How to calculate moving average without keeping the count and data-total?

I am trying to find a way to calculate a moving cumulative average without storing the count and total data that is received so far.

I came up with two algorithms but both need to store the count:

- new average = ((old count * old data) + next data) / next count
- new average = old average + (next data - old average) / next count

The problem with these methods is that the count gets bigger and bigger resulting in loosing precision in the resulting average.

The first method uses the old count and next count which are obviously 1 apart. This got me thinking that perhaps there is a way to remove the count but unfortunately I haven't found it yet. It did get me a bit further though, resulting in the second method but still count is present.

Is it possible, or am I just searching for the impossible?

moving-average

edited Sep 28 '12 at 11:43 ronalchn 10.1k 7 37 55

asked Sep 28 '12 at 8:46 user1705674 231 1 3 4

NB that numerically, storing the current total and current count is the most stable way. Otherwise, for higher counts next/(next count) will start to underflow. So **if you are really worried about losing precision, keep the totals!** – AlexR Aug 11 '16 at 16:02

5 Answers

You can simply do:

```
double approxRollingAverage (double avg, double new_sample) {

    avg -= avg / N;
    avg += new_sample / N;

    return avg;
}
```

Where `N` is the number of samples where you want to average over.

See: Calculate rolling / moving average in C++

edited May 23 at 10:31 Community ♦ 1 1

answered May 26 '13 at 8:49 Muis 3,495 6 48 88

1 Don't you have to add 1 to N in this before this line? avg += new_sample / N; – Damian Jun 13 '16 at 20:28

@Damian No, because the rolling average is apparently calculated over a fixed number of values (last `N` values). – Erik Aigner Aug 11 '16 at 7:51

4 This is not entirely correct. What @Muis describes is an exponentially weighted moving averge, which is sometimes appropriate but is not precisely what the OP requested. As an example, consider the behaviour you expect when most of the points are in the range 2 to 4 but one value is upwards of a million. An EWMA (here) will hold onto traces of that million for quite some time. A finite convolution, as indicated by OP, would lose it immediately after N steps. It does have the advantage of constant storage. – jma Mar 20 at 8:26

1 That's not a moving average. What you describe is a one pole filter that creates exponential responses to jumps in the signal. A moving average creates a linear response with length N. – ruhig brauner Jun 20 at 10:14

```
New average = old average * (n-1)/n + new value /n
```

This is assuming the count only changed by one value. In case it is changed by M values then:

```
new average = old average * (n-len(M))/n + (sum of values in M)/n).
```

This is the mathematical formula (I believe the most efficient one), believe you can do further code by yourselves

edited Jun 6 at 4:19
Mikhail
5,046 2 30 65

answered May 6 '14 at 11:41
Abdullah Al-Ageel
447 4 3

What is sum of new value? is that different somehow from "new value" in your original formula? – Mikhail Jun 4 '16 at 21:00

@Mikhail in the second example, there are `m` new values being factored into the new average. I believe that `sum of new value` here is meant to be the sum of the `m` new values being used to compute the new average. – Patrick Goley Mar 1 at 6:23

From a blog on running sample variance calculations, where the mean is also calculated using Wellford's Method:

$$M_1 = x_1$$

$$M_k = M_{k-1} + \frac{x_k - M_{k-1}}{k}$$

Too bad we can't upload SVG images.

answered Jun 15 '16 at 8:35
Flip
353 3 10

3 This is similar to what Muis implemented, except that the divide is used a common factor. Thus only one division. – Flip Jun 15 '16 at 8:41

It's actually closer to @Abdullah-Al-Ageel (essentially commutative math) in that Muis doesn't account for incrementing N; copy-paste formula reference: [Avg at n] = [Avg at n-1] + (x - [Avg at n-1]) / n – drzaus Aug 10 '16 at 20:38

@Flip & drwaus: Ain't Muis and Abdullah Al-Ageel solutions exactly the same? It's the same computation, just written differently. For me those 3 answers are identital, this one being more visual (too bad we can't use MathJax on SO). – user276648 Oct 11 '16 at 0:50

An example using javascript, for comparison:

https://jsfiddle.net/drzaus/Lxsa4rpz/

```
function calcNormalAvg(list) {
    // sum(list) / len(list)
    return list.reduce(function(a, b) { return a + b; }) / list.length;
}
function calcRunningAvg(previousAverage, currentNumber, index) {
    // [ avg' * (n-1) + x ] / n
    return ( previousAverage * (index - 1) + currentNumber ) / index;
}
```

Show code snippet

edited Aug 11 '16 at 16:13

answered Aug 11 '16 at 15:57
drzaus
12.1k 6 76 101

You can use an exponential moving average (sometimes called a "recursive average").

https://stackoverflow.com/a/10990656/166949

deleted by Bhargav Rao ♦ Mar 30 at 7:37

edited May 23 at 11:55
Community ♦
1 1

answered Dec 8 '14 at 18:01
steveha
44.7k 12 67 95

While this link may answer the question, it is better to include the essential parts of the answer here and provide the link for reference. Link-only answers can become invalid if the linked page changes. - From Review – Hakam Fostok Mar 30 at 7:15