SIAM REVIEW
Vol. 49, No. 4, pp. 673–692

Searching for Rare Growth Factors Using Multicanonical Monte Carlo Methods*

Tobin A. Driscoll†
Kara L. Maki†

Abstract. The growth factor of a matrix quantifies the amount of potential error growth possible when a linear system is solved using Gaussian elimination with row pivoting. While it is an easy matter [N. J. Higham and D. J. Higham, *SIAM J. Matrix Anal. Appl.*, 10 (1989), pp. 155–164] to construct examples of $n \times n$ matrices having any growth factor up to the maximum of 2^{n-1}, the weight of experience and analysis [N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia, 1996], [L. N. Trefethen and R. S. Schreiber, *SIAM J. Matrix Anal. Appl.*, 11 (1990), pp. 335–360], [L. N. Trefethen and I. D. Bau, *Numerical Linear Algebra*, SIAM, Philadelphia, 1997] suggest that matrices with exponentially large growth factors are exceedingly rare. Here we show how to conduct numerical experiments on random matrices using a multicanonical Monte Carlo method to explore the tails of growth factor probability distributions. Our results suggest, for example, that the occurrence of an 8×8 matrix with a growth factor of 40 is on the order of a once-in-the-age-of-the-universe event.

Key words. Gaussian elimination, growth factors, Markov chain Monte Carlo, multicanonical Monte Carlo

AMS subject classifications. 15A52, 65C05, 65C40, 65F05

DOI. 10.1137/050637662

1. Introduction.

It has long been known that Gaussian elimination, even with the common expedient of row pivoting, can be disastrously inaccurate for solving certain linear systems. For example, using the MATLAB operator \ on a particular system, we find

```
>> A = toeplitz([1 -ones(1,59)],[1 zeros(1,59)]); A(:,60)=1;
>> randn('state',3383)   % for reproducibility
>> x = randn(60,1);  b = A*x;
>> x1 = A\b;
>> norm(x-x1)/norm(x)
ans =
    0.3402
```

Considering that the data are accurate to 16 decimal digits, the method fails utterly. Usually we are able to explain such a failure in terms of the condition number $\kappa(A)$,

*Received by the editors August 5, 2005; accepted for publication (in revised form) March 2, 2007; published electronically November 1, 2007. The computational work was made possible by NSF SCREMS grant DMS-0322583.
http://www.siam.org/journals/sirev/49-4/63766.html

†Department of Mathematical Sciences, University of Delaware, Ewing Hall, Newark, DE 19716 (driscoll@math.udel.edu, maki@math.udel.edu).

which measures the sensitivity of x in the problem $Ax = b$ to perturbations in A and b. In this case, however, poor conditioning is not responsible:

```
>> cond(A)
ans =
   26.8035
```

Only one decimal digit of lost accuracy can thus be explained by the nature of the problem. Ordinarily we would then conclude that the algorithm is responsible and therefore unstable.

Why does MATLAB—and other software and textbooks—present an apparently unstable algorithm as the default choice for solving linear systems? Indeed, even though "Gaussian" elimination is ancient, appearing recognizably for a 3×3 system in the Chinese text *Nine Chapters on the Mathematical Art*, thought to be over 2,000 years old [3], experts in the early days of digital computing were rather pessimistic about its computational prospects. (See the summary and references in [22].) However, the algorithm was well known, the fastest available, and widely used without incident, so that confidence in it quickly grew. Today, all experts agree that the example above—deliberately chosen to illustrate the dangers of Gaussian elimination—is nowhere close to typical or average. In fact, one must consider it pathological; only in the early 1990s, for instance, did a few applications with large growth factors begin to be noted in the literature [10, 26]. Efforts have been made to explain this phenomenon, with some success (see [15, 22] and especially, as noted below, [21, Chap. 22]), yet the matter is far from completely understood.

In this article we try to add some computational evidence to reinforce the prevailing wisdom. We study the *growth factors* of random matrices. The growth factor (defined in the next section) quantifies the amount of mayhem possible in Gaussian elimination. While it can be as large as 2^{n-1} for an $n \times n$ matrix, as in the example above, the occurrence of a growth factor even as large as n is quite rare—so rare, in fact, that it is difficult to measure using naive experimentation with random matrices. The effect is analogous to waiting for the million proverbial monkeys to type out the works of Shakespeare spontaneously.[1]

Capturing rare events can be made practical, however, if we are willing to tinker with the simulation process. Suppose we give our monkeys an editor who is already familiar with Shakespeare. The editor will not simply accept everything the monkeys type, preferring instead to keep them on track. Thus, if the monkeys have so far managed "To be or no," the editor might be very likely to reject anything other than a "t" next. This greatly decreases the amount of time we expect to wait for the Hamlet soliloquy. Of course, to keep the statistics fair, we have to track the rejections made by the editor. Roughly speaking, this is the approach we take for growth factors.

2. Growth Factors. The growth factor was introduced to aid in rounding error analysis of the solution to $Ax = b$, as found by Gaussian elimination. The most common form of this algorithm is found in any numerical analysis textbook. First, factor $PA = LU$ using row-pivoted elimination; here, P is a permutation matrix, L is unit lower triangular (lower triangular with ones on the diagonal), and U is upper triangular. Define $c = Pb$. Solve $Lz = c$ for z with forward substitution and then solve $Ux = z$ for x with backward substitution.

[1] Mathematically, one monkey is as good as a million in this effort.

With this in mind, one way to define the *growth factor* of A is

$$\rho(A) = \frac{\max_{ij} |u_{ij}|}{\max_{ij} |a_{ij}|}. \tag{2.1}$$

(This is not to be confused with the spectral radius of A, also frequently denoted $\rho(A)$.) Another definition of the growth factor [12, 14] involves all the intermediate matrices of the elimination process, not just the final one. Our version is a lower bound of this alternative and is more convenient to compute; it has appeared in [8, 21] and is returned by the LAPACK routines xGESVX for linear system solutions [1]. Both variations share the upper bound $\rho(A) \leq 2^{n-1}$. Because the matrices in the factorization depend on discrete decisions about row swaps made during the elimination process, ρ is not a continuous function.

If $\hat{x}$ is the solution to $Ax = b$ computed as described above on a floating-point computer, then one of a family of similar results is [12]

$$(A + \delta A)\hat{x} = b \text{ for some } \delta A \text{ with } ||\delta A||_\infty \leq 3n^3 u \rho(A) ||A||_\infty, \tag{2.2}$$

where u is unit roundoff (half the distance between 1 and the next larger floating-point number), and $||B||_\infty$ is the induced operator max-norm of matrix B, or the maximum row sum in $|B|$. The derivation of the bound accounts for rounding errors introduced during both elimination and triangular system solving.

Let us reconsider the MATLAB example of the introduction in light of the error result (2.2). The first line in the example defines A as a matrix of the form

$$A = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 1 \\ -1 & 1 & 0 & \cdots & 0 & 1 \\ -1 & -1 & 1 & \cdots & 0 & 1 \\ \vdots & \ddots & \ddots & \ddots & & \vdots \\ -1 & \cdots & \ddots & \ddots & 1 & 1 \\ -1 & -1 & \cdots & -1 & -1 & 1 \end{bmatrix}. \tag{2.3}$$

Each row elimination doubles an element in the last column, so this matrix achieves the upper bound $\rho(A) = 2^{n-1}$, as pointed out by Wilkinson [25]. With $n = 60$, the bound (2.2) allows the perturbation δA to have a norm larger than that of A, since $u = 2^{-52}$ in MATLAB.

This example is actually special in that the L and U factors, found to be

$$L = \begin{bmatrix} 1 & & & \\ -1 & 1 & & \\ \vdots & \ddots & \ddots & \\ -1 & \cdots & -1 & 1 \end{bmatrix}, \qquad U = \begin{bmatrix} 1 & & & 1 \\ & 1 & & 2 \\ & & \ddots & \vdots \\ & & & 2^{n-1} \end{bmatrix}, \tag{2.4}$$

are computed exactly, as the Gaussian elimination steps all involve operations on integers only. Therefore, all of the error in this example is introduced during the triangular solves, and we need only appeal to the usual perturbation analysis for linear systems. In fact, the experiment of Figure 1 suggests that the condition number of U alone captures the essence of the numerical difficulty in solving $Ax = b$. Hence the flaw in row-pivoted Gaussian elimination is not a difficulty in finding the correct L and U

```
for n=1:60
  A=toeplitz([1 -ones(1,n-1)],[1 zeros(1,n-1)]); A(:,n)=1;
  x=randn(n,1); b=A*x;
  err(n)=norm(A\b - x);
  [L,U]=lu(A);  condU(n)=cond(U);
  rho(n)=max(max(abs(U)))/max(max(abs(A)));
end
semilogy(1:60,err,'b', 1:60,eps*condU,'k', 1:60,eps*rho,'r')
```

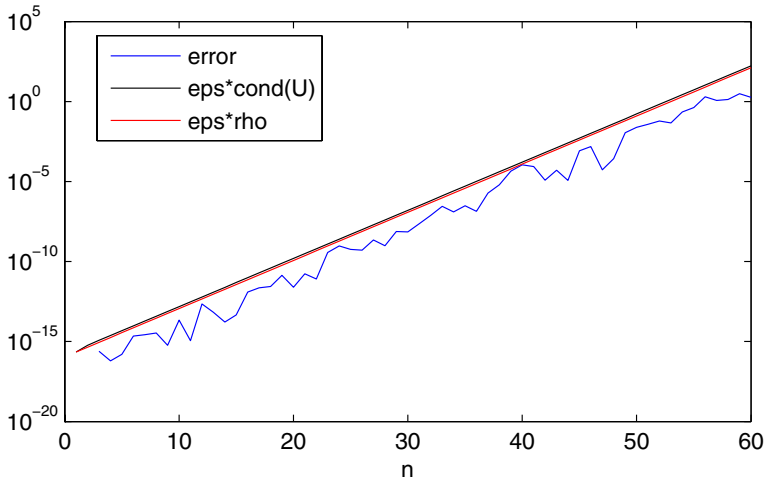

Fig. 1 *Code and error plot for the solution to Ax=b, where A is of the form* (2.3)*, along with* $\rho(A)$ *and the condition number of the computed matrix* U*. In MATLAB,* `eps` *is the relative error in the machine representation of real numbers, which represents a mathematical perturbation.*

factors in the presence of rounding errors, but in the more fundamental idea of trying to use these factors to solve a linear system in the presence of roundoff.

While this flaw runs deep, it is exceedingly narrow. We reiterate that it has long been observed that matrices with even modestly sized growth factors have proven to be very rare. For instance, based on experiments partially reproduced in Figure 3, Trefethen and Bau in [21, Chap. 22] all but conjecture that for any $\alpha > \frac{1}{2}$ and $M > 0$, the probability of encountering $\rho > n^\alpha$ among random matrices is less than n^{-M} for sufficiently large n.

In the sections to follow, we introduce and illustrate a method motivated by statistical physics for exploring just how rare large growth factors are among random matrices.

3. Monte Carlo Simulations. We next need to define what we mean by random matrices. Let n be a natural number. We will represent an $n \times n$ matrix somewhat unusually by the symbol x, where x is a vector of $N = n^2$ real numbers (though nothing in our methods precludes the use of complex matrices). The probability of choosing a particular x is controlled by a *probability density function* (pdf) $\pi(x)$. The model we shall use is

$$\pi(x) = \prod_{k=1}^{N} \frac{1}{\sqrt{2\pi}} e^{-x_k^2/2}, \tag{3.1}$$

which is simply a multidimensional normal distribution with unit variance in each entry of x. A fundamental property of any pdf is that

$$\int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \pi(x)\, dx_1 \cdots dx_N = \int_{\Omega} \pi(x)\, dx = 1,$$

where Ω is the *state space* $\mathbf{R}^N$ of all real $n \times n$ matrices.

We do not attempt to argue that matrices drawn with a probability according to (3.1) are somehow natural or even typical—whatever those terms might mean. The pdf simply serves as one way to systematically specify how matrices are to be chosen. Another simple pdf would be to choose each element of x uniformly from the interval $[-1, 1]$. One could also easily adapt our methods to select among only symmetric matrices or those with a specific sparsity pattern, for example.

For each randomly selected x we can compute its growth factor $\rho(x)$, where $0 < \rho(x) \leq 2^{n-1}$. In the language of probability theory, both x and ρ are *random variables*. We now set ourselves the task of finding the pdf for the random variable ρ. We first discretize this pdf by dividing the range of achievable growth factors into bins, $0 = \rho_0 < \rho_1 < \cdots < \rho_B = 2^{n-1}$, and letting P_b be the probability that ρ lands in bin b when x is drawn according to $\pi(x)$. The probability P_b can also be expressed as the expected value (i.e., average value) of a function indicating when ρ lands in bin b as x is drawn according to $\pi(x)$. Symbolically,

$$P_b = \int_{\Omega_b} \pi(x)\, dx = \int_{\Omega} \chi_b(x)\pi(x)\, dx = E\left[\chi_b\right], \qquad b = 1, \ldots, B, \tag{3.2}$$

where $\Omega_b = \{x \in \Omega : \rho_{b-1} < \rho(x) \leq \rho_b\}$ consists of all x landing in bin b, and χ_b is the indicator function

$$\chi_b(x) = \begin{cases} 1 & \text{if } x \in \Omega_b, \\ 0, & \text{otherwise.} \end{cases} \tag{3.3}$$

A simple consequence of our definitions is that $\sum_{b=1}^{B} P_b = 1$.

Expressing P_b as an average suggests a fairly obvious experimental approach to approximating its value. If we draw M samples $x^{(1)}, \ldots, x^{(M)}$ according to the density function π, then it seems that we could use

$$p_b = \frac{1}{M} \sum_{m=1}^{M} \chi_b\big(x^{(m)}\big) \tag{3.4}$$

as an estimate of P_b. In practice, drawing the samples from the distribution π is readily achievable, as virtually all computing languages and environments include high-quality generators of normal random (technically, pseudorandom) numbers. In MATLAB one uses `randn`.

From the perspective of probability theory, the sampling process produces a sequence of M independent random variables $x^{(1)}, \ldots, x^{(M)}$, each with pdf π. Using the linearity of the expectation operator, we find that the expected value of p_b is indeed

just what we seek:

$$\begin{aligned} E[p_b] &= \frac{1}{M}\sum_{m=1}^{M} E\left[\chi_b\big(x^{(m)}\big)\right] \\ &= \frac{1}{M}\sum_{m=1}^{M} \int_\Omega \chi_b\big(x^{(m)}\big)\pi\big(x^{(m)}\big)\,dx \\ &= \frac{1}{M}\sum_{m=1}^{M} P_b = P_b. \end{aligned}$$

The technique described in the previous paragraph is very well known as the *Monte Carlo* method—so named after the famous European gambling resort, in honor of an uncle of the mathematician Stanislaw Ulam [23]. We will illustrate the method first on a simpler problem of classical interest, the *random walk*. In this problem, a monkey (on a coffee break from typing) starts at the origin and flips a fair coin. If it is heads, the monkey walks to the right; if tails, the monkey stays put. Where can we expect the monkey to be after N flips?

To use our terminology, let x again be a vector of N components, where each component is selected from the binary set $\{0, 1\}$, and define $\rho(x)$ as the sum of the elements of x. The state space Ω of all possible x is finite, with 2^N elements, and the integrals over Ω above can be written as sums. The appropriate probability distribution is now uniform, meaning that each vector of coin flips is equally likely: $\pi(x) = 2^{-N}$. Furthermore, the only possible outcomes for ρ are integers from 0 to N, so we can let each bin contain just one integer and estimate the actual pdf of ρ, not just a discretization of it.

Figure 2 shows the results of a MATLAB Monte Carlo experiment for the random walk. The behavior of the walk is well understood, and ρ is known to obey the binomial distribution

$$P_b = 2^{-N}\binom{N}{b}, \qquad b = 0, \ldots, N.$$

This is shown for comparison to the experimental result. The estimates p_b are shown with vertical line segments to indicate confidence intervals of three standard deviations (more on this below).

Monte Carlo experiments for growth factors have been performed by Trefethen and Bau and appeared in their textbook [21]. One of their figures is reproduced here as Figure 3. From Trefethen and Bau's results we can see that the probability of encountering a matrix with growth factor ρ decreases rapidly with ρ. Indeed, while a growth factor of 2^{n-1} can be observed, Trefethen and Bau reported only two matrices out of a total of one million that have a growth factor as large as $\sqrt{n}$, and the largest overall was $\rho \approx 11.99$.

Figures 2 and 3 clearly show a major shortcoming of the Monte Carlo method: it becomes less informative as the measured probability drops. For relatively common events, i.e., larger values of the pdf, the estimates are good and our confidence in them is high, but as one moves out into the tails the estimates deteriorate and then fail altogether. The reason is obvious—by definition, rare cases are rarely encountered by chance, and thus it is hard to compile meaningful statistics about them in experiments. Another point of view is that values in the tail of the pdf are found accurately relative only to the overall maximum of the pdf, and not to the values themselves.

```
N = 100;                      % number of walking steps
M = 1e6;                      % number of MC trials
hit = zeros(N+1,1);           % hits in each bin
fact = [1 cumprod(1:N)]';     % vector of factorials
exactpdf = fact(N+1) ./ (2^N*fact.*fact(N+1:-1:1));
semilogy(0:N,exactpdf,'k-'), hold on

for m = 1:M                   % Monte Carlo trials
  r = rand(1,N);              % uniform in [0,1]
  x = double(r>0.5);          % results of the coin flips
  rho = sum(x);
  hit(rho+1) = hit(rho+1)+1;
end

p = hit/M;                    % pdf for rho
semilogy(0:N,p,'*')
sigma = sqrt( (p-p.^2)/(M-1) );  % std deviation
upper = p+3*sigma;  lower=max(realmin,p-3*sigma);
plot([0:N;0:N],[lower'; upper'],'b')
```

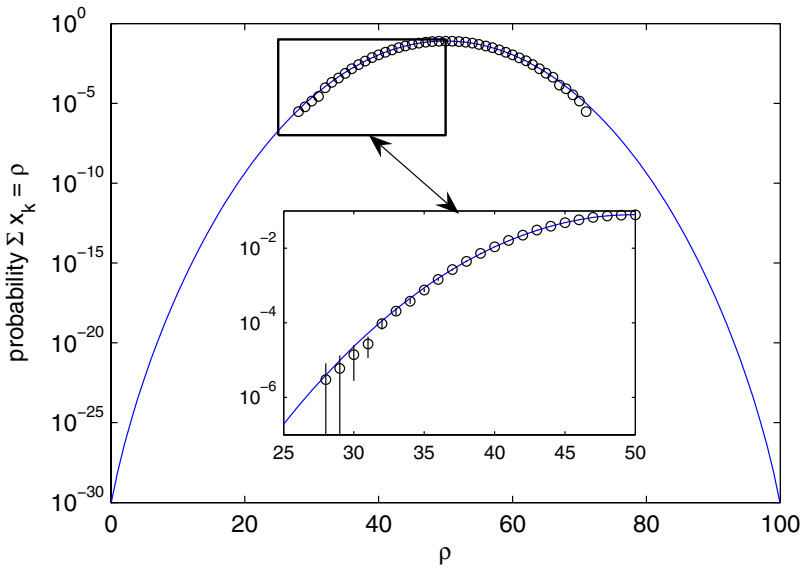

Fig. 2 *MATLAB Monte Carlo experiment for the random walk. The circles show the results; the curve is the analytically known answer. After* 10^6 *realizations, only a small part of the pdf is captured, and for probabilities near* 10^{-6} *the relative confidence (as indicated by the vertical line segments) in the Monte Carlo estimates is poor.*

The sample size M is the main parameter available in the Monte Carlo method and is of major importance to our approximation p_b. We expect that $|p_b - P_b|$ approaches zero as M increases. Indeed, the strong form of the law of large numbers says that the probability of this *not* happening is zero. However, the rate of convergence in the limiting process turns out to be critical. To study this rate, we consider the *variance* of p_b, defined as

$$\text{Var}[p_b] = E\left[(p_b - E[p_b])^2\right] = E\left[p_b^2\right] - (E[p_b])^2. \tag{3.5}$$

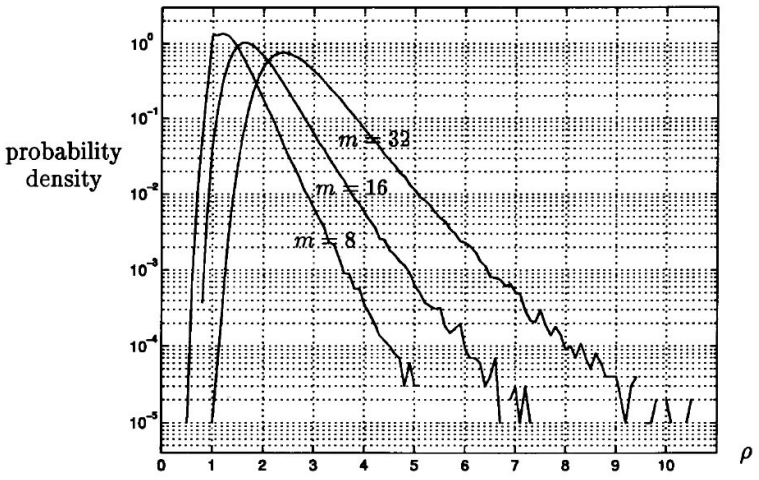

Fig. 3 *Pdfs for growth factors of random matrices of dimensions* 8, 16, *and* 32 [21]. *Copyright* ©1997 *Society for Industrial and Applied Mathematics. Reprinted with permission.*

The second equality follows easily from the linearity of expectation. Variance is a measure of the spread in a random variable. In order to have a measure with the same units as the variable itself, we can take the square root of variance to get the *standard deviation*.

The central limit theorem says that the sum of a large number of independent random variables has a distribution that is (in a particular sense) approximately normal. We know that the expected value of this distribution for p_b is the sought-after P_b. Thus for large M, p_b is very likely to be within a few standard deviations on either side of P_b. Using (3.4), some more elementary theory about variance, and the fact that $\chi_b^2(x) = \chi_b(x)$, we compute

$$\begin{aligned}
\operatorname{Var}[p_b] &= \frac{1}{M^2} \sum_{m=1}^{M} \operatorname{Var}\left[\chi_b\big(x^{(m)}\big)\right] \\
&= \frac{1}{M^2} \sum_{m=1}^{M} E\left[\chi_b^2\big(x^{(m)}\big)\right] - E\left[\chi_b\big(x^{(m)}\big)\right]^2 \\
&= \frac{1}{M^2} \sum_{m=1}^{M} (P_b - P_b^2) = \frac{P_b - P_b^2}{M}.
\end{aligned}$$

This formula appears in our computation of the confidence bars in Figure 2.[2]

Finally, we consider how the standard deviation compares to the statistic P_b itself. The ratio of standard deviation to expectation is known as the *coefficient of variation*, and its role is similar to that of relative accuracy in numerical analysis. In

[2]Dividing by $M - 1$ rather than M is a well-known consequence when the estimates p_b are used in place of the exact values P_b; the actual difference here is negligible.

our notation it equals

$$\mathrm{CV}(p_b) = \frac{1}{P_b}\sqrt{\frac{P_b(1-P_b)}{M}} = \sqrt{\frac{(1-P_b)}{P_b M}}. \tag{3.6}$$

We now see clearly that the uncertainty or variability associated with the result of the Monte Carlo method decreases like $M^{-1/2}$. That is, to reduce $\mathrm{CV}(p_b)$ by half, we need to take four times as many samples.

A second observation from (3.6), and the one most relevant to the remainder of this paper, is that if a bin is visited rarely, so that $P_b \ll 1$, then M must be at least on the order of P_b^{-1} for the method to succeed. This is a quantitative statement about the difficulty of simulating the tail of the distribution. Hence it is no coincidence in Figure 2 that when using a million samples, the estimates lose accuracy and vanish just as one approaches a probability of 10^{-6}. To achieve, say, just $\mathrm{CV}(p_b) = 10\%$ when $P_b = 10^{-9}$, we would need $M \approx 10^{11}$ trials, which might test our patience even on a fast computer.

4. Multicanonical Simulations. The difficulty of getting into the tail of the pdf is that bins in the tail are visited rarely. How do we rectify this situation? Clearly we need to load the dice, in the form of choosing samples not from the distribution π but from an alternative $\widehat{\pi}$. In light of our experience, our guiding principle will be to try to ensure that all of the histogram bins get visited equally. To this end, we define

$$\widehat{\pi}(x) = \frac{\pi(x)}{w(x)}, \qquad w(x) = \sum_{b=1}^{B} \frac{\chi_b(x)}{w_b}, \tag{4.1}$$

where $w_1, \ldots, w_B$ are positive constants that change the weights of the different bins. Choosing them defines the new sampling strategy. They are constrained by the fact that $\widehat{\pi}$ must satisfy

$$1 = \int_\Omega \widehat{\pi}(x)\,dx = \sum_{b=1}^{B} \int_{\Omega_b} w_b \pi(x)\,dx = \sum_{b=1}^{B} P_b w_b. \tag{4.2}$$

If we draw samples according to $\widehat{\pi}$ rather than π, the expectation in each bin becomes

$$\widehat{E}\,[\chi_b] = \int_\Omega \chi_b(x)\widehat{\pi}(x)\,dx = \int_{\Omega_b} w_b \pi(x)\,dx = P_b w_b. \tag{4.3}$$

We have used a hat on the expectation functional to emphasize that it is with respect to a new sampling pdf. Equal sampling therefore occurs if $P_b w_b$ is independent of the bin number b. Using (4.2) we find that $P_b w_b = 1/B$ in the ideally equidistributed case.

With the new sampling strategy, we can define Monte Carlo estimates $\widehat{p}_b$ and approximate P_b by

$$p_b = \frac{\widehat{p}_b}{w_b} = \frac{1}{w_b M} \sum_{m=1}^{M} \chi_b\big(\widehat{x}^{(m)}\big), \tag{4.4}$$

where the $\widehat{x}^{(m)}$ are drawn independently from $\widehat{\pi}$. We see from (4.3) that $\widehat{E}\,[p_b] = P_b$ as before. However, the key point is that the variance has changed. Again using the

fact that $\chi_b^2 = \chi_b$, we compute

$$\begin{aligned}\widehat{\operatorname{Var}}\,[p_b] &= \frac{1}{w_b^2 M^2}\sum_{m=1}^{M}\widehat{\operatorname{Var}}\left[\chi_b\big(\widehat{x}^{(m)}\big)\right] \\ &= \frac{1}{w_b^2 M}\left(\widehat{E}\,[\chi_b] - \widehat{E}\,[\chi_b]^2\right) = \frac{P_b}{w_b M}\big(1 - P_b w_b\big).\end{aligned} \tag{4.5}$$

Under equidistributed sampling with $P_b w_b = 1/B$, this gives a coefficient of variation

$$\widehat{\operatorname{CV}}\,(p_b) = \frac{1}{P_b}\sqrt{\frac{P_b(1-P_b w_b)}{w_b M}} = \sqrt{\frac{B-1}{M}},$$

which is also independent of the bin index. We have therefore confirmed the guiding heuristic that equidistribution is ideal for sampling across the entire pdf. This idea is referred to in statistical physics as *multicanonical Monte Carlo* (MMC) sampling.

A reality check is in order, because the perfect equidistribution weights $w_b = (BP_b)^{-1}$ are unknown unless the P_b themselves are already known! Fortunately, the Monte Carlo experiment itself gives us some estimates of them, and this fact allows us to set up an iteration. Given some estimates $p_{1,j}, \dots, p_{B,j}$ to the exact values $P_1, \dots, P_B$, we approximate the perfect weights $w_b = (BP_b)^{-1}$ by

$$w_{b,j} = \frac{\gamma_j}{p_{b,j}}, \qquad b = 0, \dots, B, \tag{4.6}$$

where γ_j is used to enforce the normalization (4.2). Equation (4.6) explains why the histogram values $p_{b,j}$ are often called *inverse weights* in the multicanonical method. The weights in turn define a reweighted pdf

$$\widehat{\pi}_j(x) = \frac{\gamma_j \pi(x)}{\displaystyle\sum_{b=1}^{B} p_{b,j}\chi_b(x)}, \tag{4.7}$$

which is used to draw samples and create Monte Carlo estimates

$$\widehat{p}_{b,j} = \frac{1}{M}\sum_{m=1}^{M}\chi_b\big(\widehat{x}^{(j,m)}\big). \tag{4.8}$$

Based on (4.4), our new estimate for the desired P_b is

$$\frac{\widehat{p}_{b,j}}{w_{b,j}} = \frac{p_{b,j}\widehat{p}_{b,j}}{\gamma_j}. \tag{4.9}$$

This seems like a natural choice for $p_{b,j+1}$, so that feedback into the next round of weights is closed and the iteration is completely defined. A sensible way to initialize the iteration is to choose $p_{b,0} = 1/B$, which leads to $w_{b,0} = 1$, i.e., unbiased sampling.

Conceptually, then, we can start with Monte Carlo sampling as usual, pause to assess the computed histogram, adjust the way we sample by trying to more heavily weight underrepresented bins, and iterate. The specific form of (4.9), however, is fatally flawed as a definition of $p_{b,j+1}$. The most glaring problem is that $\widehat{p}_{b,j}$ will be zero if bin b is never hit during iteration j, and (4.9) would then leave $w_{b,j+1}$

undefined. A more subtle problem is that (4.9) can lead to overcompensation in the weights and may not allow the iteration to lock into the desired equidistribution.

Berg [2, 16] proposed a modified iteration that avoids the empty-bin problem and improves reliability. Berg's iteration posits a nearly exponential histogram, which means that the ratios of adjacent values, $r_{b,j} = p_{b+1,j}/p_{b,j}$, are nearly constant. (This assumption holds well only in some applications, such as those for which Berg derived the iteration, but the method appears to work well in other circumstances also.) The iteration focuses on these ratios rather than the values themselves. In our notation, Berg's iteration is

$$r_{b,j+1} = \begin{cases} r_{b,j} & \text{if } \widehat{p}_{b,j} \cdot \widehat{p}_{b+1,j} = 0, \\ r_{b,j} \left[\dfrac{\widehat{p}_{b+1,j}}{\widehat{p}_{b,j}} \right]^{\widehat{g}_{b,j}}, & \text{otherwise.} \end{cases} \tag{4.10}$$

The first option says that if either bin defining $r_{b,j}$ fails to get hit during iteration j, we have insufficient data to justify changing its value. Otherwise, if during iteration j bin $(b+1)$ gets hit more often than does bin b, its inverse weight will be relatively larger in the next iteration, so its weight is reduced. The exponent $\widehat{g}_{b,j}$ is computed in such a manner as to smooth the iteration by accounting for the history of previous iterations, specifically via

$$\widehat{g}_{b,j} = \frac{g_{b,j}}{\sum_{\nu=0}^{j} g_{b,\nu}}, \qquad g_{b,j} = \begin{cases} \dfrac{M \widehat{p}_{b,j} \widehat{p}_{b+1,j}}{\widehat{p}_{b,j} + \widehat{p}_{b,j+1}} & \text{if } \widehat{p}_{b,j} > 0 \text{ and } \widehat{p}_{b+1,j} > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{4.11}$$

These formulas determine the ratios $r_{b,j+1}$ for $b = 1, \ldots, B-1$, which in turn determine the new generation of inverse weights $p_{b,j+1}$ up to a constant scaling factor. Finally, that factor is determined by the normalization $\sum_b p_{b,j+1} = 1$.

Even with Berg's improved iteration strategy, we are left with some serious issues. Most importantly, how do we go about actually drawing samples from the modified probability distribution $\widehat{\pi}_j$? In the growth factor problem, for example, it is not at all clear how to preferentially select matrices whose growth factors lie in a particular bin. Another significant detail is that we cannot compute the normalization constant γ_j in (4.7) without knowledge of the *exact* P_b values. In the next section we show how these issues may be overcome.

5. Markov Chain Monte Carlo (MCMC). The final piece of our puzzle is the *Markov chain*. Markov chains are a common way to implement Monte Carlo experiments, leading to what is widely known as the MCMC method [4, 24]. The MMC variant takes Markov chain implementation for granted, as there is no general alternative for the histogram-based biasing it requires.

A Markov chain is a rule for producing random sequences of elements (usually called states) from the space Ω. The chain is defined by an initial state and by specifying a *transition kernel* $K(x, y)$, which describes the likelihood of the chain changing from state x to state y in one step. That is, if state m is chosen according to the pdf θ_m, then the pdf of state $(m+1)$ is

$$\theta_{m+1}(y) = \int_{\Omega} \theta_m(x) K(x, y)\, dx. \tag{5.1}$$

If the state space Ω is finite, then the pdf of a state is a vector whose elements sum to one, K is a matrix, and (5.1) is just a matrix-vector multiplication.

Markov chains have many interesting properties [9, 19], but we are going to exploit just two of them here. First, under some mild conditions on K, all Markov chains have a *stationary distribution* whose pdf π_0 is defined by

$$\int_\Omega \pi_0(x)K(x,y)\,dx = \pi_0(y), \qquad y \in \Omega. \tag{5.2}$$

Thus, if one state of the chain is distributed according to π_0, then the next state is too. (In the finite case, π_0 is an eigenvector of the transition matrix K.) The stationary distribution is analogous to a steady or equilibrium state in a time-dependent process.

The second property we need from Markov chains continues this analogy: the distribution of state m resembles the stationary distribution as $m \to \infty$. That is, the chain eventually "forgets" where it started and churns out states as though they were being drawn in accordance with π_0. More precisely, we need the ergodic theorem, which states that for any function f, the Monte Carlo estimates for f based on M states of the chain converge (with probability one) as $M \to \infty$ to the expected value of f under π_0.

In the previous section, we set the goal of drawing states from a distribution $\widehat{\pi}_j$ defined by reweighting bins of the histogram. Our method will be to make $\widehat{\pi}_j$ the stationary distribution of a realizable Markov chain and then use the states of the chain to provide samples for our statistics. There are multiple practical ways to do this. We resort to one of the oldest and most popular, known as the *Metropolis–Hastings sampler* [4, 6, 13, 17]. Given a state x, the Metropolis–Hastings sampler creates a "proposed" state y according to a proposal transition probability $q(x,y)$. The chain "accepts" the proposal with a probability given by the number

$$\alpha(x,y) = \min\left\{1, \frac{\widehat{\pi}_j(y)}{\widehat{\pi}_j(x)} \cdot \frac{q(y,x)}{q(x,y)}\right\}. \tag{5.3}$$

If the proposal is accepted, the next state of the Markov chain is y; otherwise, it is x again.

There are many choices for the proposal density $q(x,y)$; we discuss ours for the random walk in section 6 and for growth factors in section 7. There is no hard rule on how to choose q, but it is standard to favor choosing a y that makes only minor changes to x, in keeping with the idea of exploring state space by incremental steps. We are in effect betting that extremely rare cases are found close to the merely rare cases in state space.

Our proposal densities do satisfy the fairly common assumption of symmetry, specifically, $q(y,x) = q(x,y)$. This property is not necessary, but it does make q drop out of the acceptance probability (5.3). Substituting (4.7), we have

$$\alpha(x,y) = \min\left\{1, \frac{\pi(y)}{\pi(x)} \cdot \frac{\sum_b p_{b,j}\chi_b(x)}{\sum_b p_{b,j}\chi_b(y)}\right\}. \tag{5.4}$$

Observe that our last obstacle has been cleared: the unknown normalization γ_j has completely canceled out.

The acceptance probability (5.4) can be interpreted intuitively. Its value is determined by the values of the histogram in the two bins occupied by the current state x and the proposal y. When the histogram is smaller for the proposed y's bin than for x's, the chances of acceptance increase. Conversely, a larger histogram value for the proposal decreases its chance of acceptance. This combination creates a ratchet effect that locks in any proposal that drifts into a bin that seems, so far, less likely to be visited—i.e., the tail of the distribution.

6. An Example: The Random Walk. To illustrate the MCMC method in the context of multicanonical sampling, we return to the random walk of N steps shown in Figure 2. Recall that in this example, each element x of the state space Ω has N bits (representing coin tosses) whose entries are 0 (tails) or 1 (heads), and the variable ρ to be measured is the sum of the elements of x. For this problem we choose a proposal $q(x, y)$ such that

$$q(x,y) = \begin{cases} 1/N & \text{if } x \text{ and } y \text{ differ in exactly one bit,} \\ 0, & \text{otherwise.} \end{cases}$$

In other words, we randomly select a bit from x and change it to get y. It is clear that $q(x, y) = q(y, x)$, as needed in (5.4). Also note that for this problem the original distribution of states π is uniform, so that $\pi(x) = \pi(y)$ in (5.4).

A working MATLAB script for the multicanonical iteration is shown in Figure 4, including the Berg variant of the biasing weights. The process starts with a flat histogram. The outer loop represents the MMC iteration, which initializes the state x (and, for efficiency, all random values needed by the Markov chain), runs the Markov chain, and performs the Berg update on the histogram. Within the Markov chain can be seen a proposed state, found by toggling the result of one randomly chosen coin flip, followed by computation of its Metropolis–Hastings acceptance probability and the acceptance decision. The Berg update is based on (4.10) and (4.11); the ratios $r_{b,j}$ are removed in favor of updating the histogram values themselves directly.

Our Markov chain features a *burn-in period* during which statistics are not collected. The burn-in period is an attempt to let the chain get close to its asymptotic stationary distribution, or "forget" the selection of its initial state. Ensuring or verifying the stationarity of the Markov chain is one of the most difficult and controversial aspects of MCMC simulation, with a growing body of literature devoted to it [4, 5, 7]. In finite state spaces possessing a smallest and largest element, Propp and Wilson [18] famously showed that one can determine with precision how to sample exactly from the stationary distribution. One can even find respectable arguments asserting that burn-in is totally unnecessary in general [11]. Our approach to burn-in is conservative and computationally intensive: we experimented until increasing the burn-in time had no noticeable effect on our results. For this and the growth factor experiments, the burn-in time represents 50% of the computation.

At the end of the script, each column of the variable `p` represents the histogram found in one MMC iteration. Figure 5 shows the evolution of the histogram over 20 iterations. The flat initial histogram corresponds to unbiased Monte Carlo simulation, so that the most frequently observed cases are found early, as in Figure 2. The MMC iteration then weights the unsampled bins much more highly, increasing the likelihood of penetration into the tails of the pdf. By the end of the 20th iteration the histogram is well approximated to graphical accuracy over the entire distribution. The entire state space of over 10^{30} elements has been well sampled using just 2 million states, including those discarded during burn-in periods.

7. Algorithm for Growth Factors. Our MATLAB code for growth factor experiments is very similar to that in Figure 4. One minor difference is that our original matrix distribution (3.1) is not uniform, so that the ratio $\pi(y)/\pi(x)$ in (5.4) is nontrivial. The most important difference, though, is in the selection of the Metropolis–Hastings

```
N = 100;                                % number of walking steps
B = N+1;                                % number of bins
M = 5e4;                                % length of MCMC chaincode
burnin = M;                             % burn-in period for chains
Niter = 20;                             % MMC iterations
p = ones(B,Niter+1)/B;                  % computed histograms
hit = zeros(B,1);                       % hits in each bin for current iter.

randn('state',sum(100*clock));  rand('state',sum(100*clock));

% *** Start MMC iterations ***
for j = 1:Niter
  x = double(rand(1,N)>0.5);  bin_x = 1+sum(x);
  index = ceil(N*rand(1,M+burnin));      % all proposal indices
  acceptval = rand(1,M+burnin);          % all acceptance tests
  for m = 1:M+burnin                     % Markov chain begins
    y = x;  y(index(m)) = 1-x(index(m)); % proposal changes one flip
    bin_y = 1+sum(y);

    % Accept proposal?
    alpha = min(1, p(bin_x,j)/p(bin_y,j) );
    if acceptval(m) < alpha
      x = y;  bin_x = bin_y;
    end

    % Record statistics (after burn-in).
    if m > burnin,  hit(bin_x) = hit(bin_x)+1;  end
  end   % Markov chain

  % Berg update.
  pnew = p(:,j);   % new inverse weights
  for b=1:B-1
    if (hit(b+1)*hit(b) == 0)
      pnew(b+1) = pnew(b)*(p(b+1,j)/p(b,j));
    else
      g(b,j) = hit(b+1)*hit(b) / (hit(b+1)+hit(b));
      g_hat(b) = g(b,j)/sum(g(b,1:j));
      pnew(b+1) = pnew(b)*(p(b+1,j)/p(b,j))*((hit(b+1)/hit(b))^g_hat(b));
    end
  end
  p(:,j+1) = pnew/sum(pnew);             % normalize histogram
  hit(:) = 0;                            % reset hit counter
end   % MMC iteration
```

Fig. 4 *MATLAB script for an MMC iteration applied to the random walk.*

proposal density $q(x, y)$. We have chosen a random walk model, in which

$$q(x,y) = f\left(\frac{y-x}{\sigma}\right), \tag{7.1}$$

where f is a student's t-distribution with 8 degrees of freedom and σ is a parameter to be explained in the next paragraph. Compared to a normal distribution for f, the t-distribution's larger tails were found to improve the rate at which the Markov chains approach stationarity. In practice, to get the proposal y we scale by σ a matrix whose

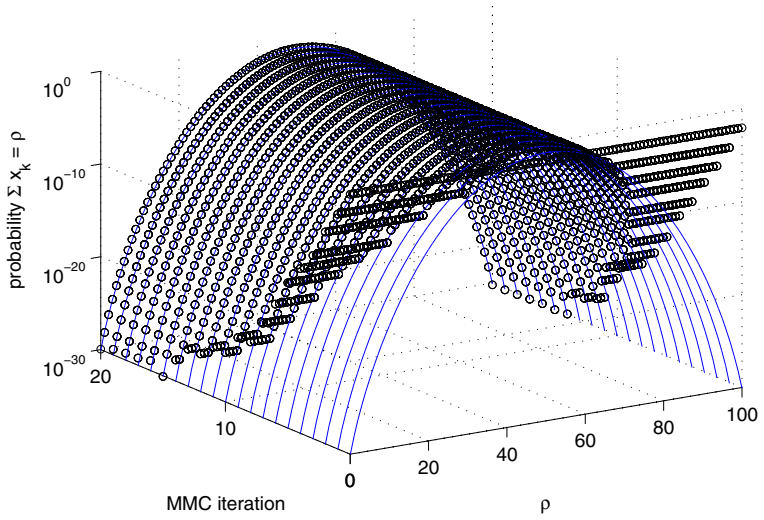

Fig. 5 *Evolution of the pdf histogram with the MMC iteration for the random walk with $N = 100$ coin tosses. (Circles show the computations; the solid curves are copies of the exact pdf.) Initially the histogram is flat. The most frequently observed cases are found first, around the center of the distribution. Then the iteration progresses into the tails.*

entries are drawn from the t-distribution using `trnd` in MATLAB, and add it to x. Note that $q(y, x) = q(x, y)$, as we have required.

Initially we let $\sigma = 1/n$ for experiments on $n \times n$ matrices. We update σ based on the acceptance rate—the proportion of Metropolis–Hastings proposals that are accepted as steps. Roberts, Gelman, and Gilks [20] recommended an acceptance rate of roughly 25%. If during burn-in we find an acceptance rate greater than 40%, we interpret the proposals as too conservative and we increase σ. Conversely, if the acceptance rate falls below 10%, we decrease σ. Overall we see σ decrease as states get into the tail, because once there large perturbations almost always propose states with more common growth factors, and the acceptance rate drops.

Figure 6 shows one evolution of the histogram over 50 iterations in the case of 8×8 matrices. For this run MMC makes excellent progress into the tail for the first ten or so iterations, then stagnates for another 20 iterations before reaching a breakthrough and continuing progress. During the stagnation the proposal acceptance rate stayed mostly between about 30% and 65%, only once getting small enough to trigger a reduction in σ. Without generalizing too much from one example, this suggests that our simple proposal adaptation strategy may not be completely effective. The value of σ at the end of the iteration was about 3×10^{-5}.

The U factor of the final matrix of the experiment was (after normalization by the largest element)

$$\begin{bmatrix} 0.5361 & 0.157 & 0.2923 & 0.1393 & 0.1605 & -0.01271 & -0.04148 & 1 \\ & -0.3612 & 0.1187 & 0.03457 & 0.4437 & -0.08554 & 0.2254 & -1.989 \\ & & -0.596 & 0.3819 & 0.2222 & -0.1974 & 0.01469 & 3.974 \\ & & & 0.4361 & 0.4217 & 0.1419 & -0.2547 & -6.988 \\ & & & & -0.2346 & 0.06456 & -0.03672 & 14.11 \\ & & & & & -0.3469 & -0.0373 & -25.56 \\ & & & & & & 0.6609 & -50.03 \\ & & & & & & & -99.65 \end{bmatrix}.$$

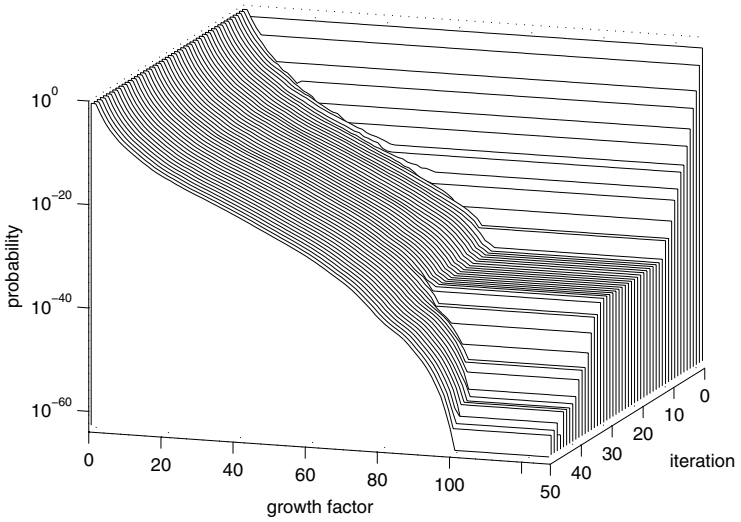

Fig. 6 *Evolution of the pdf histogram for the growth factors of* 8×8 *matrices.*

The last column of this factor (ignoring signs) looks much like that in the worst-case scenario of (2.3), with an imperfect doubling in magnitude from each row to the next. Theorem 2.2 of [15] explains that this is always the case.

The last change to our code over the basic format in Figure 4 was to try different restarting strategies between MMC iterations. Rather than always reinitializing with a matrix drawn randomly from π, we instead can start with an arbitrary matrix with a known growth factor ρ_0, using the aforementioned theorem from [15]. By varying ρ_0 for different runs of the algorithm, we get some diagnostic information about the extent to which the starting points of the Markov chains influences the results—that is, the extent to which the Markov chains have not reached stationarity.

The code we used for our experiments in the next section is available for download at the MATLAB Central website, from the URL http://tinyurl.com/nt58u.

8. Results. We ran our algorithm in MATLAB on random matrices of dimensions 8, 16, and 32. The computations were done in parallel on a 24-node Opteron cluster using the Distributed Computing Toolbox. Each instance was run for 70 MMC iterations, each iteration drawing one million matrices, including burn-in. As described at the end of section 7, for each matrix size we ran multiple independent instances using different reinitialization strategies between MMC iterations: one case with no reinitialization, one case with reinitialization drawn from the original distribution π, and ten cases with randomly constructed matrices, each case starting with a specified growth factor ρ_0, where the ten runs chose different values of ρ_0 at well-spread locations.

All of the histograms resulting from our runs are shown in Figure 7. The leveling off of each curve shows the farthest point in the tail visited by that MMC realization. While the extent of penetration varies widely from run to run, the various instances agree well for most of those parts of the histograms based on visited bins. Furthermore, there is no observable correlation between the reinitialization strategy and the depth of penetration into the tail. We take these observations as positive indications that

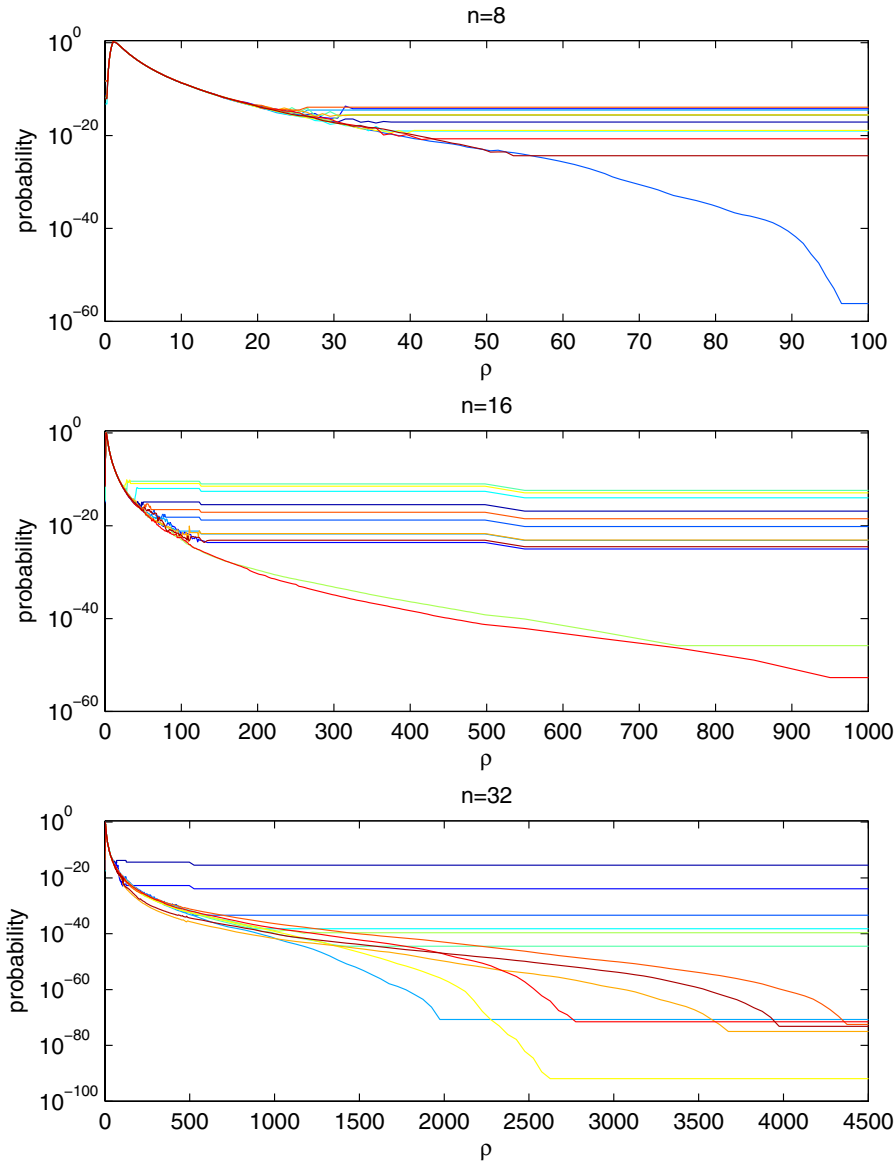

Fig. 7 *Pdfs for growth factors of random matrices of dimension* 8, 16, *and* 32. *The different curves represent independent runs of the MMC iteration using different MCMC reinitializations, as explained in the text.*

the results of our Markov chains are not strongly dependent on their starting points, i.e., that the burn-in times were sufficient.

We feel justified, for example, in estimating the probability of encountering a not-very-damaging growth factor $\rho = 40$ to be about 10^{-20} in the 8×8 case. By comparison, there are believed to have been only 10^{17} or so seconds since the beginning

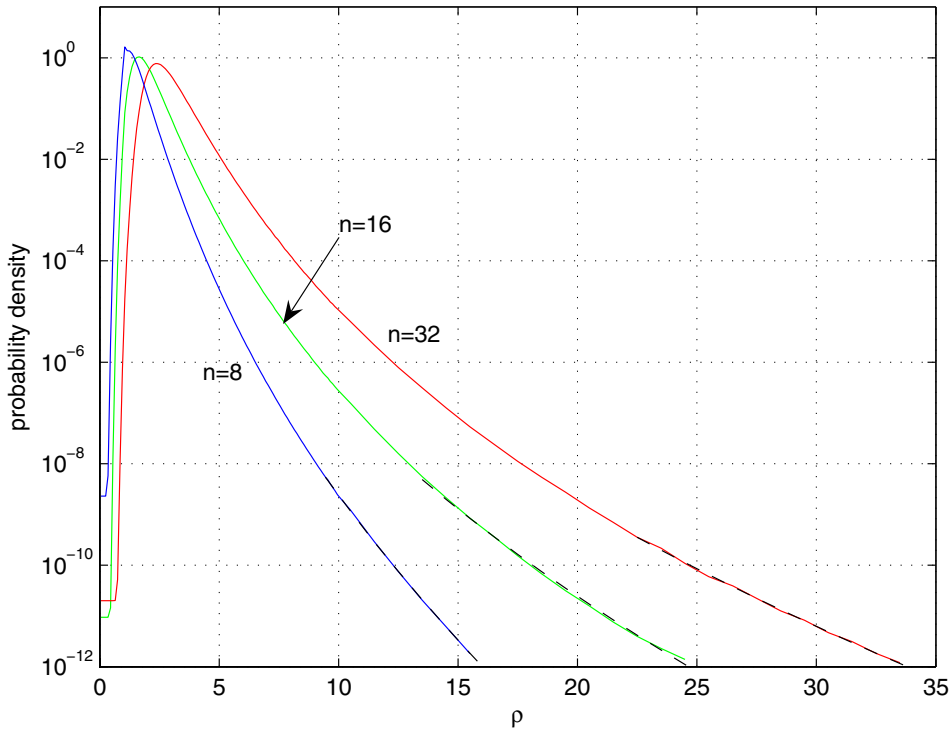

Fig. 8 *The colored lines show our best pdfs for growth factors of random matrices with dimensions* 8, 16, *and* 32. *Dashed black lines show the results of our nonlinear fits to the data, as given by* (8.1) *and explained in the text.*

of the universe! Moreover, even with a concerted effort to exert significant bias in our explorations of the state spaces, none of our experiments ever encountered a matrix in the bin containing the maximum achievable growth factor.

In Figure 8 we exhibit our data for dimensions 8, 16, and 32 down to a probability level of 10^{-12}. These were found by averaging the results of individual trials. Based on the excellent agreement in Figure 7 down to this level, we feel confident about these results in terms of the accuracy of the graph (except for the leveling off for small values of ρ, which were not a focus of the study). One can check that they also match well the unbiased Monte Carlo experiments shown in Figure 3.

Figure 8 also shows the results of fitting a function of the form

$$f(\rho) = a_1 \exp\left(-a_2 \rho^{a_3}\right) \tag{8.1}$$

to the tail end of each computed curve. (Such a functional form is compatible with the decay rates predicted in [21], as mentioned in section 2.) The parameters were found by applying MATLAB's `lsqnonlin` to find the minimum least-squares difference between $\log f$ and the log of the computed data. The black dashed curves show the part of the data selected and the resulting fit. The fitting parameters are shown in Table 1.

9. Conclusions. The MCMC method is a common and practical way to explore the distributions of random variables over large state spaces. The multicanonical variant from statistical physics aims to correct its greatest weakness, the exploration

Table 1 *Least-squares parameters for the fitting function* (8.1)*, for the data shown in Figure* 8.

n	a_1	a_2	a_3
8	30006	9.8309	0.48683
16	30006	9.8490	0.42079
32	30007	8.8811	0.41239

of rare events. Multicanonical MCMC (we resist, but only barely, the temptation to call it $(\mathrm{MC})^3$) accomplishes this in a way that does not require insights into the random process being evaluated. Instead, one only needs the ongoing results of the experiment in order to adjust the sampling process. This makes the method widely applicable with little adjustment—indeed, our code could easily be modified to change the state space to complex, Toeplitz, symmetric, banded, or orthogonal matrices, or to change the random variable to, say, condition numbers rather than growth factors.

Of course, the results of MCMC iterations are, in most cases, not rigorous. Furthermore, these methods offer many choices and parameters that can have strong effects on the quality and speed of the results. The references give some recommendations on how to choose these, but wide variations of opinions can be found. In our experience the most important choice is the Metropolis–Hastings proposal density $q(x, y)$, which must strike a balance between minuscule changes to the current state, leaving too much unexplored, and major changes that usually lead out of the tail and get rejected. Nor can issues of Markov chain initialization, convergence, and diagnostics be overlooked. Still, as a tool for the difficult problem of simulating rare events, the multicanonical method puts one in mind of another line from *Hamlet*: "Though this be madness, yet there is method in 't."

Acknowledgment. We are grateful to Nick Trefethen for his enthusiastic and insightful feedback.

REFERENCES

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3rd ed., SIAM, Philadelphia, 1999.
[2] B. A. Berg, *Introduction to multicanonical Monte Carlo simulations*, Fields Inst. Commun., 26 (2000), pp. 1–24.
[3] C. B. Boyer, *A History of Mathematics*, John Wiley & Sons, New York, 1991.
[4] S. P. Brooks, *Markov chain Monte Carlo method and its application*, The Statistician, 47 (1998), pp. 69–100.
[5] S. P. Brooks and G. O. Roberts, *Convergence assessment techniques for Markov chain Monte Carlo*, Statist. Comput., 8 (1998), pp. 319–335.
[6] S. Chib and E. Greenberg, *Understanding the Metropolis-Hastings algorithm*, Amer. Statist., 49 (1995), pp. 327–335.
[7] M. K. Cowles and B. P. Carlin, *Markov chain Monte Carlo convergence diagnostics: A comparative review*, J. Amer. Statist. Assoc., 91 (1996), pp. 883–904.
[8] J. W. Demmel, *Applied Numerical Linear Alegbra*, SIAM, Philadelphia, 1997.
[9] P. Diaconis and D. Freedman, *Iterated random functions*, SIAM Rev., 41 (1999), pp. 45–76.
[10] L. V. Foster, *Gaussian elimination with partial pivoting can fail in practice*, SIAM J. Matrix Anal. Appl., 15 (1994), pp. 1354–1362.
[11] C. Geyer, *Burn-in is unnecessary*, available online from http://www.stat.umn.edu/∼charlie/mcmc/burn.html.
[12] G. Golub and C. V. Loan, *Matrix Computations*, 3rd ed., Johns Hopkins University Press, Baltimore, MD, 1996.
[13] W. K. Hastings, *Monte Carlo sampling methods using Markov chains and their applications*, Biometrika, 57 (1970), pp. 97–109.

[14] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia, 1996.
[15] N. J. Higham and D. J. Higham, *Large growth factors in Gaussian elimination with pivoting*, SIAM J. Matrix Anal. Appl., 10 (1989), pp. 155–164.
[16] R. Holzlöhner and C. R. Menyuk, *Use of multicanonical Monte Carlo simulations to obtain accurate bit error rates in optical communications systems*, Optics Lett., 28 (2003), pp. 1894–1896.
[17] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, *Equations of state calculations by fast computing machines*, J. Chem. Phys., 21 (1953), pp. 1087–1092.
[18] J. G. Propp and D. B. Wilson, *Exact sampling with coupled Markov chains and applications to statistical mechanics*, Random Structures Algorithms, 9 (1996), pp. 223–252.
[19] C. P. Robert and G. Casella, *Monte Carlo Statistical Methods*, 2nd ed., Springer-Verlag, New York, 2004.
[20] G. O. Roberts, A. Gelman, and W. R. Gilks, *Weak convergence and optimal scaling of random walk Metropolis algorithms*, Ann. Appl. Probab., 7 (1997), pp. 110–120.
[21] L. N. Trefethen and D. Bau, III, *Numerical Linear Algebra*, SIAM, Philadelphia, 1997.
[22] L. N. Trefethen and R. S. Schreiber, *Average-case stability of Gaussian elimination*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 335–360.
[23] S. Ulam, *Adventures of a Mathematician*, University of California Press, Berkeley, CA, 1991.
[24] W. R. Gilks, S. Richardson, and D. J. Spiegelhalter, *Markov Chain Monte Carlo in Practice*, 1st ed., Chapman & Hall, Boca Raton, FL, 1996.
[25] J. H. Wilkinson, *The Algebraic Eigenvalue Problem*, Oxford University Press, New York, 1965.
[26] S. J. Wright, *A collection of problems for which Gaussian elimination with partial pivoting is unstable*, SIAM J. Sci. Comput., 14 (1993), pp. 231–238.