroes:

```
zeroes = [0] * 50
```

List Operations

- Lists are often built up one piece at a time using append

```
nums = []
x = eval(input('Enter a number: '))
while x >= 0:
    nums.append(x)
    x = eval(input('Enter a number: '))
```

- `nums` is being used as an accumulator, starting out empty, and each time through the loop a new value is added

List Operations

Method	Meaning
<list>.append(x)	Add element x to end of list.
<list>.sort()	Sort (order) the list. A comparison function may be passed as a parameter.
<list>.reverse()	Reverse the list.
<list>.index(x)	Returns index of first occurrence of x.
<list>.insert(i, x)	Insert x into list at index i.
<list>.count(x)	Returns the number of occurrences of x in list.
<list>.remove(x)	Deletes the first occurrence of x in list.
<list>.pop(i)	Deletes the ith element of the list and returns its value.

List Operations

```
>>> lst = [3, 1, 4, 1, 5, 9]
>>> lst.append(2)
>>> lst
[3, 1, 4, 1, 5, 9, 2]
>>> lst.sort()
>>> lst
[1, 1, 2, 3, 4, 5, 9]
>>> lst.reverse()
>>> lst
[9, 5, 4, 3, 2, 1, 1]
>>> lst.index(4)
2
>>> lst.insert(4, "Hello")
>>> lst
[9, 5, 4, 3, 'Hello', 2, 1, 1]
>>> lst.count(1)s
2
>>> lst.remove(1)
>>> lst
[9, 5, 4, 3, 'Hello', 2, 1]
>>> lst.pop(3)
3
>>> lst
[9, 5, 4, 'Hello', 2, 1]
```

- Suppose I give you a sorted list. How do you find the median (middle) number?

- Suppose the user enters a bunch of numbers (as done above). When he's done, how could you print out all those numbers backwards?


WINTER IS COMING


SO, RON AND GINNY ARE LOVERS?
WHAT? NO, THEY'RE BROTHER AND SISTER.
OH...
NOT EVEN A LITTLE BIT?
THE REST OF LUNCH BETWEEN GEORGE R.R. MARTIN AND J.K. ROWLING WAS PRETTY AWKWARD.
#YOUCHOOSE
@YOUCHOOSEBOOKS