Reliable real arithmetic

and

one-dimensional dynamical systems

Keith Briggs, BT Research

Keith.Briggs@bt.com

`http://keithbriggs.info`

QMUL Mathematics Department 2007 March 27 1600

TYPESET 2007 MARCH 26 14:44 IN PDFLATEX

Arithmetic

- can we compute with elements of $\mathbb{R}$ (an infinite complete ordered field) on a machine with only finite resources?
  - *answer: to some extent*
- we have to give up something - computability, accuracy, ordering, . . .
  - *computability - some things may in principle be uncomputable*
  - *accuracy - we may incur a small error in each operation, and these errors may accumulate*
  - *ordering - if x and y are computed results, we may no longer be able to determine whether $x < y$ or not*
- I claim that for work in dynamical systems, we do *not* want to give up the ordering property
  - *. . . but little attention has been paid to software with this property*

Floating point with Maple

- example: $\sin(6303769153620408 \times 2^{971})$
- Maple 9 & 10 `evalf[digits](sin(6303769153620408*2^971));`:
  - `evalf[ 10](...)` *gives -0.8021127471*
  - `evalf[ 20](...)` *gives -0.9482478427...*
  - `evalf[ 50](...)` *gives 0.3915937923...*
  - `evalf[200](...)` *gives -0.3887412074...*
- not a bug!
- hysteresis!!
- with `mpfs` (`http://keithbriggs.info/mpfs.html`):

```
sin(6303769153620408*2^971)=0.1600997259 +or- 9.8e-29
sin(6303769153620408*2^971)>0?: True
```

Computer arithmetic - a layered model

hardware:

- 0: fixed-size integers
- 1: floating point

software:

- 2: arbitrary-size integers (`GMP`)
- 3: double or quad size floats (e.g. `doubledouble`)
- 4: arbitrary (but fixed) size floats with exact rounding (`MPFR`)
- 5: arbitrary (but fixed) size floats with error propagation (`mpfre`)
- 6: dynamically-sized floats with automatic recomputation (various strategies exist; e.g. `mpfs`)

Computer arithmetic - hardware layer

- integers: typically 32 or 64 bits, $+, -, *$ exact if result in range
- floating point: typically 53-bit mantissa, 11-bit exponent
- IEEE 754 property (round-to-nearest): if $\circ \in \{+, -, *, /\}$, then fl$(x \circ y) = (x \circ y)(1+\delta)$, where $|\delta| < u$, $u = 2^{-p}$ ($p =$ precision in bits, subject to no underflow or overflow)
- . . . in other, words, the relative error is bounded
- even more useful: fl$(x \circ y) = (x \circ y)/(1+\delta)$, $|\delta| \leqslant u$
- we can in principle propagate errors using these formulas, by computing bounds on the absolute error for each step
- . . . but in practice it's difficult and slow as we have to constantly switch the rounding mode

Computer arithmetic - software integers and floats

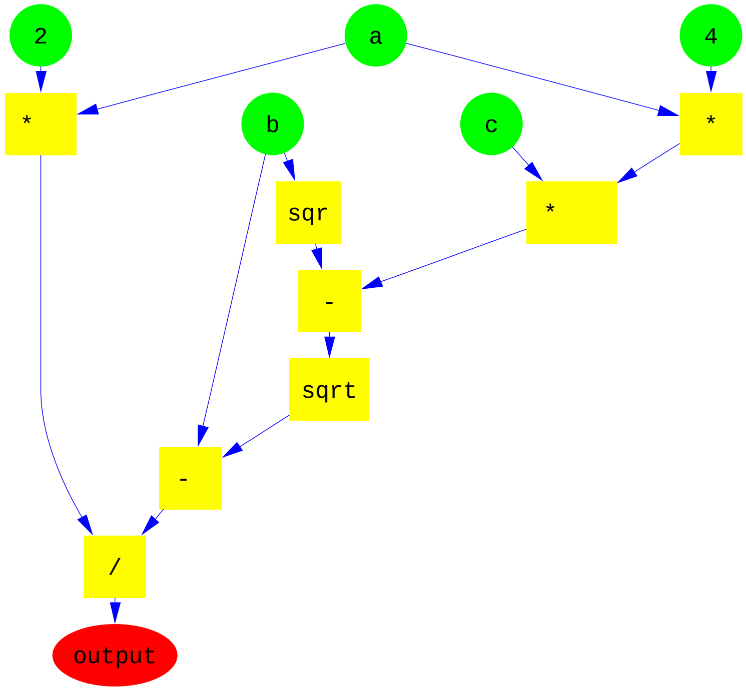
- integers are easy to represent, but it's hard to design fast algorithms
- multiplication: for n bits, the simple algorithm takes $\mathcal{O}(n^2)$ time; Karatsuba is $\mathcal{O}(n^{\log(3)/\log(2)})$; Toom 3-way is $\mathcal{O}(n^{\log(5)/\log(3)})$; FFT is $\mathcal{O}(n^{\log(k)/\log(k-1)}), k=3,4,5,\dots$; faster methods (Bernstein) . . . ?
- NB: above estimates determine the ultimate efficiency of *every-thing* we do
- state-of-the-art: GMP `http://gmplib.org`
- floating-point (correctly rounded to nearest, up or down): MPFR `http://www.mpfr.org` (builds on GMP)

Error propagation

- definitions: exact number $\check{x}$, computed approximation x, error bound $\delta_x \geqslant |x-\check{x}|$, mantissa precision p_x, exponent e_x, unit roundoff $u_x = 2^{e_x-p_x}$ (for $x \neq 0$), $\mathcal{N}$=round-to-nearest
- $z=\mathcal{N}(x\pm y)$: $\delta_z \leqslant u_z/2+\delta_x+\delta_y$
- $z=\mathcal{N}(x*y)$: $\delta_z \leqslant u_z/2+(1+\delta_y 2^{-e_y})|y|\delta_x+(1+\delta_x 2^{-e_x})|x|\delta_y$
- $z=\mathcal{N}(x^{1/2})$: $\delta_z \leqslant u_z/2+\delta_x/(x^{1/2}(1+(1-2^{1-p_x})^{1/2}))$
- $z=\mathcal{N}(\log(x))$: $\delta_z \leqslant u_z/2+\delta_x/x$
- and so on - difficult cases include asin near ± 1
- implemented in an `mpfre` layer: floating point real + error bound
- typically 32 bits are enough for the error bound
- tradeoffs are possible here - tighter bounds need more computing

Data flow example

★ find the sign of one root $x = \frac{-b+\sqrt{b^2-4ac}}{2a}$ of the quadratic $ax^2+bx+c=0$



▷ *input nodes don't know how much precision to send*

▷ *all input nodes send data, even if it eventually may not be needed*

▷ *to recalculate requires the whole tree to be re-evaluated*

mpfs `keithbriggs.info/mpfs.html`

- The method is a stochastic variation of exact real arithmetic. The novelty is a way to avoid the 1-bit graininess
- the complete DAG is stored
- nodes cache the most precise value so far computed
- lazy: compute-on-demand
  - *heuristic: when output error bounds are too big (e.g. to decide an inequality), we add precision to intermediate steps in random places, and recompute*
  - *cf. TCP*
- the intermediate value at each node is a floating-point approximation x and absolute error bound δ. The interval $[x-\delta, x+\delta]$ always strictly bounds the correct result, and will be refined automatically as necessary
- user interface design: hide all internals, so that user just calls functions in the normal way

Classical theory of continued fractions

- regular continued fractions are symbolic dynamics of the Gauss map:
$$g(x) = 1/x - \lfloor 1/x \rfloor \quad \text{for} \quad x \in (0,1]$$
where the digit x_k (partial quotient) output at the kth iteration is $\lfloor 1/x \rfloor$
- we write $x = [x_1, x_2, x_3, \ldots]$, where $x_k \in \{1, 2, 3, \ldots\}$
- the continued fraction is finite iff x is rational
- for almost all x, the digit i occurs with relative frequency $\mu(i) \equiv \log_2 \left[\frac{(i+1)^2}{i(i+2)}\right]$
- the continued fraction is eventually periodic iff x is a quadratic irrational

An algorithm for continued fractions 1

- motivation - statistical studies of distribution of partial quotients
- many strategies are possible, but those offering guaranteed output are typically much slower than otherwise
- consider first rationals $n/d > 1$:

  while $d > 0$ do
  $q \leftarrow \lfloor n/d \rfloor$
  output partial quotient q
  $(n, d) \leftarrow (d, n - qd)$
  end while

- most time is spent in the division step
- we should be able to speed this up, as we know that usually q is small

Doing the division fast

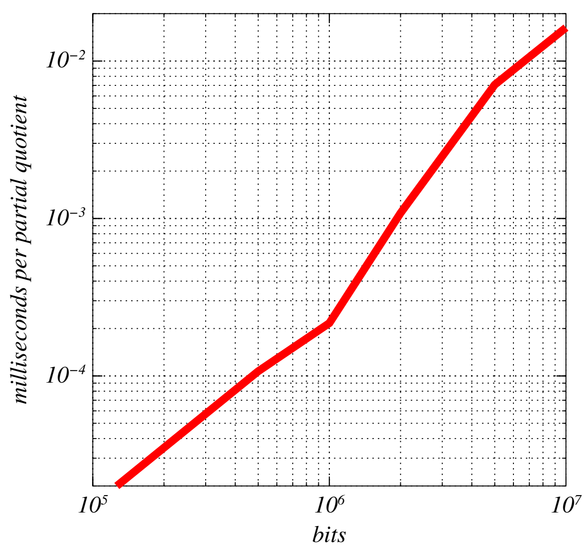
- ★ definition: a *limb* is a word (block of 32 or 64 bits in a hardware integer) of a multi-word integer
- ★ theorem (Zimmermann): Divide the 2 most significant limbs of n by the most significant limb of d. This will yield either $q, q+1$ or $q+2$ where q is the exact quotient
- ★ theorem (Granlund and Möller): Divide the 3 ms limbs of n by the 2 ms limbs of d. This yields q with very high probability ($q+1$ is produced only with probability 2^{-32} or 2^{-64})
- ★ Correcting the result:

  while $n-qd<0$ do
  $\quad q \leftarrow q-1$
  end while

- ★ NB: for our continued fraction application, we will abort if q is greater than the word size. If we do not abort, the only multi-word operation is the multiplication in $n-qd$. But this is linear in the size of d. Thus, everything is fast.

An algorithm for continued fractions 2

- for an arbitrary computable irrational x, I construct rational lower (n/d) and upper (n'/d') bounds using `mpfr`, by rounding down and up and extracting the mantissa and exponent
- I then compute the continued fraction of (n/d) and (n'/d'), using the above algorithm. As long as the partial quotients agree, they are the correct partial quotients of x
- typical scaling of the running time:



More theory

★ [8, p226] gives a formula for relative frequency of the m-block $i=(i_1,i_2,\ldots,i_m)$ which holds as $n\to\infty$ for almost all irrationals:

$$\text{card}\{\kappa : (x_\kappa,\ldots,x_{\kappa+m-1})=i\ ,1\leqslant\kappa\leqslant n\}/n = \log_2\left[\frac{1+v(i)}{1+u(i)}\right]+o\left(n^{-1/2}\log^{(3+\epsilon)/2}(n)\right)$$

$\forall\epsilon>0$ and where (with $[i]=p_m/q_m$ for the m-block i)

$$u(i)=\begin{cases}\frac{p_m+p_{m-1}}{q_m+q_{m-1}} & \text{if } m \text{ is odd}\\ \frac{p_m}{q_m} & \text{if } m \text{ is even}\end{cases}$$

$$v(i)=\begin{cases}\frac{p_m}{q_m} & \text{if } m \text{ is odd}\\ \frac{p_m+p_{m-1}}{q_m+q_{m-1}} & \text{if } m \text{ is even}\end{cases}$$

Yet more theory

- note that $\mu(i)$ is unchanged if we reverse the block i, whatever the length. I do not know what other symmetries exist
- if a particular x has all blocks occurring with these expected frequencies, we call x *normal*
- note that because of the rapid decay of correlations (approximately $(-0.3)^n$ at lag n), there is not much point in studying very long blocks ($n > 5$, say). For long blocks, the two ends are effectively independent. This makes an empirical study such as the present one feasible
- of course, we can never *prove* abnormality (if it exists) merely by a statistical analysis of a finite portion of the infinite continued fraction. However, we might hope to find *evidence* of abnormality, which can then be proven by other methods

Literature survey

- ⋆ [2] examines the frequency of digits amongst the first 1000 of several cubic irrationals
- ⋆ [3] examines the frequency of digits amongst the first 200000 of several algebraic irrationals
- ⋆ none of the above papers find any evidence of abnormality amongst the numbers examined
- ⋆ in [7], we have the result

$$\text{Pr}\left[x_n = r \;\&\; x_{n+k} = s\right] = \text{Pr}\left[x_n = r\right]\text{Pr}\left[x_{n+k} = s\right]\left(1+O(q^k)\right),$$

  where $q \approx -0.303663$ is the Gauss-Kuzmin-Wirsing constant. But this is too weak to allow explicit statistical tests

- ⋆ no papers look at the distribution of blocks of length > 1

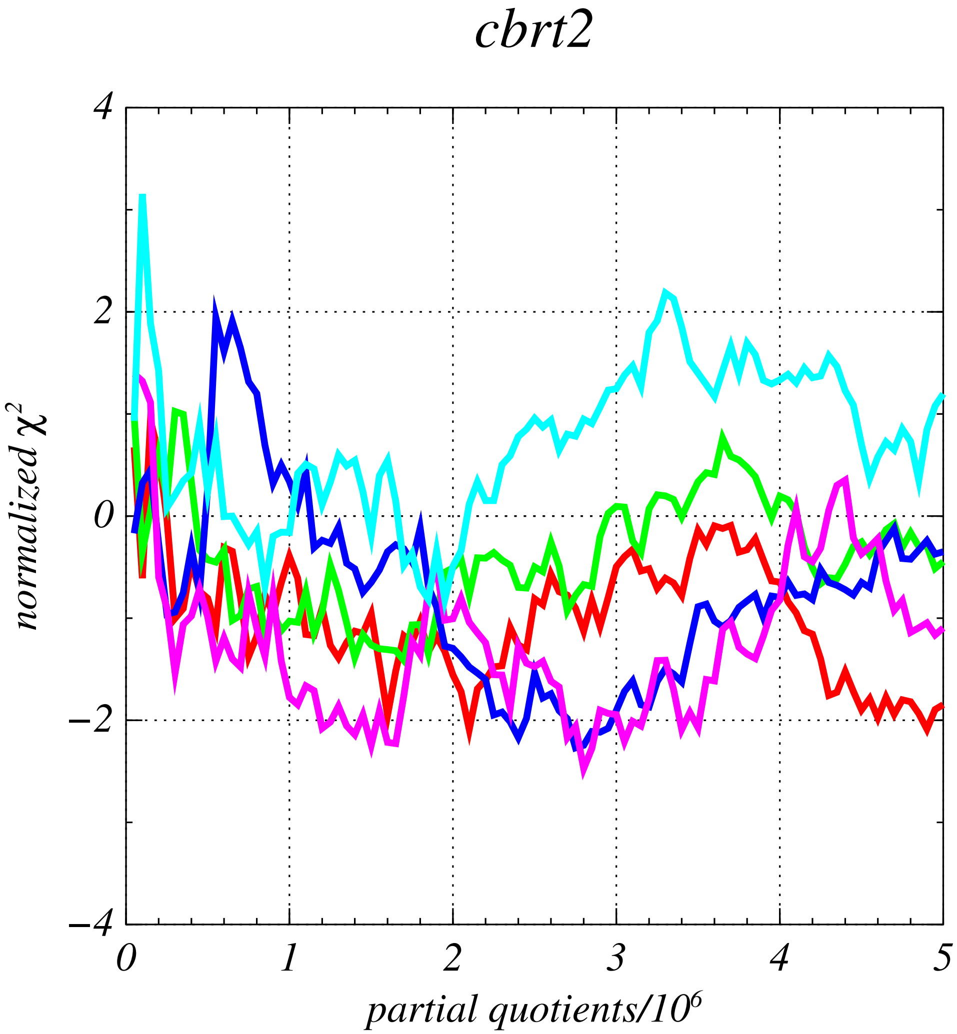
Explicit examples of abnormal numbers

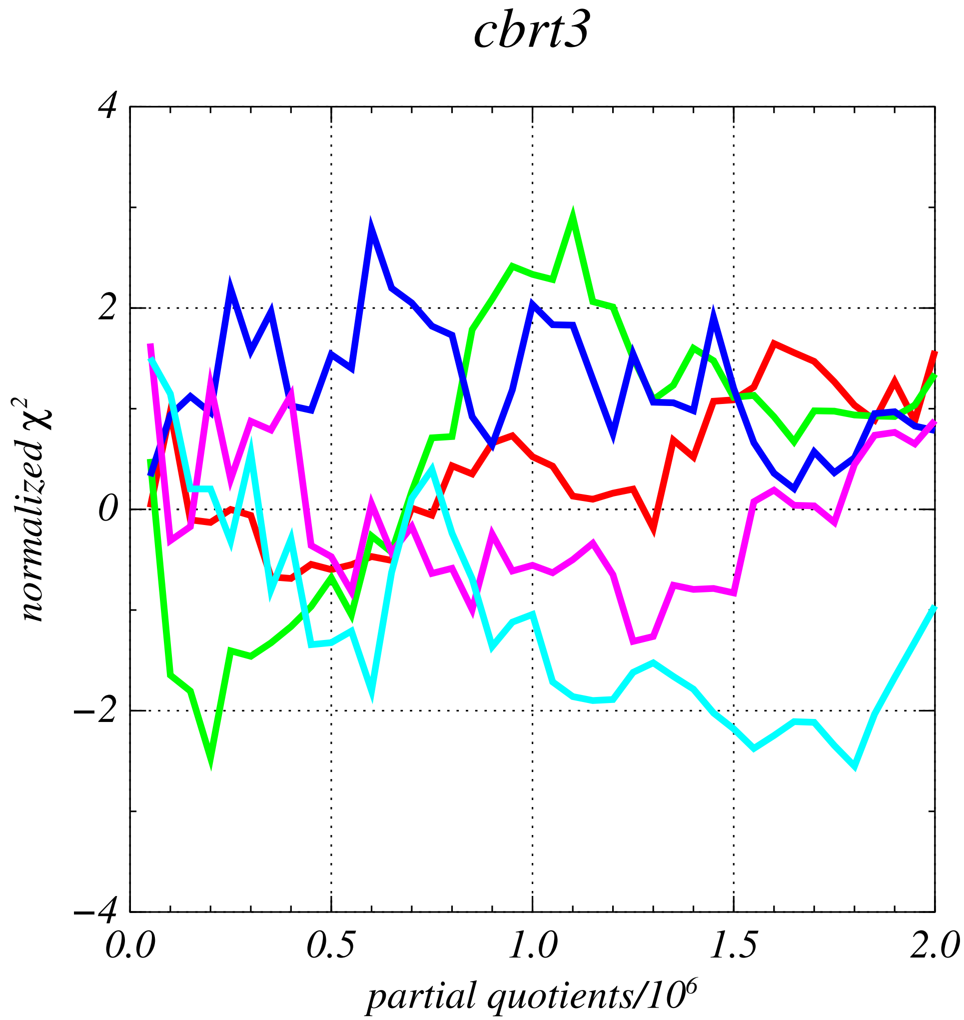
- all quadratic irrationals, e.g. $2^{1/2} = 1+[2, 2, 2, 2, \ldots]$
- $\mathsf{I}_1(2)/\,\mathsf{I}_0(2) = [1, 2, 3, 4, \ldots]$ (ratio of modified Bessel functions)
- $\mathsf{I}_{1+a/d}(2/d)/\,\mathsf{I}_{a/d}(2/d) = [a+d, a+2d, a+3d, \ldots]$
- $\tanh(1) = [1, 3, 5, 7, \ldots]$
- $\exp(1/n) = [1, n-1, 1, 1, 3n-1, 1, 1, 5n-1, \ldots];\ n = 1, 2, 3\ldots$
- $\exp(2) = 7+[2, 1, 1, 3, 18, 5, 1, 1, 6, 30, 8, 1, 1, 9, 42, 11, 1, 1, 12, 54, \ldots]$
- $\exp(2/(2n+1));\ n = 1, 2, 3\ldots$
- $\sum_{k=1}^{\infty} 2^{-\lfloor k\phi\rfloor} = [2^0, 2^1, 2^1, 2^3, 2^5, 2^8, 2^{13}, \ldots];\ \phi = (\sqrt{5}-1)/2$

Method

- ⋆ I calculate a few million digits for several cubic irrationals and a few other irrationals
- ⋆ I count exactly the observed frequency of all blocks of lengths 1,2,3,4,5
- ⋆ I calculate a Pearson χ^2 test statistic which measures the deviation of the observed frequencies from the expected frequencies
- ⋆ because the number of degrees of freedom ν is so large (typically several thousand), a normal approximation is sufficiently accurate. The transformation is $Z \equiv \sqrt{2\chi^2} - \sqrt{2\nu - 1}$. Under the assumption of normality (of the cf of x!), Z is distributed $N(0, 1)$
- ⋆ I plot this Z for blocks of length 1 (red), 2 (green), 3 (dark blue), 4 (light blue), 5 (violet) as a function of the number of digits computed. We are looking for large deviations (say, > 3) away from zero as a sign of abnormality

Pearson χ^2 results: $2^{1/3}$ and $3^{1/3}$





Autocorrelation of digits

- we would expect the the autocorrelation function (acf) of any analytic function of the digits that has a finite mean (for example, the log or the reciprocal) would decay like q^k at lag k, where $q \approx -0.3$ is Wirsing's constant
- this is investigated in the following graphs. I plot $\log_{10}$ of the absolute value of the acf as a function of lag. The green line has the Wirsing slope
- it is known that for almost all x, the mean of the digits does not exist. However, the mean of the log and mean of the reciprocal do exist and are approximately 0.98784905683381078769204 and 1.7454056624073468 respectively. All my examples give results consistent with these
- similarly for the mean of $(x_j)^{-k}, k = 2, 3, 4, \ldots, 10$

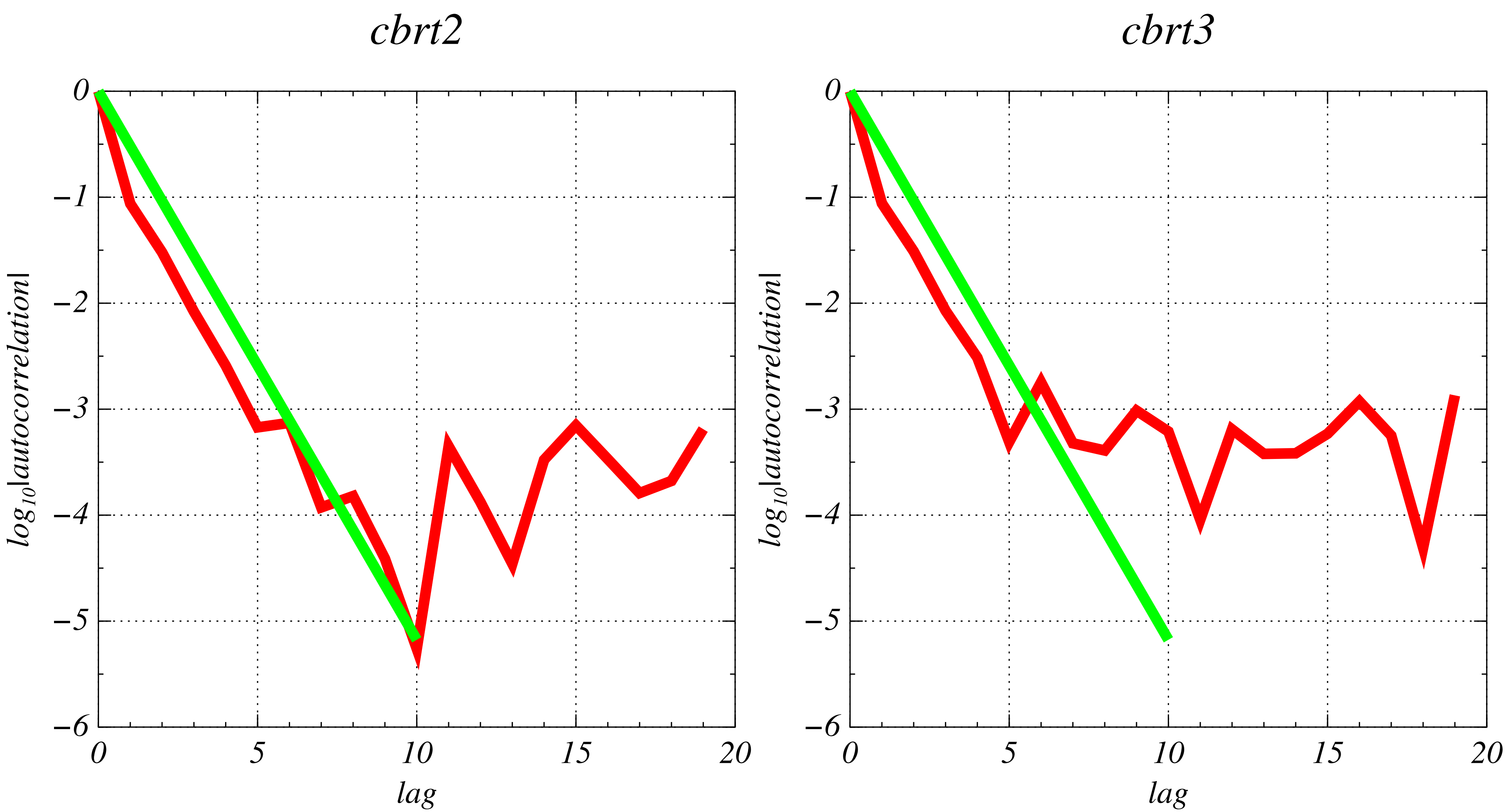
acf estimation difficulties

- for the AR(1) process $x(t+1) = \alpha x(t) + \epsilon$, $|\alpha| < 1$, the exact acf at lag k is $\rho(k) = \alpha^k$
- but the usual acf estimator r for a sample of size n has variance

$$\text{var}\,[r_n(k)] = \frac{1}{n}\left[\frac{(1+\alpha^2)(1+\alpha^{2k})}{1-\alpha^2} - 2k\alpha^{2k}\right]$$

- more generally, for a process whose acf decays for large k in the same exponential fashion, we have approximate variance $\text{var}\,[r_n(k)] = \frac{1}{n}\left[\frac{1+\alpha^2}{1-\alpha^2}\right]$ for large k
- I expect my process to conform to this behaviour, and if it does, putting in the numbers gives an estimator of $k = 6$ for the largest k for which the acf estimates are meaningful

Autocorrelation of logs of digits: $2^{1/3}$ and $3^{1/3}$



References

[1] R F Churchhouse & S T E Muir *Continued fractions, algebraic numbers and modular invariants*, J. Inst. Maths Applics **5**, 318-328 (1965)

[2] S Lang & H Trotter *Continued fractions for some algebraic numbers*, J. reine ang. Math. **255**, 112-134 (1972)

[3] R P Brent, A J van der Poorten & H J J te Riele *A comparative study of algorithms for computing continued fractions of algebraic numbers* in: Algorithmic Number Theory (H Cohen, ed.), LNCS, vol 1122, Springer-Verlag, 1996, 35-47

[4] D H Bailey, J M Borwein & R E Crandall *On the Khintchine constant*, Math. Comp. **66**, 417-431 (1997)

[5] D Hensley *Metric Diophantine approximation and probability*, New York J. Math. **4**, 249-257 (1998)

[6] D Hensley *The statistics of the continued fraction digit sum*, Pac. J. Math. **192**, 103-120 (2000)

[7] A M Rockett & P Szüsz *Continued Fractions*, World Scientific 1992, ISBN 981-02-1052-3

[8] M Iosifescu & C Kraaikamp *Metrical Theory of Continued Fractions*, Kluwer 2002, ISBN 1402008929 (Mathematics and Its Applications, vol 547)

[9] R Brent & P Zimmermann *Modern Computer Arithmetic*, version 0.1.1, November 2006, `www.loria.fr/~zimmerma/mca/mca-0.1.1.pdf`

[10] B Daireux & B Valleé *Dynamical analysis of the parameterized Lehmer-Euclid algorithm*, preprint

[11] E Cesaratto et al. *Analysis of fast versions of the Euclid algorithm*, preprint

[12] J P Soreson *Lehmer's algorithm for very large numbers*, preprint

[13] D J Bernstein *Fast multiplication and its applications*, preprint `http://cr.yp.to/arith.html`