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

James Propp
UMass Lowell
last revised August 22, 2011

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

Slides for this talk are at

http://jamespropp.org/Slope.nb

See

http://mathoverflow.net/questions/23124/
sums-involving-the-nearest-integer-function

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

http://mathoverflow.net/questions/55535/
inferring-the-slope-of-a-digitized-line

to get a sense of how MathOverflow helped me shape my inquiry as it progressed.

In[12]:=

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.

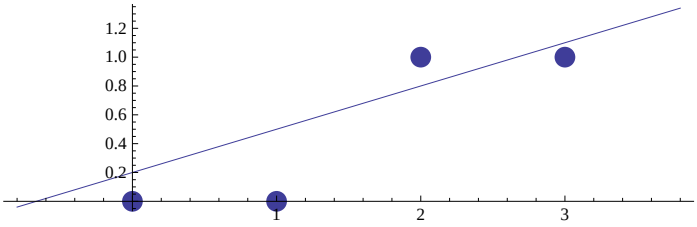
(If $x = k + \frac{1}{2}$ so that the integers $k = x - \frac{1}{2}$ and $k + 1 = x + \frac{1}{2}$ are equally close to x, we take nint(x) = $x - \frac{1}{2} = k$.)

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.

For most of this talk, we'll 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 corresonds 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 $O(n)$ (other solutions appear in the literature as well).

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

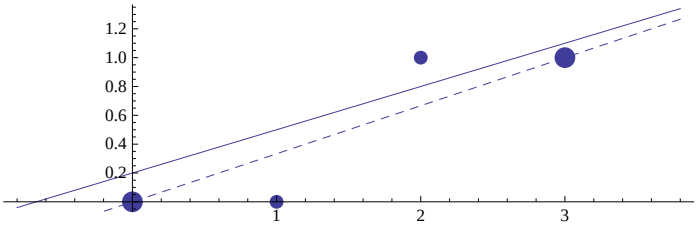
I have found very little literature on this.

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 $\frac{1}{n}$.

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

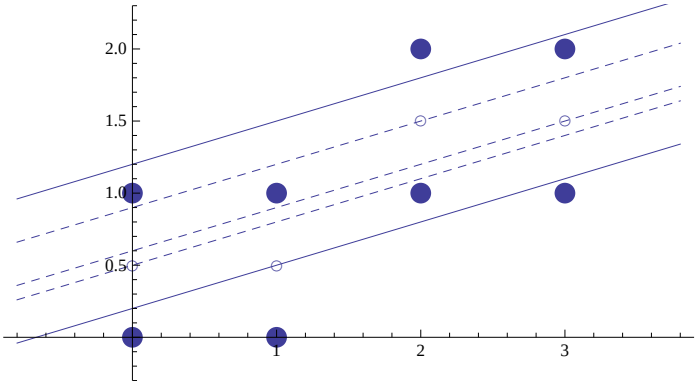


Estimated slope = $\frac{1-0}{3-0} = \frac{1}{3} \approx .3$.

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 $O(1/n)$.

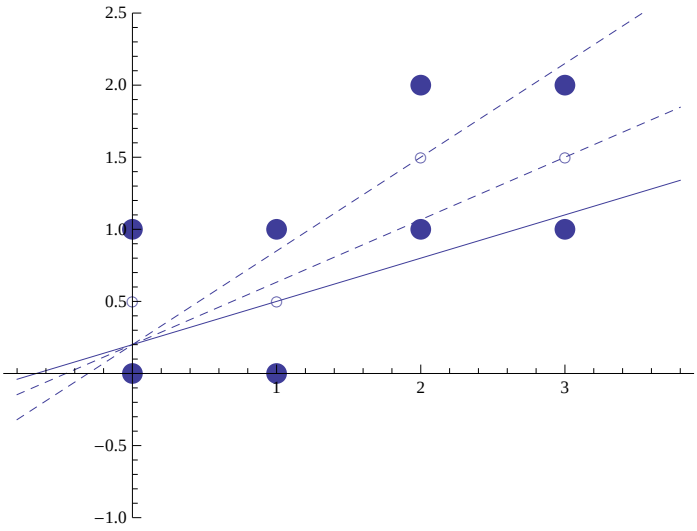
We can see this using a simple geometry-of-numbers-argument: What happens as you gradually increase b by 1, holding the slope a fixed?



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

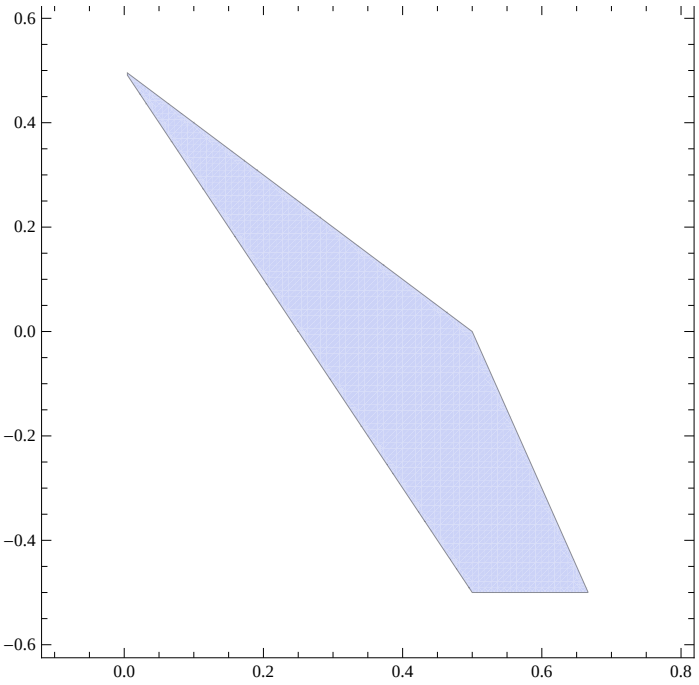
(2) $\frac{1}{2} \le 2\ a + b \le \frac{3}{2}$

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

Example (continued):

In[18]:=
```
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]
```

Out[18]=



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

The feasible region for (a,b) is always a polygon with 4 or fewer sides (Berstel and Pocchiola, 1996).

Experiments suggest that the median width of the interval for a has width about $10/n^2$.

The mean is bigger (because when a is close to a rational number with small denominator the interval tends to have width more like $O(1/n)$ than $O(1/n^2)$) but not horrendously so; we suspect that the mean behaves like $(\log n) / n^2$.

Why the linear programming method of estimating a isn't so suited to my purposes:
(a) It might take a long time to compute (say for $n = 10^6$) (or are there efficient ways to do the computation?).
(b) The assumptions it relies on don't apply in the context that most interests me (rotor-routing).
(c) It is 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., Stojmenović, I., and Žunić, 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 $y = \overline{a}x + \overline{b}$ that minimizes the sum of the squares of the vertical displacements of the n points with respect to the line.

Example (n=4):
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: $b-p$, $a+b-q$, $2a+b-r$, $3a+b-s$.

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

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

In[4]:=
```
Solve[{D[f, a] == 0, D[f, b] == 0}, {a, b}]
```

Out[4]= $\left\{\left\{a \to \frac{1}{10}\,(-3\,p - q + r + 3\,s),\ b \to \frac{7\,p}{10} + \frac{2\,q}{5} + \frac{r}{10} - \frac{s}{5}\right\}\right\}$

So our estimate of the slope is $\overline{a}$ = ($-3p-q+r+3s$)/10.
Applying this to our running example (p=q=0, r=s=1) we get $\overline{a}$ = .4 .

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 $\bar{a}$ 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 $n = 4$:

In[5]:=
```
Simplify[1 ((q - p) / 1) + 1 ((r - q) / 1) +
   1 ((s - r) / 1) + 2 ((r - p) / 2) + 2 ((s - q) / 2) + 3 ((s - p) / 3)]
```

Out[5]= $-3\,p - q + r + 3\,s$

Curiously, if you give the estimator $\frac{j'-j}{i'-i}$ weight proportional to $(i' - i)^2$ you get the same average!

E.g., for $n = 4$:

In[6]:=
```
Simplify[1 ((q - p) / 1) + 1 ((r - q) / 1) +
   1 ((s - r) / 1) + 4 ((r - p) / 2) + 4 ((s - q) / 2) + 9 ((s - p) / 3)]
```

Out[6]= $-6\,p - 2\,q + 2\,r + 6\,s$

The weighting by $(i' - i)^2$ may be more convincing if you're fond of Gaussians: when averaging Gaussians with the same mean, you get the best estimate when weights are inversely proportional to variance.

(**Bibliographic question**: What's a reference for this last fact? It must be standard.)

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:

In[7]:=
```
d[n_, a_, b_] := Sum[(-(n - 1) + 2 i) Round[i a + b], {i, 0, n - 1}] / ((n^3 - n) / 6) - a
```

In[8]:=
```
Table[Log[StandardDeviation[Table[N[d[10^k, RandomReal[], RandomReal[]]],
      {10 000}]]] / Log[10^k], {k, 1, 3}]
```

Out[8]= {−1.49288, −1.50551, −1.50402}

The governing exponent appears to be about −1.5.

Theorem: If a and b are chosen uniformly at random in [0,1], then the root-mean-square error for the slope-estimate $\overline{a}$ derived from the n sample-points

$$\{(i, \text{nint}(ai+b)): 1 \le i \le n\}$$

is $O(1/n^{3/2})$.

Remark: You may be wondering how this is possible. Specifically, if you hold a fixed (assume for simplicity that a is irrational) and vary b, sliding the line upward, at some point the digitization changes (when the line passes through a point of the form $(i, j+1/2)$); won't this cause a big jump in $\overline{a}$? No it won't: 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 $O(1/n^2)$.

Proof idea: If a and b are independent random variables, uniform in [0,1], then the fractional parts of $ai+b$ and $aj+b$ are uncorrelated for $i \ne j$. (In fact, we could take a,b independent with a uniform in $[0,m]$ and b uniform in $[0,n]$ for arbitrary positive integers m,n, and get root-mean-square error $O(1/n^{3/2})$ by the same argument.)

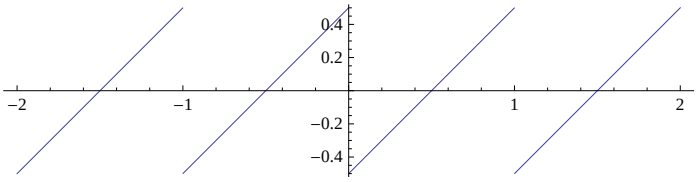
Proof: We will prove an equivalent result, where nint($ai+b$) is replaced by nint($ai+b-\frac{1}{2}$); since the error only depends on b mod 1, and since b varies uniformly over [0,1] mod 1 iff $b-\frac{1}{2}$ does, this doesn't change anything.

Also, define

$$\{x\} = \begin{cases} x - \lfloor x \rfloor - \frac{1}{2} & x \notin \mathbb{Z} \\ 0 & x \in \mathbb{Z} \end{cases}$$

so that

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



Then (by way of example with $n = 4$)

$$- 3\ \mathrm{nint}(1a+b-\tfrac{1}{2}) - 1\ \mathrm{nint}(2a+b-\tfrac{1}{2})$$

$$+ 1\ \mathrm{nint}(3a+b-\tfrac{1}{2}) + 3\ \mathrm{nint}(4a+b-\tfrac{1}{2})$$

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

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

$$+ 1\ (3a+b-\tfrac{1}{2}) + 3\ (4a+b-\tfrac{1}{2})$$

$(=10a)$ minus

$-3\ \{a+b\} - 1\ \{2a+b\} + 1\ \{3a+b\} + 3\ \{4a+b\}$,

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

$(-3\ \{a+b\} - 1\ \{2a+b\} + 1\ \{3a+b\} + 3\ \{4a+b\})/10$.

This function of a and b (which we can think of as a random variable) is easily seen to have mean 0 over $[0,1] \lozenge [0,1]$; to find its variance, we integrate its square.

All the cross-terms vanish when we integrate, while all the diagonal terms integrate to $\frac{1}{12}$.

So the variance of the error

$(-3\ \{a+b\} - 1\ \{2a+b\} + 1\ \{3a+b\} + 3\ \{4a+b\})/10$

is equal to $((9+1+1+9)\frac{1}{12}) / 10^2 = 1/60$.

More generally, we get variance $1/(n^3-n)$, so the standard deviation is $\Theta(1/n^{3/2})$. ■

In fact, an easy application of Lagrange multipliers shows that among all the estimators of a that are linear combinations of the y-coordinates of the points on the digital line, $\overline{a}$ as defined above is the **unique choice** that **minimizes the mean-squared error** (when a and b are chosen uniformly at random in [0,1]).

(**Open question**: What if we restrict to $b = 0$?)

But where's the "averaging kernel"? ...

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
$(3x+4y+3z)/10$.

In this weighted average, the middle bit y has greater weight than x and z.

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 $n-1$.

The earlier Theorem implies that for a and b chosen uniformly at random in [0,1], the nth quadratic average usually differs from a by about $1/n^{3/2}$.

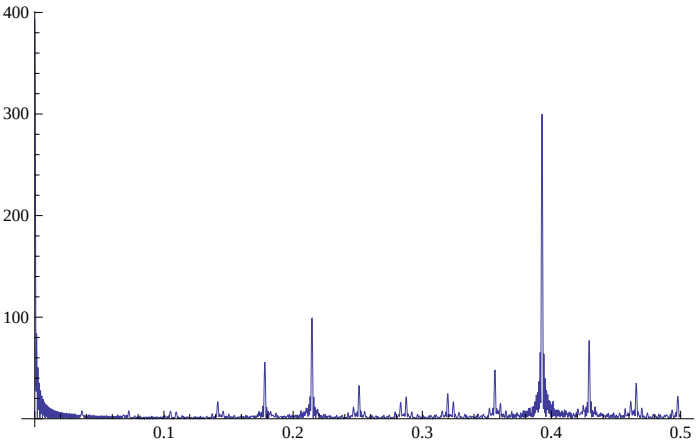
It should be stressed that balanced bit strings are rather special. One way to see this is to take a Fourier transform.

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

Here is the Fourier transform of a balanced (i.e., Sturmian) bit-string:

In[9]:=
```
Plot[Abs[Sum[Exp[I 2 Pi t (k - 1)] * (Round[(k + 1) Pi / 8] - Round[k Pi / 8]), {k, 1, 10^3}]],
 {t, 0, 0.5}, PlotRange → Full]
```

Out[9]=



Compare this with what we get with a pseudorandom bit-string:

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 (the mean value of f)?

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):

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

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

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

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

So if $f(t)$ is given by an infinite sum

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

where the numbers $\beta_k/(1- \exp(i\omega_k))$ are absolutely summable, then the quadratic weighted average of $f(1)$, ..., $f(n-1)$ is an asymptotically 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
$((1)(n-1)f(x_1) + (2)(n-2)f(x_2) + ... + (n-1)(1)f(x_{n-1})) / T_n$
(typical error $O(1/n^2)$) does better than the uniform average
$(f(x_0) + f(x_1) + f(x_2) + ... + f(x_{n-1})) / n$
(typical error $O(1/n)$), though neither does as well as
$(f(0) + f(1/n) + f(2/n) + ... + f((n-1)/n)) / n$.

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 (the error looks to be $O(1/n^{1+1/d})$).

V. How many dots in a digital disk?

The number of lattice points in a disk of radius $\sqrt{n}$ is about πn, so π is close to the number of lattice points in the disk divided by n; the error is on the order of n^c where $c \approx -.6$ (finding c exactly is Gauss' Circle Problem, still unsolved).

Suppose we use a quadratic weighting instead of a uniform weighting, and give the lattice point (i, j) weight $n - i^2 - j^2$ if (i, j) is in the disk of radius $\sqrt{n}$ and weight 0 otherwise.

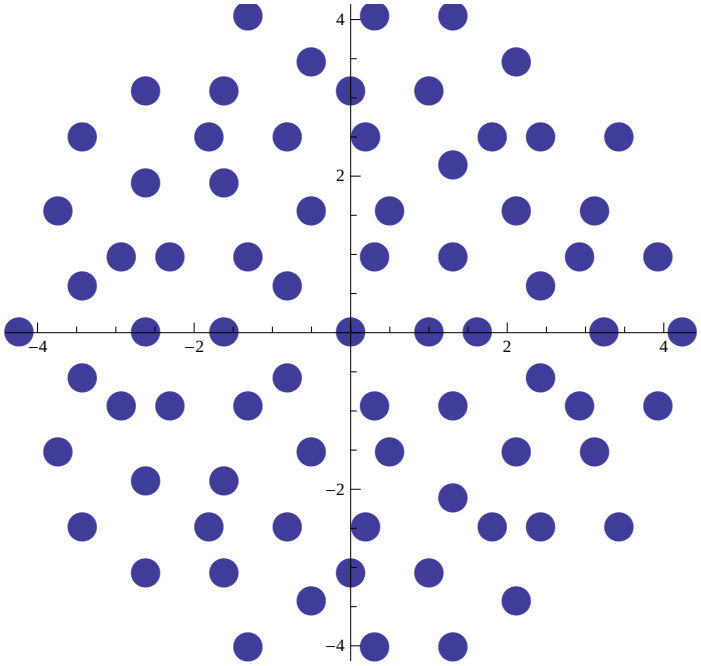
If we take the sum of the weights of the lattice points and divide by the integral of the weighting function (a quadratic function that vanishes on the circle of radius n), we get an approximation to π whose error seems to be about $n^{-1.2}$.

To put this exponent into perspective, note that in the context of counting lattice points in a disk of radius π and area πn, a single lattice point contributes relative error on the order of n^{-1}. So a relative error of $n^{-1.2}$ is less than a lattice point's worth of error.

An equivalent way to describe this approximation to π is as the sum of $(n-k)\ r_2(k)$ with k going from 0 to n divided by $n^2/2$, where $r_2(k)$ denotes the number of ways to write k as a sum of two squares. That is, it's a kind of Cesaro average of $r_2(k)$, and many of the other examples from this talk can be viewed as variants of Cesaro averaging.

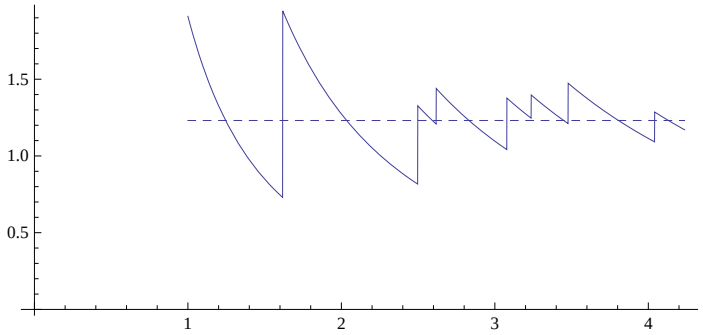
VI. A(lmost)periodic point sets

One version of the 5-fold symmetric Penrose tiling of the plane is a tiling composed of 2 kinds of rhombuses. Here is a patch of this tiling, showing the vertices of the rhombuses.

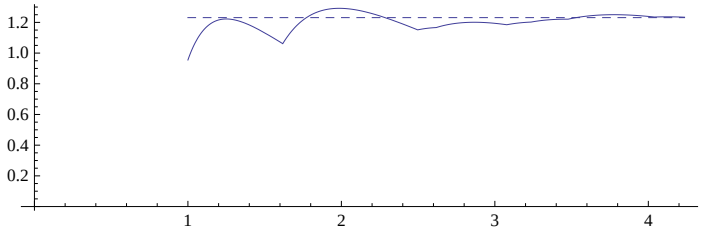


How can we best estimate the density of dots empirically? (Its actual value is $2\sqrt{2}\left(3+\sqrt{5}\right)$ divided by $2\sqrt{5-\sqrt{5}}+\left(1+\sqrt{5}\right)\sqrt{5+\sqrt{5}}$, or about 1.231, but suppose we didn't know this.)

If we count the dots that occur in a disk of radius R, and divide by πR^2, we get a highly discontinuous function that converges to the true density rather slowly:

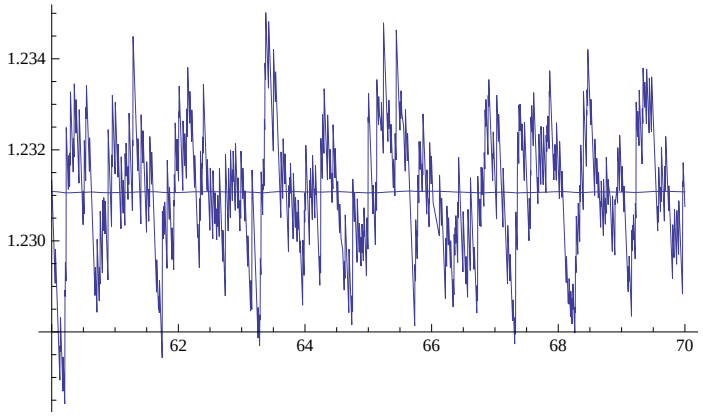


But if we give each dot that occurs at distance r from the origin weight $\max(R - r, 0)$ (a linear weighting function), we get faster convergence:



E.g., the error near $R = 4$ drops by a factor of 15.

Here are the results of a larger calculation, using the 18,946 dots within distance 70 of the origin (computed using *Mathematica* code provided by Uwe Grimm). The following picture shows the uniform-weighting estimate of the density (as a function of R for R between 60 and 70), overlaid with the linear-weighting estimate of the density.



In this range, the linear-weighting estimate is better by a factor of 200.

VII. Markov chains and rotor walk

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

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

A more recent article with a more explicit theory-of-computation flavor is my 2010 *Chaos* article "Discrete analogue computing with rotor-routers":

http://arxiv.org/abs/1007.2389

One motivation for work on rotor-routing is that it's a general trick for derandomizing the simulation approach for many probabilistic quantities: escape probability, stationary measure, mean hitting time, etc.

I'll focus on a single application involving an escape probability associated with a two-dimensional random walk.

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*.

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 $\mathbb{Z}\Diamond\mathbb{Z}$ (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* vs. *unsuccessful* runs.

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**? That is, can we make better use of the rotor walk runs to get a quantity that concentrates more tightly around p?

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 "$\pi/8$ machine" is a success or a failure.

This bit-string (the "escape sequence") 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. *)
```

In[29]:=
```
Table[Table[TenK[[8 i + j + 1]], {j, 0, 7}], {i, 0, 12}]
```

Out[29]=

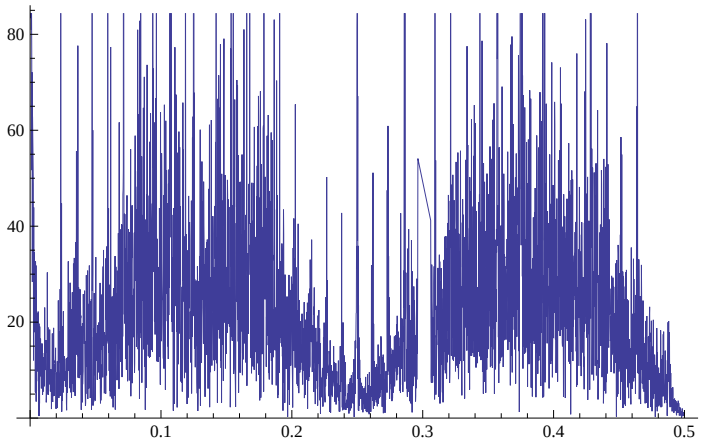
$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 &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

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.

In[30]:=
```
Plot[Abs[Sum[Exp[I 2 Pi t (k - 1)] * TenK[[k]], {k, 1, 10^4}]], {t, 0, .5}]
```

Out[30]=

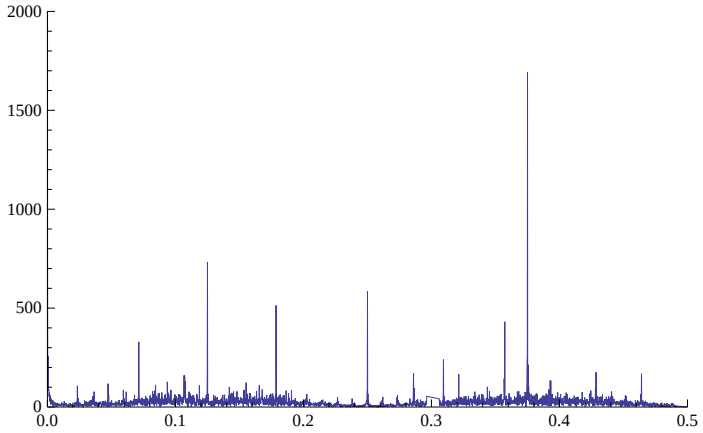


(The plot from .5 to 1.0 is the same, reversed.)

It looks like there's some suppression of peaks near quarter-integers, and to a lesser extent near 1/8, 3/8, 5/8, 7/8.

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

In[32]:=
```
e = Abs[Sum[Exp[I 2 Pi t (k - 1)] * TenK[[k]], {k, 1, 10^4}]];
Plot[e, {t, 0, .5}, PlotRange → {{0, .5}, {0, 2000}}]
```

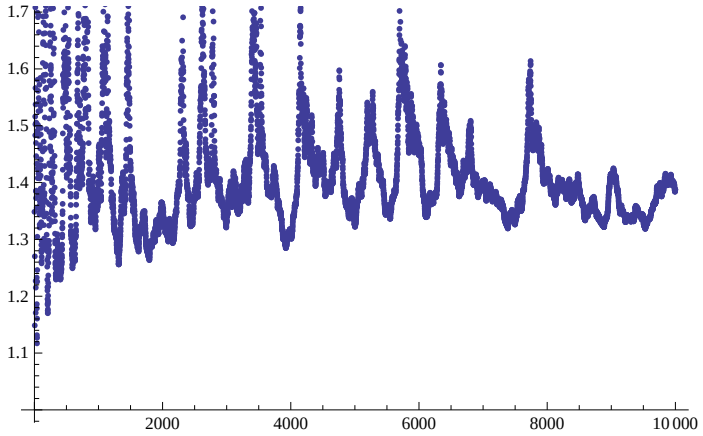


The tallest peaks are the ones at .1250, .2500, and .3750 (no peak at .5000 though).

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 $\approx \pi - 3$; coincidence?

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

```
(* Here are the first 10^4 points on the digital-
  line given by the bit-string. *)
ListPlot[Exponents, AxesOrigin → {1, 1.0}]
```



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

The underlying question is:

Question: How well can one see the asymptotic average of a rotor-router escape sequence, given its first n terms?

David Einstein showed that for finite rotor-router machines, quadratic averaging achieves error $O(1/n^2)$, and I know of one example of an infinite (the “goldbug machine”) for which quadratic averaging achieves error $O((\log n)/n^2)$.

I have very few ideas about how to prove anything about quadratic averaging for escape sequences for rotor-routing on general infinite graphs, but here's another problem I can't solve whose solution would probably require techniques that would in turn be useful for rotor-routing escape sequences:

Question: How well can you see the slope of a "composite digital line" $y = \text{nint}(a' \text{ nint}(a\,x + b) + b')$?

Does quadratic averaging let one achieve error $O(1/n^{3/2})$ when a, b, a', b' are chosen randomly?

From a pure math perspective, it seems natural to take the closure of the set of digital linear functions under addition and composition, but I haven't found any literature on this.

VIII. Other directions

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

In[42]:=
```
Integrate[1 Sin[t], {t, 0, T}] / Integrate[1, {t, 0, T}]
```

Out[42]=
$$\frac{1-\cos(T)}{T}$$

In[43]:=
```
Integrate[t (T - t) Sin[t], {t, 0, T}] / Integrate[t (T - t), {t, 0, T}]
```

Out[43]=
$$\frac{6\,(-T\sin(T)-2\cos(T)+2)}{T^3}$$

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 $O(1/T^2)$.

Here the averaging kernel is an integration kernel rather than a summation kernel. 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:

In[44]:=
```
Integrate[t^2 (T - t)^2 Sin[t], {t, 0, T}] / Integrate[t^2 (T - t)^2, {t, 0, T}]
```

Out[44]=
$$\frac{1}{T^5} 30 \left(2 \left(T^2 - 12\right) (\cos(T) - 1) - 12\, T \sin(T)\right)$$

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

Indeed, going back to the problem of estimating the mean 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
$[(1)(n-1)]^2z+[(2)(n-2)]^2z^2+...+[(n-1)(1)]^2z^{n-1}$
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 that's $O(1/n^3)$.

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

IX. Summary

When a source of data has built-in periodicity, and you want to compute its long-term average from data in a finite tine-window, 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.

Also: 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!

Slides for this talk are at

http://jamespropp.org/Slope.nb