Some Comments on C. S. Wallace's Random Number Generators

RICHARD P. BRENT

Oxford University Computing Laboratory,
Wolfson Building, Parks Road,
Oxford OX1 3QD, UK
Email: rpb213@rpbrent.co.uk

Dedicated to Chris Wallace on the occasion of his 70th birthday.

We outline some of Chris Wallace's contributions to pseudo-random number generation. In particular, we consider his recent idea for generating normally distributed variates without relying on a source of uniform random numbers, and compare it with more conventional methods for generating normal random numbers. Implementations of Wallace's idea can be very fast (approximately as fast as good uniform generators). We discuss the statistical quality of the output, and mention how certain pitfalls can be avoided.

Keywords: Gaussian distribution, maximum-entropy distributions, normal distribution, orthogonal transformations, random number generation, Wallace algorithm

Received 22 March 2003; revised 14 June 2003; accepted 12 May 2003

1. INTRODUCTION

In many simulation, graphics, simulated-annealing, cryptographic and Monte Carlo/Las Vegas programs, a substantial fraction of the time is used in generating pseudo-random numbers from the uniform, normal or other distributions, so methods of generating such numbers have received much attention.

This paper is dedicated to Chris Wallace, and our intention is to outline Wallace's substantial contribution to several aspects of random number generation, both in hardware and software.

In §2 we consider hardware random number generators (RNGs), and in §3 we mention (software) uniform RNGs. In §4 we consider "conventional" normal random number generators, and in §5 we consider Wallace's new "maximum entropy" idea for normal RNGs that do not depend in an essential way on a source of uniform RNGs. This idea is aesthetically appealing (why bother to generate uniform random numbers just in order to transform them by some time-consuming process into normal random numbers?) and has the potential to give extremely fast normal RNGs.

2. HARDWARE RNGS

In some cryptographic applications it is important for the numbers to be genuinely random, in the sense of being unpredictable, and not merely "pseudo-random", in the sense of passing various statistical tests. For example, this is the case when generating "one-time pads", or when constructing random primes whose products are to be made public for use with the Rivest, Shamir and Adleman "RSA" public-key cryptosystem [34], or when constructing exponents to be used in the Diffie-Hellman key exchange protocol or the El Gamal public-key cryptosystem [27].

Wallace in [37] described a simple hardware device that could provide a stream of unpredictable 32-bit numbers at a rate of 64 Mbit/sec, using a 4 MHz clock. The device was connected to the memory-mapped I/O bus of a multiprocessor computer, and appeared to a software process as a single 32-bit word of memory whose content was different (and unpredictable) every time it was read. Technology has advanced since 1990 so an implementation using similar ideas could now use a much faster clock. This could, for example, be used in the implementation of Rabin's "everlasting encryption" scheme [4, 5, 26, 32], which depends on the availability of a high-volume stream of random and unpredictable bits.

3. UNIFORM RNGS

In [36], Wallace considered several ways of obtaining uniform pseudo-random number generators with period close to 2^{64} on machines with 32-bit words. There are many ways to accomplish this [7, 13, 21, 23, 25], but most of them require a large amount of state information. This can be a problem if several independent streams of random numbers are required simultaneously. With Wallace's proposal, only two 32-bit words of state information are required.

4. CONVENTIONAL NORMAL RNGS

Most popular algorithms for generating normally distributed pseudo-random numbers are based on some variant of the rejection method, pioneered by Von Neumann [29]. More recent references are [1, 6, 12, 14, 20, 22]. Wallace [35] contributed some elegant and efficient generators of this class.

Rejection methods for normally distributed pseudo-random numbers require on average some number $U > 1$ of uniformly distributed numbers per normally distributed number. Thus, they can not be faster than the uniform random number generator, and are typically several times slower. Rejection methods for the normal distribution usually (though not always [6, 14]) involve the computation of functions such as log, sin, cos, which is slow compared to the time required to generate a uniform pseudo-random number. Leva [22, Table 1] compared several of the better methods and found that they are at least five times slower than a fast uniform generator on the same machine.

5. MAXIMUM ENTROPY NORMAL RNGS

Wallace [38] recently revolutionised normal random number generation by his discovery of a class of methods that do not depend in an essential way on uniform generators. Similar ideas can be used to generate pseudo-random numbers with some other distributions. In Wallace's paper [38] the uniform, Gaussian (normal) and exponential distributions are considered as maximum-entropy distributions subject to the following constraints:

Uniform: $0 \le x \le 1$
Gaussian: $\mathrm{E}(x^2) = 1$
Exponential: $\mathrm{E}(x) = 1, x \ge 0$.

The idea of a maximum-entropy distribution is most easily seen in the discrete case of N possibilities with probabilities $p_1, \ldots, p_N$. Subject to the constraints $p_j \ge 0$ and $\sum p_j = 1$, the uniform distribution $p_1 = \cdots = p_N = 1/N$ maximises the entropy $S = -\sum p_j \log p_j$. This can be proved using Lagrange multipliers. Similarly, the continuous distribution on $[0,1]$ that maximises $-\int_0^1 f(x) \log f(x) dx$ is the uniform distribution, and the continuous distribution on $(-\infty, +\infty)$ that maximises $-\int_{-\infty}^{+\infty} f(x) \log f(x) dx$ subject to $\int_{-\infty}^{+\infty} x^2 f(x) dx = 1$ is the Gaussian distribution. These statements can be proved using the calculus of variations. For the reader unfamiliar with Bayesian and maximum entropy methods, a good introduction is Jaynes [17]. An annotated bibliography is available at [18].

In the following we restrict our attention to the Gaussian case, since that is where Wallace's idea gives the most significant speedup over conventional methods. For example, Wallace's own implementation *FastNorm* is reported in [38, §5] to be only 13 percent slower than a generalised Fibonacci uniform random number generator on a RISC workstation.

Wallace proposed his method in a Technical Report in 1994, and a revision of this Report appeared two years later [38] along with an implementation *fastnorm*. Some changes in the implementation were made in 1998, resulting in an improved implementation *fastnorm2* [39]. There is an even better implementation *fastnorm3* [41], but this has not yet been published and we have had little time to test it, so we restrict our comments to *fastnorm2*.

Wallace [38] describes two implementations – one using integer arithmetic, and the other using floating-point arithmetic. On the workstation that he tested it on, the integer version was faster, but this might not be true on more recent machines with faster floating-point hardware.

Traditional normal RNGs are inefficient on vector processors. In 1993 the author compared various normal RNGs on vector processors and concluded that careful implementations of old methods such as the 1958 *Polar* method of Box, Muller and Marsaglia (see Knuth [21, Algorithm P]) and the 1959 Box-Muller method [21, 28] were faster than more recent methods [22] on vector processors produced by companies such as Cray, Fujitsu, and NEC: see [8, 30]. When Wallace's maximum-entropy idea appeared, it was clear that the landscape had changed, although the published implementation *fastnorm* was not intended to be efficient on vector processors. Thus, the author implemented an efficient vectorised version *rann4* [9, 10] of Wallace's maximum-entropy idea. *rann4* and a more recent implementation *rannw* [11] are more than three times faster than the methods previously thought to be the most efficient on vector processors.

5.1. Wallace's *fastnorm* algorithm

Many uniform random number generators generate one or more new uniform variables from a set of previously-generated uniform variables. Wallace's idea is to apply the same principle to normal random number generators. Given a set of normally distributed random variables, we can generate a new set of normally distributed random variables by applying a linear transformation that respects the "maximum entropy" constraint. This avoids the time-consuming conversion of uniform to normal variables that is required in conventional normal random number generators (see §4).

The key idea is: if x is an n-vector of independent, identically distributed $N(0,1)$ random variables $x_1, \ldots, x_n$, and Q is any $n \times n$ orthogonal matrix, then $y = Qx$ is another n-vector of independent, identically distributed $N(0,1)$ random variables. (Of course, the components y_i of y are dependent on the components x_j of x.) To prove the claim, observe that the component x_j has probability density $(2/\pi)^{-1/2} \exp(-x_j^2/2)$, so the vector x has probability density $(2/\pi)^{-n/2} \exp(-r^2/2)$, where $r = \|x\|_2$. This density depends only on r, the distance of x from the origin. However, since Q is orthogonal, $\|y\|_2 = \|Qx\|_2 = \|x\|_2 = r$.

Suppose that the n-vector x is a pool of n pseudo-random numbers that (we hope) are independent and normally distributed. We can generate a new pool $y = Qx$ by applying an orthogonal transformation Q. However, several problems arise.

5.2. Undesirable correlations

y_i is correlated with x_j. In fact, $y_i = q_{i,j} x_j + \cdots$, so $\mathrm{E}(y_i x_j) = \mathrm{E}(q_{i,j} x_j^2) = q_{i,j}$. This problem can be overcome by applying several different orthogonal tranformations $Q_1, Q_2, \ldots$ with a random choice of signs, so when averaged over all transformations $\mathrm{E}(q_{i,j}) \approx 0$.

5.3. Cost of transformations

It is too expensive to apply a general $n \times n$ orthogonal transformation Q to produce n new random numbers. This would involve of order n multiplications (and a similar number of additions) per random number generated. To overcome this problem, we can take Q to have a special form, e.g. in *rann4* we use a product of plane rotations of the form

$$R(\theta) = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix},$$

where θ varies, but is held constant within each inner loop. We do not need to compute trigonometric functions, since $\sin\theta = 2t/(1+t^2)$ and $\cos\theta = (1-t^2)/(1+t^2)$, where $t = \tan(\theta/2)$ varies; the angle θ is defined only for mathematical convenience and is never computed.

In his implementation *fastnorm*, Wallace preferred to use 4×4 orthogonal matrices A_1, A_2, A_3, A_4, where

$$A_1 = \frac{1}{2}\begin{bmatrix} 1 & 1 & -1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 \end{bmatrix},$$

and A_2, A_3, A_4 are similar. The advantage (on a machine with slow floating-point multiplication) is that multiplication of a 4-vector by A_1 requires only seven additions and one division by two (for details see [38, §2.1]).

The inner loop of the implementation is similar to the inner loop for the popular "generalised Fibonacci" uniform random number generators [3, 7, 15, 16, 21, 23, 31, 33]. Wallace's implementation of *fastnorm* on a RISC workstation is about as fast as a good uniform random number generator on the same workstation.

5.4. Mixing

As Wallace observes [38, §2.2], it is desirable that any value in the pool should eventually contribute to every value in the pools formed after several passes. In other words, the transformation from one pool to the next should be strongly "mixing". In our experience this is a tricky aspect of the implementation of generators based on Wallace's idea – several attempts which appeared plausible did not produce acceptable random numbers (after transformation to uniform variates they failed various statistical tests in Marsaglia's Diehard package [24]).

In *fastnorm*, Wallace ensures mixing by regarding the pool of 1024 values as a 256×4 matrix which is (implicitly) transposed at each pass; an additional ad hoc permutation is applied by stepping some row indices with an odd stride (mod 256). For details see [38, §2.2].

In *rann4* we effectively apply permutations of the form $\pi_1(j) = \alpha j + \gamma \bmod n$, $\pi_2(j) = \beta j + \delta \bmod n$, where $\gcd(\alpha, n) = \gcd(\beta, n) = 1$. Since n is a power of 2, any odd α and β can be chosen. For details see [9, §3].

Although the mixing transformations used in *fastnorm* and *rann4* appear satisfactory, they seem *ad hoc* and there is little helpful theory here – all we can do is apply empirical tests.

5.5. Chi-squared correction

Because Q is orthogonal, $\|Qx\|_2 = \|x\|_2$, so the sum of squares of numbers in a pool remains constant. This is unsatisfactory, because if $x_1, \ldots, x_n$ were independent samples from the normal $N(0,1)$ distribution, we would expect $\sum_{1 \le i \le n} x_i^2$ to have a chi-squared distribution χ^2_ν, where $\nu = n$ is the pool size.

To overcome this defect, Wallace suggests that one pseudo-random number from each pool should not be returned to the user, but should be used to approximate a random sample S from the χ^2_ν distribution. A scaling factor can be introduced to ensure that the sum of squares of the ν values in the pool (of which $\nu - 1$ are returned to the user) is S. If the routine is written to provide random numbers with mean μ and variance σ^2, then scaling by $S^{1/2}$ can be done at the same time as scaling by σ, so it is essentially free.

There are several approximations to the χ^2_ν distribution for large ν. For example, the one used in *rann4* is

$$2\chi^2_\nu \simeq \left(x + \sqrt{2\nu - 1}\right)^2,$$

where x is $N(0,1)$. It would not be much more expensive to use the (more accurate) Wilson-Hilferty approximation [42]

$$\chi^2_\nu \simeq \nu \left(\left(\frac{2}{9\nu}\right)^{1/2} x + \left(1 - \frac{2}{9\nu}\right) \right)^3.$$

Even better is

$$\chi^2_\nu \simeq A(x^2 - 1) + (2(\nu - A^2))^{1/2} x + \nu,$$

where

$$A = 2\sqrt{\nu}\, \sin\left(\frac{1}{3} \arcsin \frac{1}{\sqrt{\nu}}\right) = \frac{2}{3} + O\left(\frac{1}{\nu}\right)$$

satisfies the cubic equation $A^3 - 3\nu A + 2\nu = 0$. We can assume that ν is large ($\nu = 1024$ in *fastnorm*; ν depends on the size of the buffer provided by the user in *rann4/rannw*), so all of these approximations are sufficiently accurate. A slow but exact χ^2_ν algorithm, such as that of Ahrens and Dieter [2], is not required.

In the above approximations to χ^2_ν, the variable x was supposed to have a normal distribution. If only $n - 1$ values are returned to the user from a pool of n values, the remaining (scaled) value x can be used to approximate χ^2_ν for the next pool. This is a point where the implementations of *fastnorm* and *fastnorm2* differ. In *fastnorm*, x is taken from the current pool, but in *fastnorm2* it is taken from the previous pool. The choice used in *fastnorm* is undesirable because a large value of x, and hence a large scaling factor from the χ^2_ν approximation, is correlated with a small sum of squares of the remaining values in the pool (since the sum of squares including x is invariant).

5.6. More subtle correlations

In §5.2 we saw how, by using several orthogonal transformations, we could ensure that $\mathrm{E}(y_i x_j) \approx 0$. However, more subtle correlations persist. Consider the simplified model

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = R(\theta) \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} ,$$

where $R(\theta)$ is a plane rotation as above, and θ is distributed uniformly in $[0, 2\pi)$. We write $c = \cos\theta$, $s = \sin\theta$. Thus $y_1 = cx_1 + sx_2$, $y_2 = -sx_1 + cx_2$. Suppose that x_1 and x_2 are independent and normally distributed, with zero mean and unit variance. Then

$$E(x_1^2 y_1^2) = E(c^2)E(x_1^4) + E(s^2)E(x_1^2 x_2^2) = 2 \neq E(x_1^2)E(y_1^2) .$$

In *fastnorm/fastnorm2* and *rann4/rannw*, similar effects occur, although the undesirable correlations are small and they occur between well-separated outputs (the separations are of the order of the pool size) because of the permutations used to provide mixing (*cf* §5.4).

5.7. Other finite pool size effects

Chris Wallace [40] has observed a phenomenon that, like the one discussed in §5.6, becomes less significant as the pool size increases, but never disappears entirely for any finite pool size n.

Consider a rare event such as the occurrence of a large normal variate x that is expected to occur say once in every $10n$ samples, i.e. once in every 10 pools. The "energy" x^2 is distributed over only a small number (four) of variables in the next pool. Thus we can expect one or more of these variables to be unusually large. Although the distribution of values considered over many pools is correct, it is more likely that rare events will occur in adjacent pools.

It is possible to devise statistical tests that detect this behaviour and/or the correlations described in §5.6. However, we have not obtained any statistically significant results with a sample size of less than $10^4 n$.

Clearly, one way to reduce (though not eliminate) the significance of such effects is to increase the pool size (easy for *rann4/rannw*). Another way is to discard some of the numbers produced by the random number generator – e.g. we could use every third value, or the values in every third pool. This has an obvious effect on the speed of the generator, but because the underlying algorithm is so fast we can afford to do it and still have a random number generator that is faster than more conventional generators (*cf* §4).

5.8. Use of uniform RNGs

Although normal generators based on the maximum entropy idea do not use uniform random numbers in any *essential* way, it is convenient to use a uniform RNG for purposes such as initialisation, selection of orthogonal transformations, etc. The advantage of the maximum entropy methods is that the number U of uniform distributed numbers required per normally distributed number is very small (of the order of $1/n$ for pool size n), whereas for rejection methods $U > 1$.

If we choose a uniform random number generator with known long period, and use it at least once for each pool of normal random numbers (e.g. to select from a set of possible orthogonal transformations), then it is easy to guarantee that the period of the normal random number generator is at least as great as that of the uniform random number generator. Thus, although any use of a uniform random number generator might be considered contrary to the spirit of the maximum entropy method, it does have the practical benefit of guaranteeing a long period. If (as is certainly possible) we avoided using a uniform generator except perhaps for initialisation, then we could not *guarantee* a long period, although a short period would be extremely unlikely, since it would require an implausible coincidence in the initialisation.

5.9. Summary

Although care needs to be taken in the implementation of normal random number generators like *fastnorm*, and the end-user should be aware of the small but unavoidable defects discussed in §§5.6-5.7, these generators have such a performance advantage over more conventional generators that they can not be ignored in applications where the speed of generation of pseudo-random numbers is critical.

ACKNOWLEDGEMENTS

Comments by David Dowe, Chris Wallace, and two anonymous referees have helped to improve this paper and are gratefully acknowledged.

REFERENCES

[1] J. H. Ahrens and U. Dieter (1972) Computer methods for sampling from the exponential and normal distributions. *Comm. ACM*, **15**, 873–882.

[2] J. H. Ahrens and U. Dieter (1982) Generating Gamma variates by a modified rejection technique. *Comm. ACM*, **25**, 47–54.

[3] S. L. Anderson (1990) Random number generators on vector supercomputers and other advanced architectures. *SIAM Review*, **32**, 221–251.

[4] Y. Aumann, Y. Z. Ding and M. O. Rabin (2002) Everlasting security in the bounded storage model. *IEEE Transactions on Information Theory*, **48**, 1668–1680.

[5] Y. Aumann and M. O. Rabin (1999) Information theoretically secure communication in the limited storage space model. *Advances in Cryptology – CRYPTO '99, Lecture Notes in Computer Science*, **1666**, Springer-Verlag, New York, 65–79.

[6] R. P. Brent (1974) Algorithm 488: A Gaussian pseudo-random number generator (G5). *Comm. ACM*, **17**, 704–706.

[7] R. P. Brent (1992) Uniform random number generators for supercomputers. *Proc. Fifth Australian Supercomputer Conference*, Melbourne, 95–104.

[8] R. P. Brent (1993) *Fast Normal Random Number Generators for Vector Processors*. Technical Report TR-CS-93-04, Computer Sciences Laboratory, Australian National University, 7 pp. `http://www.comlab.ox.ac.uk/oucl/work/richard.brent/pub/pub141.html`.

[9] R. P. Brent (1997) *A fast vectorised implementation of Wallace's normal random number generator*. Technical Report TR-CS-97-07, Computer Sciences Laboratory, Australian National University, 9 pp.

http://www.comlab.ox.ac.uk/oucl/work/richard.brent/pub/pub170.html.

[10] R. P. Brent (1998) Random number generation and simulation on vector and parallel computers. *Proc. Fourth International Euro-Par Conference* (Southampton, UK), D. Pritchard and J. Reeve (editors), *Lecture Notes in Computer Science*, **1470**, Springer-Verlag, Berlin, 1–20.

[11] R. P. Brent (2002) *Some Uniform and Normal Random Number Generators*. Software available from http://www.comlab.ox.ac.uk/oucl/work/richard.brent/random.html.

[12] L. Devroye (1986) *Non-Uniform Random Variate Generation*. Springer-Verlag, New York.

[13] A. M. Ferrenberg, D. P. Landau and Y. J. Wong (1992) Monte Carlo simulations: hidden errors from "good" random number generators. *Phys. Review Letters*, **69**, 3382–3384.

[14] G. E. Forsythe (1972) Von Neumann's comparison method for random sampling from the normal and other distributions. *Math. Comp.*, **26**, 817–826.

[15] B. F. Green, J. E. K. Smith and L. Klem (1959) Empirical tests of an additive random number generator. *J. ACM*, **6**, 527–537.

[16] F. James (1990) A review of pseudorandom number generators. *Computer Physics Communications*, **60**, 329–344.

[17] E. T. Jaynes (1986) Bayesian methods: general background. In [19], 1–25.

[18] E. T. Jaynes (1994) References: a partially annotated bibliography. http://omega.math.albany.edu:8008/ETJ-PS/crefsq.ps.

[19] J. H. Justice (ed.) (1986) *Maximum Entropy and Bayesian methods in Applied Statistics*. Cambridge University Press, Cambridge.

[20] A. J. Kinderman and J. F. Monahan (1977) Computer generation of random variables using the ratio of uniform deviates. *ACM Transactions on Mathematical Software*, **3**, 257–260.

[21] D. E. Knuth (1998) *The Art of Computer Programming, Volume 2: Seminumerical Algorithms* (third edition). Addison-Wesley, Menlo Park.

[22] J. L. Leva (1992) A fast normal random number generator. *ACM Transactions on Mathematical Software*, **18**, 449–453.

[23] G. Marsaglia (1985) A current view of random number generators. *Computer Science and Statistics: The Interface* (edited by L. Billard), Elsevier Science Publishers B. V. (North-Holland), 3–10.

[24] G. Marsaglia (1995) *Diehard*. Available from http://stat.fsu.edu/~geo/.

[25] G. Marsaglia and A. Zaman (1991) A new class of random number generators. *Annals of Applied Probability*, **1**, 462–480.

[26] U. Maurer (1992) Conditionally perfect secrecy and a provably-secure randomized cipher. *J. of Cryptology*, **5**, 53–66.

[27] A. J. Menezes, P. C. van Oorschot and S. A. Vanstone (1997) *Handbook of Applied Cryptography*. CRC Press, New York, 1997. http://cacr.math.uwaterloo.ca/hac/.

[28] M. E. Muller (1959) A comparison of methods for generating normal variates on digital computers. *J. ACM*, **6**, 376–383.

[29] J. Von Neumann (1951) Various techniques used in connection with random digits. *The Monte Carlo Method*, National Bureau of Standards (USA) Applied Mathematics Series, **12**, 36. Also *Collected Works*, **5**, Pergamon Press, New York, 1963, 768–770.

[30] W. P. Petersen (1988) Some vectorized random number generators for uniform, normal, and Poisson distributions for CRAY X-MP. *J. Supercomputing*, **1**, 327–335.

[31] W. P. Petersen (1994) Lagged Fibonacci series random number generators for the NEC SX-3. *Internat. J. High Speed Computing*, **6**, 387–398.

[32] M. O. Rabin (1983) Transaction protection by beacons, *J. Computer and System Sciences*, **27**, 256–267.

[33] J. F. Reiser (1977) *Analysis of Additive Random Number Generators*. Ph. D. thesis and Technical Report STAN-CS-77-601, Stanford University, Stanford, California.

[34] R. L. Rivest, A. Shamir and L. Adelman (1978) A method for obtaining digital dignatures and public-key cryptosystems. *Comm. ACM*, **21**, 120–126.

[35] C. S. Wallace (1976) Transformed rejection generators for gamma and normal pseudorandom variables. *Australian Computer Journal*, **8**, 103–105.

[36] C. S. Wallace (1989) *A Long-Period Pseudo-Random Generator*. Technical Report 89/123, School of Computer Science and Software Engineering, Monash University, 7 pp.

[37] C. S. Wallace (1990) Physically random generator. *Computer Systems Science and Engineering*, **5**, 82–88.

[38] C. S. Wallace (1996) Fast pseudo-random generators for normal and exponential variates. *ACM Trans. on Mathematical Software*, **22**, 119–127. Also Report TR#94/197, Department of Computer Science, Monash University, February 1994.

[39] C. S. Wallace (1998) *MDMC Software - Random Number Generators*. Available from http://www.csse.monash.edu.au/research/mdmc/software/random/ (version dated 25 June 1998).

[40] C. S. Wallace (2003a) *FastNorm*. Personal communication by email, 26 February 2003.

[41] C. S. Wallace (2003b) *FastNorm3*. Personal communication by email, 20 May 2003.

[42] E. B. Wilson and M. M. Hilferty (1931) The distribution of chi-square. *Proc. Nat. Acad. Sci. (USA)*, **17**, 684–688.