How well can you see the slope of a digital line? (and other applications of non-uniform averaging)

James Propp
UMass Lowell
March 29, 2011

This is joint work with David Einstein, Gerry Myerson, Sinai Robins, and others, and was greatly facilitated by MathOverflow (see http://mathoverflow.net).

See

http://mathoverflow.net/questions/24517/
dedekind-esque-sums

http://mathoverflow.net/questions/26608/
a-specific-dedekind-esque-sum

http://mathoverflow.net/questions/36929/
accelerated-convergence-to-the-mean-using-quadratic-weights

http://mathoverflow.net/questions/54731/
sums-of-fractional-parts-of-linear-functions-of-n

I. Digital lines

The *digital line* associated with the (Euclidean) line

$$\{(x, ax+b): x \in \mathbb{R}\}$$

(with $a,b \in \mathbb{R}$) is the set of lattice points

$$\{(i, \text{nint}(ai+b)): i \in \mathbb{Z}\}$$

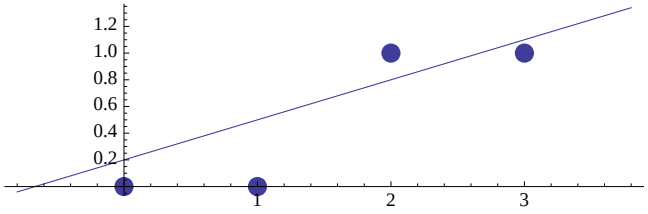
where nint(x) is the integer nearest to x.

(If $x = k + \frac{1}{2}$ so that the integers $k = x - \frac{1}{2}$ and $k + 1 = x + \frac{1}{2}$ are equally close to x, we take nint(x) = $x - \frac{1}{2} = k$.)

A *digital line segment* is the set of such lattice points
where i ranges over some interval in $\mathbb{Z}$; typically we take the interval $[0, n - 1]$ or $[1, n]$ or $[0, n]$, and we call the digital line segment a "digital line" for short.

Assume $0 < a < 1$.

Example: $a = .3$, $b = .2$, $n = 4$:
(0, nint(0.2)) = (0, 0)
(1, nint(0.5)) = (1, 0)
(2, nint(0.8)) = (2, 1)
(3, nint(1.1)) = (3, 1)



There is a large literature on digital lines, with strong ties to the theory of Sturmian words; in that setting, a digital line corresponds to a *balanced word* from an alphabet of two letters (see below).

Recognition problem: Given a set of points, determine whether it is a digital line.

This is solved in

Kim, C.E. and Rosenfeld, A. (1982),
Digital straight lines and convexity of digital regions,
IEEE Trans. Pattern Anal. Machine Intell. **4**, 149-153

with an algorithm that runs in time $O(n)$.

Reconstruction problem: Given a digital line, recover a and b (to the extent that this is possible).

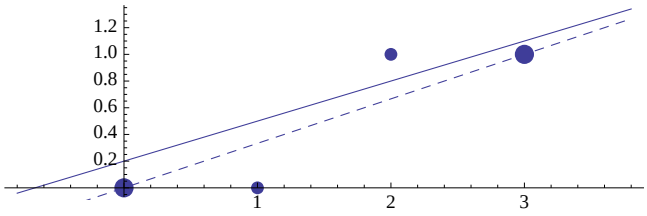
I have not been able to find literature on this.

Quickest way to estimate the slope of a digital line: take the slope of the line joining the two farthest points.

That is, if the leftmost point is (i, j) and the rightmost point is (i', j'), estimate the slope by $\frac{j'-j}{i'-i}$.

The error is likely to be on the order of $1/n$.

Example: $a = .3$, $b = .2$, $n = 4$:



Estimated slope = $\frac{1-0}{3-0} = \frac{1}{3} \approx .3$.

Can we do better job of estimating the slope a of a line $y = ax + b$, given its digitization?

If you're trying to estimate the intercept b, you can't do better than $O(1/n)$.

We can see this using a simple geometry-of numbers-argument: What happens as you gradually increase b by 1, holding the slope a fixed?

The digitization changes at most n times.

Note however that this argument yields a different conclusion if the roles of a and b are reversed.

What happens as you gradually increase a by ϵ, holding the slope b fixed?

The area swept out by the line is $\epsilon\, n^2$.

The digitization changes about $\epsilon\, n^2$ times.

So if you're trying to estimate the slope a, you can't do better than $O(1/n^2)$.

The most accurate way to approach the problem of estimating a and b is linear programming.

Each of the n points on the digital line gives two linear inequalities satisfied by a and b.

Example: $a = .3$, $b = .2$, $n = 4$:
The digital line consists of (0,0), (1,0), (2,1), (3,1).

(0) $-\frac{1}{2} \le 0\, a + b \le \frac{1}{2}$

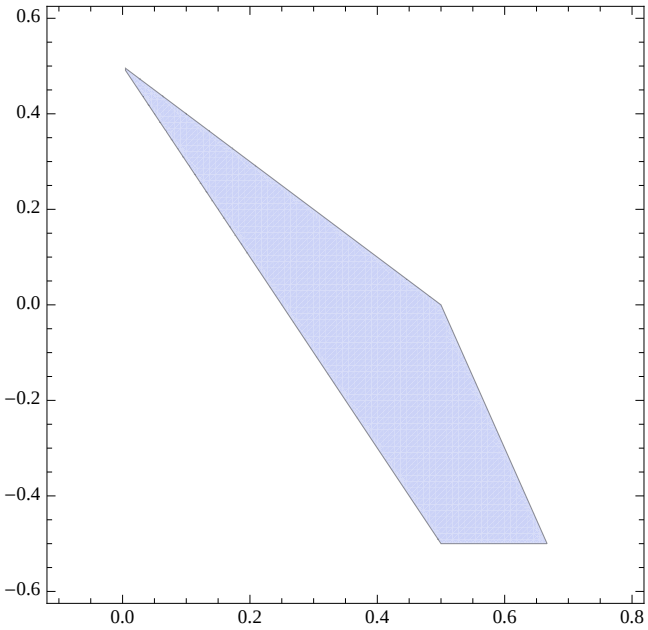
(1) $-\frac{1}{2} \le 1\, a + b \le \frac{1}{2}$

(2) $\frac{1}{2} \le 2\, a + b \le \frac{3}{2}$

(3) $\frac{1}{2} \le 3\, a + b \le \frac{3}{2}$

Example (continued):

```
RegionPlot[-1/2 ≤ 0 a + b && 0 a + b ≤ 1/2 &&
  -1/2 ≤ 1 a + b && 1 a + b ≤ 1/2 && 1/2 ≤ 2 a + b &&
  2 a + b ≤ 3/2 && 1/2 ≤ 3 a + b && 3 a + b ≤ 3/2,
 {a, -.1, .8}, {b, -.6, .6}, PlotPoints → 100]
```



In this case *a* ranges from 0 to $\frac{2}{3}$, and *b* ranges from $-\frac{1}{2}$ to $\frac{1}{2}$.

Question: How long does this linear programming computation take if you do it intelligently?

Question: Does this method achieve the lower bound $O(1/n^2)$?

The next most accurate way is to get interval-estimates for a alone, using pairs of points (i, j), (i', j') on the digital line: subtract

$$j - \frac{1}{2} \le i\, a + b \le j + \frac{1}{2}$$

from

$$j' - \frac{1}{2} \le i'a + b \le j' + \frac{1}{2}$$

to get

$$j' - j - 1 \le (i' - i)\, a \le j' - j + 1.$$

so that a lies within 1 of $\frac{j'-j}{i'-i}$.

Returning to our running Example, we have

$(i, i') = (0, 1)$: $-1 \le 1\, a \le 1 \Rightarrow -1 \le a \le 1$
$(i, i') = (1, 2)$: $0 \le 1\, a \le 2 \Rightarrow 0 \le a \le 2$
$(i, i') = (2, 3)$: $-1 \le 1\, a \le 1 \Rightarrow -1 \le a \le 1$
$(i, i') = (0, 2)$: $0 \le 2\, a \le 2 \Rightarrow 0 \le a \le 1$
$(i, i') = (1, 3)$: $0 \le 2\, a \le 2 \Rightarrow 0 \le a \le 1$
$(i, i') = (0, 3)$: $0 \le 3\, a \le 2 \Rightarrow 0 \le a \le \frac{2}{3}$

So a lies in $[0, \frac{2}{3}]$.

Question: How long does this computation take if you do it intelligently?

Question: Does this always give the same interval for a as the linear program does?

Experiments suggest that this method (intersecting intervals of width 2) gives very tight bounds on *a*:

```
interval[n_, a_, g_] :=
 Min[Flatten[Table[Table[(Round[j a + g] - Round[i a + g] + 1) / (j - i), {j, i + 1, n}], {i, 1, n - 1}]]] -
  Max[Flatten[Table[Table[(Round[j a + g] - Round[i a + g] - 1) / (j - i), {j, i + 1, n}], {i, 1, n - 1}]]]
```

```
Median[Table[N[interval[10, RandomReal[], RandomReal[]]], {100}]]
```

0.125

```
Median[Table[N[interval[100, RandomReal[], RandomReal[]]], {100}]]
```

0.00124275

```
Median[Table[N[interval[1000, RandomReal[], RandomReal[]]], {100}]]
```

0.0000113671

That looks a lot like C/n^2 (with C in the vicinity of 10).

Why I don't like these two approaches:
(a) They take a long time to compute (say for $n = 10^6$).
(b) The assumptions they rely on don't apply in the context that really interests me (rotor-routing).
(c) They are not easy to analyze.

More useful to me is the *least squares method* of estimating the slope *a*, which can be found in the literature as far back as

Melter, R.A., Stojmenovi■, I., and Žuni■, J. (1993),
A new characterization of digital lines by least square fits,
Pattern Recognition Letters **14**, 83-88.

This method has linearity properties that make it especially tractable and useful.

We treat the *n* points on the digital line as a scatter diagram and compute the line of the form $y = \bar{a}x + \bar{b}$ that minimizes the sum of the squares of the vertical displacements of the *n* points with respect to the line.

Example: $a = .3$, $b = .2$, $n = 4$.
Data points: (0, 0), (1, 0), (2, 1), (3, 1).
Points on line: (0, b), (1, a+b), (2, 2a+b), (3, 3a+b).
Vertical displacements: b, a+b, 2a+b−1, 3a+b−1.

```
f = (b)^2 + (a + b)^2 + (2 a + b - 1)^2 + (3 a + b - 1)^2
```

$$(a + b)^2 + (2\,a + b - 1)^2 + (3\,a + b - 1)^2 + b^2$$

```
Solve[{D[f, a] == 0, D[f, b] == 0}, {a, b}]
```

$$\left\{\left\{a \to \frac{2}{5},\ b \to -\frac{1}{10}\right\}\right\}$$

So the best-fit line is $y = .4\,x - .1$, and our estimate of the slope is $\overline{a} = .4$.

More generally:
Data points: (0, p), (1, q), (2, r), (3, s).
Points on line: (0, b), (1, a+b), (2, 2a+b), (3, 3a+b).
Vertical displacements: $b-p$, $a+b-q$, $2a+b-r$, $3a+b-s$.

```
f = (b - p)^2 + (a + b - q)^2 + (2 a + b - r)^2 + (3 a + b - s)^2
```

$(a + b - q)^2 + (2\,a + b - r)^2 + (3\,a + b - s)^2 + (b - p)^2$

```
Solve[{D[f, a] == 0, D[f, b] == 0}, {a, b}]
```

$$\left\{\left\{a \to \frac{1}{10}(-3\,p - q + r + 3\,s),\ b \to \frac{1}{10}(7\,p + 4\,q + r - 2\,s)\right\}\right\}$$

So our estimate of the slope is $\overline{a}$ = ($-3p-q+r+3s$)/10.

More generally, if our data points are
($0, y_0$), (1, y_1), (2, y_2), ..., ($n-1$, y_{n-1})
then $\overline{a}$ is equal to
$-(n-1)y_0 - (n-3)y_1 - (n-5)y_2 \ldots + (n-1)y_{n-1}$
divided by
(n^3-n)/6.

There's a nice interpretation of $\overline{a}$ as a weighted average of all the secant-slope estimators that gives the estimator $\frac{j'-j}{i'-i}$ weight proportional to $i' - i$.
E.g., for $n = 4$:

```
1 ((q – p) / 1) + 1 ((r – q) / 1) + 1 ((s – r) / 1) +
  2 ((r – p) / 2) + 2 ((s – q) / 2) + 3 ((s – p) / 3)
```

$-3\,p - q + r + 3\,s$

We can compute $\overline{a}$ in time $O(n)$ (if we ignore the complexity of arithmetic operations), and the space requirements are quite minimal.

Question: How close to a does $\overline{a}$ tend to be?

Try it:

```
d[n_, a_, b_] :=
 Sum[(-(n - 1) + 2 i) Round[i a + b], {i, 0, n - 1}] / ((n^3 - n) / 6) - a

Table[
 Log[StandardDeviation[Table[N[d[10^k, RandomReal[], RandomReal[]]],
     {1000}]]] / Log[10^k], {k, 1, 6}]
```

{−1.50762, −1.514, −1.61077, −1.58668, −1.43658, −1.71882}

The governing exponent appears to be about −1.5.

```
(* Fix this; what's the precision problem? *)
```

How can this be?

Specifically, if you hold a fixed (assume for simplicity that a is irrational) and vary b, sliding the continuous line up, at some point the digitization changes (when the continuous line passes through a point of the form $(i, j+1/2)$); won't this cause a big jump in $\overline{a}$?

No: the numerator of $\overline{a}$ is

$$-(n-1)y_0 - (n-3)y_1 - (n-5)y_2 \ldots + (n-1)y_{n-1}$$

which changes by $O(n)$ when y_i increases by 1, and the denominator of $\overline{a}$ is $(n^3-n)/6$, so the change in $\overline{a}$ is $O(1/n^2)$.

Theorem: If a is chosen uniformly at random in [0,1], with $b = -\frac{1}{2}$, then the mean-squared of the error (the difference between a and $\overline{a}$) is $\theta(1/n^{3/2})$.

Reformulation: Write

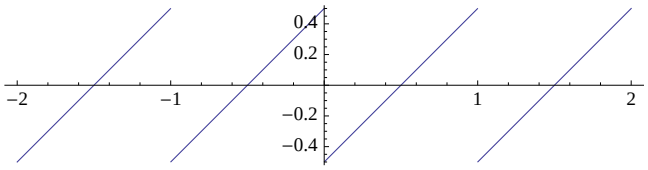
$((x)) = x - \text{floor}(x) - \frac{1}{2}$ if x is not an integer and

$((x)) = 0$ if x is an integer,

so that

$\text{nint}(x) = x - ((x + \frac{1}{2}))$.

```
Plot[x - Floor[x] - 1 / 2, {x, -2, 2}, AspectRatio → Automatic]
```



Then (by way of example)

$$-3\ \text{nint}(0-\tfrac{1}{2}) - 1\ \text{nint}(a-\tfrac{1}{2}) + 1\ \text{nint}(2a-\tfrac{1}{2}) + 3\ \text{nint}(3a-\tfrac{1}{2})$$

$(=10\overline{a})$ is equal to

$$-3\,(0-\tfrac{1}{2}) - 1\,(a-\tfrac{1}{2}) + 1\,(2a-\tfrac{1}{2}) + 3\,(3a-\tfrac{1}{2})$$

$(=10a)$ minus

$$-3\,((0)) - 1\,((a)) + 1\,((2a)) + 3\,((3a)),$$

so that the error $a - \overline{a}$ equals

$$(-3\,((0)) - 1\,((a)) + 1\,((2a)) + 3\,((3a)))/10.$$

Something is wrong here; I'll fix up this section later if I can.

Since ((x)) is periodic with period 1 and has mean 0, the quantity
$d(a) = (-3\,((0)) - 1\,((a)) + 1\,((2a)) + 3\,((3a)))/10$
(as a function of a) also has period 1 and mean 0.
We want to know its squared mean.

```
Sum[(-(4 - 1) + 2 * j) PP[j x], {j, 0, 4 - 1}]
```

$$\lfloor x\rfloor - \lfloor 2x\rfloor + 3\left(-\lfloor 3x\rfloor + 3x - \frac{1}{2}\right) + x + \frac{3}{2}$$

```
Integrate[Out[81], {x, 1, 2}]
```

$$\frac{3}{2}$$

```
Sum[(-(n - 1) + 2 * j) PP[j x], {j, 0, n - 1}] /. n → 4
```

$Aborted

```
PP[3 x]
```

$$-\lfloor 3x\rfloor + 3x - \frac{1}{2}$$

```
Clear[x]
```

```
PP[2]
```

$$-\frac{1}{2}$$

```
d[n_] := Integrate[Sum[-(n - 1) + 2 * j
```

```
Table[n (n - 1) (n + 1) / 6, {n, 1, 7}]
```

{0, 1, 4, 10, 20, 35, 56}

```
Integrate[PP[x] PP[2 x], {x, 0, 2}]
```

$$\frac{1}{12}$$

```
Integrate[((-1 PP[0] + 1 PP[x]) / 1)^2, {x, 0, 1}] * 3^2
```

3

```
Integrate[((-2 PP[0] + 0 PP[x] + 2 PP[2 x]) / 4)^2, {x, 0, 1}] * 4^2
```

$$\frac{4}{3}$$

```
Integrate[((-3 PP[0] - 1 PP[x] + 1 PP[2 x] + 3 PP[3 x]) / 10)^2, {x, 0, 1}] * 5^2
```

$$\frac{3}{4}$$

```
Integrate[((-4 PP[0] - 2 PP[x] + 0 PP[2 x] + 2 PP[3 x] + 4 PP[4 x]) / 20)^2, {x, 0, 1}] * 6^2
```

$$\frac{1}{2}$$

```
Integrate[((-5 PP[0] - 3 PP[x] - 1 PP[2 x] + 1 PP[3 x] + 3 PP[4 x] + 5 PP[5 x]) / 35)^2, {x, 0, 1}] * 7^2
```

$$\frac{653}{1800}$$

```
FactorInteger[653]
```

$$(653 \quad 1)$$

```
Integrate[((-6 PP[0] - 4 PP[x] - 2 PP[2 x] + 0 PP[3 x] + 2 PP[4 x] + 4 PP[5 x] + 6 PP[6 x]) / 56)^2,
  {x, 0, 1}] * 8^2
```

$$\frac{208}{735}$$

```
-3 (0 - 1 / 2) - 1 (a - 1 / 2) + 1 (2 a - 1 / 2) + 3 (3 a - 1 / 2)
```

$$a + 3\left(3a - \frac{1}{2}\right) + \frac{3}{2}$$

```
Simplify[%]
```

$$10\,a$$

```
Integrate[PP[x]
```

```
PP[x_] := x - Floor[x] - 1 / 2
```

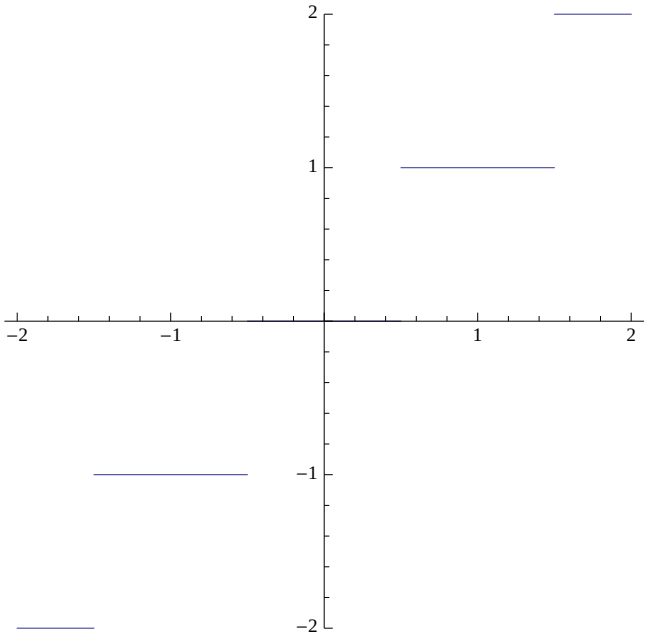
```
0 - PP[0 + 1 / 2]
```

$$0$$

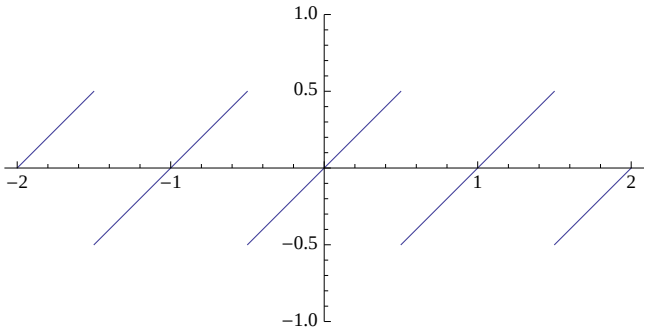
```
1 - PP[1 + 1 / 2]
```

$$1$$

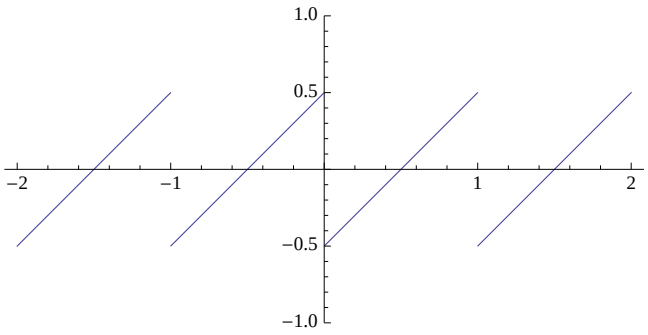
```
Plot[x - PP[x + 1 / 2], {x, -2, 2}, PlotRange → {-2, 2}, AspectRatio → Automatic]
```




```
Plot[x - Round[x], {x, -2, 2}, PlotRange → {-1, 1}, AspectRatio → Automatic]
```




```
Plot[x - Floor[x] - 1 / 2, {x, -2, 2}, PlotRange → {-1, 1}, AspectRatio → Automatic]
```



Problem: Repeat this for the more natural case $b = 0$, and for the case where a and b are both uniformly random in [0,1]. (And what if a and b are governed by some other distribution?)

II. Bit-strings

Recall that we're assuming that the slope a of our digital line is between 0 and 1, so as we go from left to right, the ordinate increases by 0 or 1.

Thus we get a bit-string of length $n - 1$ associated with a digital line $\{(0, y_0), ..., (n-1, y_{n-1})\}$, where the ith bit is

$$\text{nint}(ia+b) - \text{nint}((i-1)a+b) \quad (1 \le i \le n-1)$$

(example: the digital line with ordinates 0,0,1,1 yields the bit-string 0-0, 1-0, 1-1 = **0**, **1**, **0**).
The bit-string associated with a digital line is highly non-random (e.g., the substring **0**, **0**, **1**, **1** never occurs).
More specifically, the bit-string is *balanced*: two substrings of the same length have sums that differ by at most 1.

The bit-strings obtained from digital lines in this way are highly repetitive; and the exceptions to repetitivity are themselves repetitive at a higher scale; etc.

Note that we can reconstruct the digital line (up to an additive constant) from the bit-string by taking partial sums: e.g., **0**, **1**, **0** → 0, **0**, **0**+**1**, **0**+**1**+**0** = 0, 0, 1, 1.

In this context,

a = the slope of the digital line
= the asymptotic density of 1's in the bit-string
= the asymptotic average of the bits.

We're trying to estimate the asymptotic average of a string of bits, given the first $n-1$ bits, on the assumption that the bit-string is balanced.

Surprise: One can do better than simply average the bits with equal weight.

Recall that our least-squares estimator of the slope of the digital line $(0, p)$, $(1, q)$, $(2, r)$, $(3, s)$ is $(-3p-q+r+3s)/10$.
Replacing p, q, r, s by $p, p+x, p+x+y, p+x+y+z$ (where x, y, z are the bits in our bit-string), this becomes $(3x+4y+3z)/10$.

More generally, the least-squares estimator of the asymptotic mean of an infinite balanced bit-string, based on its first n-1 bits x_1, x_2, ..., x_{n-1}, is
$(1)(n-1)x_1+(2)(n-2)x_2+(3)(n-3)x_3+ ...+(n-1)(1)x_{n-1}$
divided by $(n^3-n)/6$.

We may call this the nth "quadratic average" of the initial segment of the bit-string, in contrast to the uniform average $(1)x_1+(1)x_2 +(1)x_3 + ...+(1)x_{n-1}$ divided by $n-1$.

Einstein: If the slope a is rational, the nth quadratic average converges to a with error $O(1/n^2)$.

Matthews: If a has bounded convergents in its continued fraction expansion (e.g., $a = \frac{1}{2}\left(-1+\sqrt{5}\right)$), the nth quadratic average converges to a with error $O((\log n)/n^2)$.

Problem: Show that for almost every a and b, the nth quadratic average converges to a with error $O(1/n^{3/2})$.

III. Almost periodic functions

Problem: Suppose f is almost periodic (a sum of sine-waves with incommensurable periods, plus a constant C). We're given $f(0), f(1), f(2), ..., f(n-1)$.
How do we estimate C?

Simplest interesting case: $f(t) = A + B \sin(\omega t + \phi)$,
where A, B, ω, ϕ are unknown real numbers
(and where in particular there's no reason to think
the period is rational).

To make things conceptually clearer, let's replace A and B by complex numbers and write

$$f(t) = \alpha + \beta \exp(i\omega t)$$

so that

$$f(n) = \alpha + \beta z^n$$

where $z = \exp(i\omega)$ (so that $|z| = 1$). Assume that ω is not a multiple of 2π, so that $z \neq 1$.

First we'll consider the ordinary average:

$(f(0) + f(1) + f(2) + ... + f(n-1))/n$

$= \alpha + \beta\,(1 + z + z^2 + ... + z^{n-1}) / n$

$= \alpha + \beta\,((1 - z^n) / (1 - z)) / n$

$= \alpha + O(1/n)$.

Now we'll consider the quadratic average (where for concision I'll write $(n^3-n)/6$ as T_n):

$((1)(n\text{-}1)\,f(1) + (2)(n\text{-}2)\,f(2) + ... + (n\text{-}1)(1)\,f(n\text{-}1)) / T_n$

$= \alpha + \beta((1)(n\text{-}1)z+(2)(n\text{-}2)z^2+...+(n\text{-}1)(1)z^{n-1}) / T_n$

$= \alpha + \beta(((n\text{-}1)z-(n+1)z^2+(n+1)z^{n+1}-(n\text{-}1)z^{n+2})/(1\text{-}z)^3)/T_n$

$= \alpha + O(1/n^2)$.

If

$$f(t) = \alpha + \Sigma_k\, \beta_k \exp(i\omega_k t)$$

where the β_k's fall off sufficiently quickly as the ω_k's go to infinity (especially for ω_k's that are close to multiples of 2π), then we expect the quadratic weighted average of $f(1)$, ..., $f(n-1)$ to be a better estimate of α (with error $O(1/n^2)$) than the unweighted average (with error $O(1/n)$).

Question: Can this really be new?

IV. Integrals

Suppose f is a continuous periodic function on $\mathbb{R}$ with period 1, written as a function on [0,1). We're given

$f(x_0), f(x_1), f(x_2), ..., f(x_{n-1}),$

where x_i is an arithmetic progression mod 1 with irrational difference.

How do we estimate the integral of f on [0,1) (which can be thought of as the expected value of $f(x)$ if x is chosen uniformly from [0,1))?

The quadratically weighted average

$((1)(n\text{-}1)f(x_1) + (2)(n\text{-}2)f(x_2) + ... + (n\text{-}1)(1)f(x_{n-1})) / T_n$

does better than the uniform average

$(f(x_0) + f(x_1) + f(x_2) + ... + f(x_{n-1})) / n$

though neither does as well as

$(f(0) + f(1/n) + f(2/n) + ... + f((n\text{-}1)/n)) / n.$

This method of integrating can also be applied to integrate functions on $(-\infty,+\infty)$ that fall off at $\pm\infty$ sufficiently quickly (by change of variables), and can also be applied in a multidimensional setting, though it does not appear to be superior to traditional approaches.

V. Markov chains

A random walker starts at (0, 0) and repeatedly takes random unit steps chosen uniformly at random from {(1, 0), (−1, 0), (0, 1), (0, −1)} (independently of the random choices made before) until the walk either arrives at (1,1) ("success") or returns to (0,0) ("failure").

It can be shown that with probability 1, either success or failure will eventually occur.

It can also be shown (with some real work) that the probability p of success (aka the "escape probability") is $\pi/8 \approx .39$.

If we didn't know this, how could we estimate p numerically?

The number of successes in n trials will be

$$pn \pm O(\sqrt{n}).$$

So the proportion of successes in n trials will be

$$p \pm O(1/\sqrt{n}).$$

That is, the "Monte Carlo approach" to determining p will not be feasible if we want 6 digits of accuracy: we'd need 10^{12} trials.

We can reduce the discrepancy in the number of successes in the first n trials from $O(\sqrt{n})$ to $O(\log n)$, and correspondingly reduce the error in our estimate of p from $O(1/\sqrt{n})$ to $O((\log n)/n)$, if instead of doing random walk we do *rotor walk*.

A nice introduction to rotor walk is Michael Kleber's 2005 *Mathematical Intelligencer* article "Goldbug Variations":

http://arxiv.org/pdf/math/0501497v1

To see how the specific random walk described on the previous page gets turned into a rotor walk, go to

http://www.cs.uml.edu/~jpropp/rotor-router-model/

click on "The Applet", and set Graph/Mode to "2-D Walk". Then hit "Step" or "Stage" repeatedly until you get the idea.

We start with a collection of arrows, or rotors, associated with all the sites in $\mathbb{Z}\times\mathbb{Z}$ (except (1,1)); the rotor at site (i, j) points to one of the four neighbors $(i+1, j)$, $(i-1, j)$, $(i, j+1)$, $(i, j-1)$.
When the walker arrives at a site $(i, j) \neq (1,1)$, the rotor at that site advances 90 degrees clockwise, and the walker steps to the neighbor of (i, j) that the rotor currently points to.
When the walker arrives at (1,1), the walker steps immediately to (0,0).

A *stage* or *run* is the process whereby the walker, starting from (0,0), either arrives at (1,1) (without returning to (0,0)) or returns to (0,0) (without visiting (1,1)); we call these *successful* and *unsuccessful* runs, respectfully.

Theorem (Holroyd and Propp, 2010): The number of successes in the first n runs is $pn \pm O(\log n)$ (assuming a particular initial setting of the rotors that keeps the walker from wandering off to infinity).

The motivation behind the present work is the question: **Can we do better**?

Consider the bit-string in which the *i*th bit is a 1 or a 0 according to whether the *i*th run of the "π/8 machine" is a success or a failure.

This bit-string is not balanced (that is, it is not the bit-string associated with a digital line), but it comes close:
for over half the values of n between 1 and 10^4, the number of 1's in the first n bits equals the integer closest to $(\pi/8)n$.

Also, the bit-string has some periodicities of the sort that (according to the examples we've seen in earlier sections) are likely to be conducive to making the quadratic average a good bet.

The easiest periodicity to see in the data is period 8: here are the first 96 bits, arranged in 12 rows of 8:

```
(* Here are the first 10^4 bits;
double-click at right to view. *)
Table[Table[TenK[[8 i + j + 1]], {j, 0, 8}], {i, 0, 12}]
```

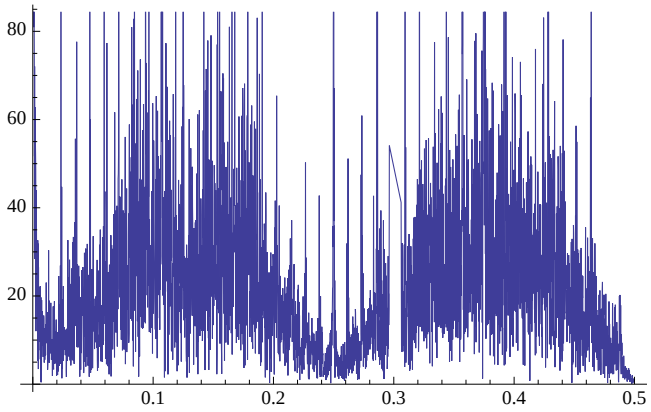
$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

Question: Four of the eight columns are constant as far as they've been computed; why?

To see the other periodicities, we need to take a Fourier transform of the bit-string.

Question: Is there an established theory of Fourier transforms of bit-strings?

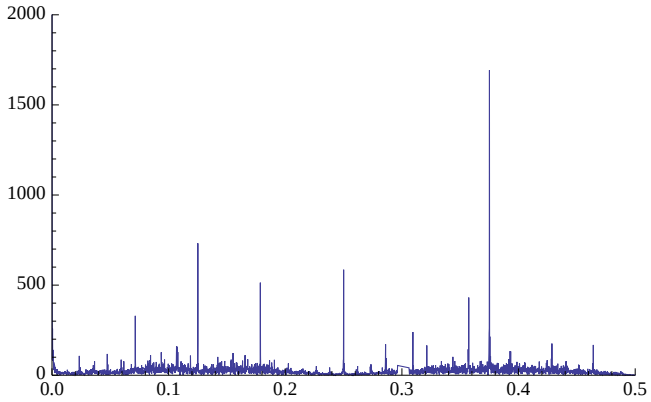
```
e = Abs[Sum[Exp[I 2 Pi t (k - 1)] * TenK[[k]], {k, 1, 10^4}]];
Plot[e, {t, 0, .5}]
```



(The plot from .5 to 1.0 is the same, reversed.)

We can zoom out to see the tallest peaks in this spectrum:

```
Plot[e, {t, 0, .5}, PlotRange → {{0, .5}, {0, 2000}}]
```

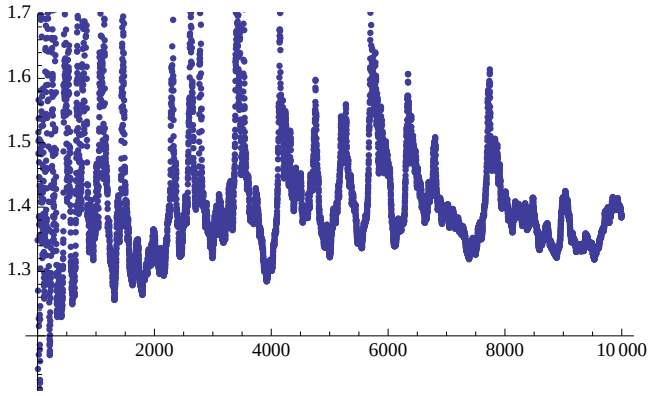


The tallest peaks are the ones at .1250, .2500, and .3750 (no peak at .5000 though).

Also: If we extract every 8th bit, form a new sequence from them, and take its Fourier transform, we get a spectrum whose second-highest peak is around .1415 ≈ $\pi - 3$; coincidence?

Here's what we get if we use quadratic averages to estimate the hitting probability *p*:

```
(* Here are the first 10^4 points on the digital-
  line given by the bit-string. *)
Approx[n_] := Fit[Table[pairs[[i]], {i, 1, n}], {1, x}, {x}][[2]] / x
Expo[n_] := -Log[Abs[Approx[n] - Pi / 8]] / Log[n]
Exponents = Table[Expo[m], {m, 2, 10 000}];
ListPlot[Exponents]
```



It's clear unclear whether the error for quadratic averaging is falling off like $O(1/n^{1.5})$, but it's looking better than $O(1/n)$.

VI. Other directions

The quadratic averaging trick works in the continuous domain as well:

```
Integrate[1 Sin[t], {t, 0, T}] / Integrate[1, {t, 0, T}]
```

```
Integrate[t (T - t) Sin[t], {t, 0, T}] / Integrate[t (T - t), {t, 0, T}]
```

$$\frac{6\,(-T\sin(T) - 2\cos(T) + 2)}{T^3}$$

Hence, if you average an oscillatory term sin ($\omega t + \phi$) out to time T in the ordinary way, you get $O(1/T)$, but with (continuous) quadratic averaging you get $O(1/T^2)$.

(Surely *this* is in the Fourier literature!)

In this setting, we can see that $O(1/T^2)$ is not the end of the line:

```
Integrate[t^2 (T - t)^2 Sin[t], {t, 0, T}] / Integrate[t^2 (T - t)^2, {t, 0, T}]
```

$$\frac{30\left(2\left(T^2-12\right)(\cos(T)-1)-12\,T\sin(T)\right)}{T^5}$$

So with (continuous) quartic averaging you get $O(1/T^3)$.

Indeed, going back to the problem of estimating the average value of an almost periodic function given its value at evenly-spaced discrete times, we see that we can also achieve $O(1/n^3)$ error in that context: If you multiply
$[(1)(n-1)]^2 z+[(2)(n-2)]^2 z^2+...+[(n-1)(1)]^2 z^{n-1}$
by $(1-z)^5$, you get a numerator in which coefficients are all $O(n^2)$, and when you divide by the normalizing constant $[(1)(n-1)]^2+[(2)(n-2)]^2+...+[(n-1)(1)]^2$ which is on the order of n^5, you get something $O(1/n^3)$.

(Overly vague**) Problem**: What is the best way to estimate the average value of an almost periodic function?

Of course this won't give us error $O(1/n^3)$ in the setting of digital lines and bit strings, where the function we're sampling at discrete times takes on discrete values; we already know that $O(1/n^2)$ is the best one can hope for there. But maybe it'll improve on the performance of quadratic averaging ($O(1/n^{1.5})$) and move us closer to the $O(1/n^2)$ limit.

Let's try it and see!:

```
d4[n_, a_, b_] :=
 Sum[(i + 1)^2 (n - i)^2 (Round[(i + 1) a + b] - Round[i a + b]),
    {i, 0, n - 1}] / Sum[(i + 1)^2 (n - i)^2, {i, 0, n - 1}] - a
```

```
Table[
 Log[StandardDeviation[Table[N[d2[10^k, RandomReal[], RandomReal[]]],
     {1000}]]] / Log[10^k], {k, 1, 6}]
```

{−1.55562, −1.51545, −1.47829, −1.51034, −1.38751, −1.69315}

```
(* Should I trust those last two numbers? Try d2! *)
```

```
Table[
 Log[StandardDeviation[Table[N[d2[10^k, RandomReal[], RandomReal[]]],
     {1000}]]] / Log[10^k], {k, 1, 4}]
```

{−1.54536, −1.49444, −1.52075, −1.52169}

It doesn't look like there's any improvement.

Question: Why this difference between the two cases?

Question: Are there computationally efficient ways to get closer to the $O(1/n^2)$ limit?

VII. Summary

When a source of data has built-in periodicity, you shouldn't use a simple uniform average, but rather use an average in which values from the middle are weighted more heavily than samples from the beginning and end.

When you're doing research in one area and find it has connections to other areas you aren't an expert in, consider making use of the friendly experts who populate MathOverflow!