General Disclaimer

One or more of the Following Statements may affect this Document

- This document has been reproduced from the best copy furnished by the organizational source. It is being released in the interest of making available as much information as possible.

- This document may contain data, which exceeds the sheet parameters. It was furnished in this condition by the organizational source and is the best copy available.

- This document may contain tone-on-tone or color graphs, charts and/or pictures, which have been reproduced in black and white.

- This document is paginated as submitted by the original source.

- Portions of this document are not fully legible due to the historical nature of some of the material. However, it is the best reproduction available from the original submission.

Golden Ratio Versus Pi as Random Sequence Sources for Monte Carlo Integration

S.K. Sen[a,*], Ravi P. Agarwal[a], and Gholam Ali Shaykhian[b]

[a]*Department of Mathematical Sciences, Florida Institute of Technology, 150 West University Boulevard, Melbourne, FL 32901-6975, United States*
[b]*National Aeronautics and Space Administration (NASA), Engineering Directorate, NE-C1, Kennedy Space Center, FL 32899, United States*

Abstract

The algebraic irrational number golden ratio $\varphi = (1+\sqrt{5})/2 =$ one of the two roots of the algebraic equation $x^2 - x - 1 = 0$ and the transcendental number $\pi = 2\sin^{-1}(1) =$ the ratio of the circumference and the diameter of any circle both have infinite number of digits with no apparent pattern. We discuss here the relative merits of these numbers as possible random sequence sources. The quality of these sequences is not judged directly based on the outcome of all known tests for the randomness of a sequence. Instead, it is determined implicitly by the accuracy of the Monte Carlo integration in a statistical sense. Since our main motive of using a random sequence is to solve real world problems, it is more desirable if we compare the quality of the sequences based on their performances for these problems in terms of quality/accuracy of the output. We also compare these sources against those generated by a popular pseudo-random generator, viz., the Matlab *rand* and the quasi-random generator *halton* both in terms of error and time complexity. Our study demonstrates that consecutive blocks of digits of each of these numbers produce a good random sequence source. It is observed that randomly chosen blocks of digits do not have any remarkable advantage over consecutive blocks for the accuracy of the Monte Carlo integration. Also, it reveals that π is a better source of a random sequence than φ when the accuracy of the integration is concerned.

1. Introduction

The distribution of digits in an irrational number[1] — both algebraic and transcendental — has not been extensively explored. Only to a limited extent this has been discussed in [1]. So far as a rational number is concerned, it could have either infinite number of digits in the conventional decimal number system or it could have a finite number of digits in a number system (not necessarily decimal). Whenever a rational number has infinite number of digits, these digits are necessarily recursive in some form(s). Hence the distribution of its digits is usually known/can be determined or even predicted. Our conjecture is that digits of a rational number cannot be uniformly randomly distributed. Much effort is not required to prove/disprove this conjecture. However, our inquisitiveness is with respect to the distribution of digits in an irrational number in which the knowledge of the current and previous digits does not help to predict the next digit.

*Corresponding author: Tel.:+1 321 674 7714; fax: +1 321 674 7412

- *E-mail addresses:* sksen@fit.edu (S.K. Sen), agarwal@fit.edu (Ravi P. Agarwal), ali.shaykhian@nasa.gov (Gholam Ali Shaykhian)

[1] An irrational number is one which cannot be expressed as the ratio of two integers. This implies that an irrational number will have infinite number of digits for its exact representation.

To start with we have chosen one popular irrational number, viz., golden ratio φ from among the algebraic numbers[2] and one extremely used more popular irrational number, viz., π from among the transcendental numbers[3]. We have studied the distribution of the digits of φ [1] as well as those of π, which are then employed as a random sequence source (RSS) to compute the values of Monte Carlo integrals. The question "Which is better — φ or π — in uniformity in distribution of its digits as well as in computing an integral?" is attempted for an answer. In fact, similar exploration could be carried out for other well-known/famous irrational numbers or even ordinary irrational numbers. Some of the ordinary irrational numbers could possibly turn out to be remarkable/outstanding in terms of uniformity of their digits or randomness of their digits. Some of the other famous irrational numbers are the Hilbert number $2^{\sqrt{2}} \approx 2.66514414269023$, the Euler-Mascheroni constant $\gamma = \lim_{n\to\infty}(\sum_{k=1}^{n}\frac{1}{k} - \ln n) \approx 0.57721566490153$, and the numbers $i^i = e^{-\pi/2} \approx 0.207879576350762$, $\pi^e \approx 22.4591577183611$ (believed (not proved) to be a transcendental number) and $e^\pi \approx 23.1406926327793$. Such an exploration will help one to decide which irrational number should be used as an RSS not only for Monte Carlo integration but also for randomized/evolutionary algorithms such as ant system approaches, genetic algorithms, and simulated annealing. Some randomized algorithms perform better with a quasi-random sequence while others perform better with a pseudo-random sequence [2-36]. It may be noted that quasi-random sequences are more uniformly distributed (with less discrepancy) than pseudo-random sequences. One important advantage of using an irrational number for an RSS is that it obviates the need of employing a random number generator. We would need, for a reasonable real world problem, a generation of truly large random sequence — a sequence of billions or possibly trillions of random numbers — to solve multidimensional problems. Here the extra time of generation of random numbers are eliminated since we can simply pick out blocks of digits consecutively from an irrational number. However, such an extra time appears near about the same as that required for retrieval of numbers. An efficient storage and retrieval system possibly could improve the time complexity and could even prove better than instant generation of random numbers. It may be pointed out that not all irrational numbers are having digits randomly occurring. For instance, the Liouville number $\approx 0.11000100000000000000000010000$ which has a 1 in the 1st, 2nd, 6th, 24th, 120th etc. places and 0s elsewhere, is an irrational number where 0's and 1's are not random. These are exactly predictable.

In section 2, we describe simple Monte Carlo procedures for numerical integrations by appropriately slicing the n-dimensional domain after mapping it onto a standard domain. Since our main purpose of using a random sequence is to solve a real world problem, it is more desirable if we compare the popular random number generators along with the concerned irrational numbers based on their performances in terms of quality (error bounds) and cost (computational/time complexity) of the solutions that they produce. In section 3, we present the numerical results considering typical single integrals. Section 4 comprises conclusions.

---

[2] An algebraic number is a root of any rational polynomial.

[3] Transcendental number is a number that is not the root of any polynomial (of finite degree) whose coefficients are rational (or integer) numbers. That is, it is not the root of any integer polynomial implying that it is not an algebraic number of any degree. Every real transcendental number must be irrational since a rational number is, by definition, an algebraic number of degree 1. Some of the proven transcendental numbers are $ln\ 2$, e, π, $2^{\sqrt{2}}$, $\sin(1)$, $\Gamma(1/3)$, and e^π.

2. Monte Carlo Integration

A Monte Carlo (MC) procedure is a method based on using uniformly distributed random numbers (RNs). This procedure is sometimes preferred over a numerical quadrature when the integrand is violently fluctuating or involves long trigonometric/special functions. We have chosen here the Monte Carlo integration for the purpose of comparing π and φ as RSS's. Also, compared are π and φ against the popular pseudo-random generator Matlab rand and quasi-random generator halton.

Let the single integral be

$$I = \int_a^b f(x)dx = \int_\alpha^\beta f(x)dx, \qquad (1)$$

where the function $f(x)$ is continuous, non-negative, and single real-valued in the interval $[a,b]$. The MC procedure is as follows.

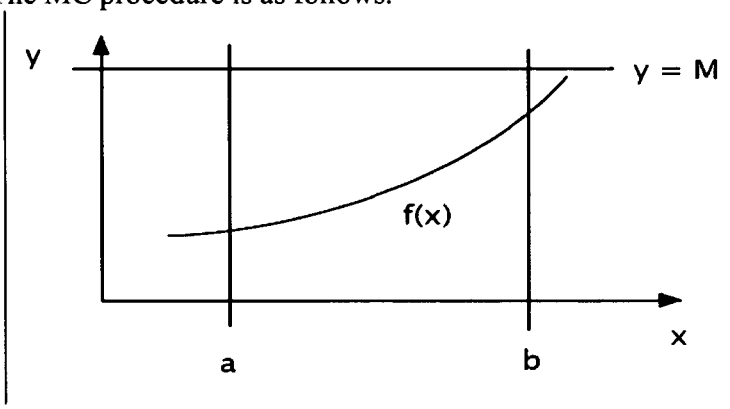

Figure 1 MC integration of $f(x)$ in $[a,b]$ determines the area under the curve $y = f(x)$ from $x = a$ to $x = b$.

S.1 Choose $M \geq$ largest value of $f(x)$ in the interval $[a,b]$. Initialize $hit = 0$ and $N = 5000$ (say).
S. 2 Generate/choose a pair of random numbers (x,y) such that $x \in [a,b]$ and $y \in [0,M]$. If $f(x) \leq y$ then it is a hit and hence update $hit = hit + 1$.

S. 3. Continue S. 2 N times. The integration value is then $I = \frac{hit}{N}(b-a)M$.

This MC procedure can be generalized for the integrand $f(x)$ which has both negative as well as positive values in the interval $[a,b]$ by introducing the minimum value R of $f(x)$, replacing $y \in [0,M]$ by $y \in [R,M]$, and appropriately determining hit with proper signs. An alternative way is to divide the integration into or more parts such that in each part the function $f(x)$ is either wholly nonnegative or wholly negative. WE then apply for each part the procedure similar to the foregoing MC procedure.

However, we prefer to use MC method on an integral with fixed limits of integration, say, 0 and 1 since the summation-based MC method (presented later) remains invariant. Any integral with limits of integration $[a,b]$ can be mapped onto the integral with limits of integration [0,1] where the new integrand is a linear transformation of the original integrand. For example, the integral

$$I = \int_a^b f(x)dx = \int_0^1 F(t)dt .$$

where the new integrand $F(t) = pf(pt+q)$, $p = b-a, q = a, dx = pdt$. These values are derived from the linear transformation $x = pt+q$. When $x = a, t = 0 \Rightarrow q = a$. When $x = b, t = 1 \Rightarrow p = b-a$ .If $a = 0, b = \infty$, then

$$I = \int_0^\infty f(x)dx = \int_0^r f(x)dx + \int_r^\infty f(x)dx ,$$

where r is a finite positive real number. The interval $[0,r]$ can be changed, as before, to [0,1]. For the interval $[r,\infty]$, use the nonlinear transformation $x = 1/(pt+q)$. The interval $[-\infty,\infty]$ can be similarly changed (mapped) to finite intervals [37].

Without any loss of generality, we now consider the single, double, and triple integrals

$$I_1 = \int_0^1 f(x)dx,\ I_2 = \int_0^1\int_0^1 f(x,y)dydx,\ I_3 = \int_0^1\int_0^1\int_0^1 f(x,y,z)dzdydx .$$

The unbiased MC estimate of the values of I_1, I_2, I_3 are

$$I_1 = \frac{1}{r}\sum_{i=1}^{r} f(x_i),\quad I_2 = \frac{1}{r^2}\sum_{i=1}^{r}\sum_{j=1}^{r} f(x_i,y_j),\quad I_3 = \frac{1}{r^3}\sum_{i=1}^{r}\sum_{j=1}^{r}\sum_{k=1}^{r} f(x_i,y_j,z_k) ,$$

where each of x_i, y_j, z_k is chosen uniformly randomly in the interval [0,1]. The constant $r = N$ in the expression of I_1 is the number of random numbers used to compute I_1. The constant $r^2 = N$ in the expression of I_2 is the number of random numbers used to compute I_2 while the constant $r^3 = N$ in the expression of I_3 is the number of random numbers used to compute I_3.The generalization to an n-dimensional integral is straight-forward. The MC estimate tends to the true value of the integral as $N \to \infty$. We, however, restrict ourselves to single integrals for the purpose of comparison between π and φ. The MC multiple integration will be discussed in a future work.

Let the function $f(x)$ be square integrable[4]. From the central limit theorem, the standard error tends to $1/\sqrt{N}$, the variance of the MC estimate I_1 is $S = \frac{1}{N-1}\sum_{i=1}^{N}(f(x_i)-I_1)^2$, and an estimate of the error in I_1 is $E = \sqrt{S}/N$. However, a general numerical way of the relative error estimation [38] is to compute I_1 with $N = 5000$ uniformly distributed RNs (pseudo-RNs or preferably quasi-RNs) assuming sufficiently large precision RNs. Call the resulting integration value as the quantity of lower order accuracy Q'. Once again compute I_1 with $N = 50000$ uniformly distributed RNs. Call the resulting integration value as the quantity of higher order accuracy Q. The relative error is

---

[4] A function is square integrable if $\int_{-\infty}^{\infty} |f(x)|^2 dx$ is finite.

then $(Q-Q')/Q$. If the precision (length) of the RNs is not sufficiently large then even with higher number of RNs, accuracy may not improve. It may sometimes even become worse due to clustering.

Dividing the interval/domain of integration For the sake of obtaining desirable accuracy in the integration value, it is, in general, necessary to keep a provision of writing the given integral into a series of integrals by appropriately slicing/dividing the interval/domain into a number of subintervals/subdomains. Integrate the function in each subdomain and add the resulting integration values. The subdomains may or may not be equal. We, however, have considered them equal irrespective of whether the function (integrand) is ill-posed or not in a subdomain. For a real-world problem, such a programming simplification, in a Matlab environment or for that matter in most other programming environment, does not have a significant negative effect on the computational complexity or on accuracy. Since the sensitivity of the integrand could vary from one subdomain to another, accuracy of computation will also vary from one subdomain to another. When we add up the resulting integration values with unequal accuracies, the final integration value will be less accurate than the least accurate integration value. However, when one works with a precision of 15 digits with appropriate equal subdomains, the relative error propagation is most unlikely to hit the fifth digit in the final integration value. It may be noted that a real world engineering implementation cannot be usually more accurate than four significant digits. This is because no measuring device can measure, in general, a quantity with more than 0.005% accuracy [38]. We can write, allowing $\alpha = a_1$ and $\beta = a_n$, $p_i = a_i - a_{i-1}$, $q_i = a_{i-1}$, $a_i > a_{i-1}$, $i = 2(1)n-1$, (i.e., $i = 2,3,4,\cdots,n-1,$)

$$I = \int_{\alpha}^{\beta} f(x)dx = \int_{a_1}^{a_n} f(x)dx = \sum_{i=2}^{n-1} \int_{0}^{1} f(p_i t + q_i) p_i dt$$

If $f(x) = x\sin(1/x)$, $\alpha = a_1$ and $\beta = a_n$, $p_i = a_i - a_{i-1}$, $q_i = a_{i-1}$, $a_i > a_{i-1}$, $i = 2(1)n-1$, then

$$I = \int_{\alpha}^{\beta} x\sin(1/x)dx = \sum_{i=2}^{n-1} \int_{0}^{1} (a_i - a_{i-1})[(a_i - a_{i-1})t + a_{i-1}]\sin(\frac{1}{(a_i - a_{i-1})t + a_{i-1}})dt.$$

If $f(x) = \dfrac{\sin x}{x}$, $\alpha = a_1 = 0$ and $\beta = a_n = \infty$, then, using the transformation $x = 1/(pt+q)$ [37], we have

$$\int_0^{\infty} \frac{\sin x}{x} dx = \int_0^{1} [\frac{\sin t}{t} + \frac{1}{1-t}\sin(\frac{1}{1-t})]dt = \frac{\pi}{2} \text{ (exact value).}$$

Considering the second integral, we have $\alpha = 0, \beta = 1$ if we desire to change the interval of integration $[0,\infty)$ to $[0,1]$. However, we can instead choose ∞ as 20000 (say) for sufficient numerical accuracy for real world implementation and write

$$\int_0^{\infty} \frac{\sin x}{x} dx \approx \int_0^{20000} [\frac{\sin x}{x} dx = 20000 \int_0^{1} \frac{\sin(20000t)}{20000t} dt \approx \frac{\pi}{2}.$$

It may be observed that as $t \to 0$, $\sin(20000t)/(20000t) \to 1$ (in the limit). Or, in other words, if we choose $t = 10^{-10}$, we automatically get, in Matlab, $\sin(20000t)/(20000t)$ as 0.999999999999333

which is correct up to 12 decimal digits (sufficiently accurate for a real world application). Thus we may replace the lower limit of integration, viz., 0 by 10^{-10} whenever a division by zero could occur. We may also vary the decimal number 20000 and observe the change (if any) in the value of integration up to, say, seven digits.

3. Numerical Experiments

We have considered several typical single test integrals having variable sensitivity over the interval of integration. Also, we have considered an integral whose solution is not known. Our focus is to compare π and φ to determine which one is a better source of a random sequence although we have brought in the pseudo-random generator, viz. the widely used Matlab rand generator and the quasi-random generator, viz., the halton generator [2, 3, 11]. The quasi-random generators for Monte Carlo integration are expected to perform better since these produce more uniformly distributed (with less discrepancy) random sequences than those produced by pseudo-random generators. Our numerical experiments also depict this fact.

Let the general form of our single integral be as given in the equation (1). We have used the following notations in Table 1 that depicts Monte Carlo integration values for the following single integrals I_1 *and* I_2 along with relative error in each of rand, halton, pi, and phi: RSS = random sequence source/generator, n = number of subintervals for the interval $(\beta-\alpha)$, N = 5000 = number of random numbers used in each subinterval, True value = True/exact value of the integral.

The single integrals along with their true/exact values, that we have considered are

(i) $I_1 = \int_0^1 x^2 dx = \frac{1}{3}$,

(ii) $I_2 = \int_0^\infty \frac{\sin x}{x} dx = \int_0^1 [\frac{\sin x}{x} + \frac{1}{1-x}\sin(\frac{1}{1-x})]dx = \frac{\pi}{2} \approx \int_{10^{-10}}^{10000} \frac{\sin x}{x} dx$,

(iii) $I_3 = \int_0^\infty (\frac{\sin x}{x})^2 dx = \frac{\pi}{2} \approx \int_{10^{-10}}^{10000} (\frac{\sin x}{x})^2 dx \approx 1.5707963267949$,

(iv) $I_4 = \int_0^\infty \frac{1}{1+x^2} dx = \int_0^1 [\frac{1}{1+x^2} + \frac{1}{(1-x)^2+1}]dx = \frac{\pi}{2} \approx 1.5707963267949$,

(v) $I_5 = \int_0^1 x \sin\frac{1}{x} dx \approx \int_{10^{-10}}^1 x \sin\frac{1}{x} dx \approx 0.3785283345$

(the value 0.3785283345 of the integral I_5 is computed dividing the interval $[10^{-10}, 1]$ into one million equal subintervals since the exact/true value of I_5 is not known.),

(vi) $I_6 = \int_{-\infty}^\infty \frac{\cos 3x}{x^2+4} dx = 2\int_0^\infty \frac{\cos 3x}{x^2+4} dx = \frac{\pi}{2e^6} \approx 0.00389361481 \approx 2\int_0^{10000} \frac{\cos 3x}{x^2+4} dx$,

(vii) $I_7 = \int_{-\infty}^\infty \frac{x \sin x}{1+x^2} dx = 2\int_0^\infty \frac{x \sin x}{1+x^2} dx = \frac{\pi}{e} \approx 1.155734979 \approx 2\int_0^{10000} \frac{x \sin x}{1+x^2} dx$,

(viii) $I_8 = \int_{10^{-10}}^1 \ln x dx = [x \ln x - x]_1 - [x \ln x - x]_{10^{-10}} \approx -.999999997597415$,

(ix) $g(\omega) = \int_0^{\infty} \frac{\cos \omega x \sin x}{x} dx = \begin{cases} \pi/2, 0 < \omega < 1 \\ \pi/4, \omega = 1 \\ 0, w > 1 \end{cases}$

We take $\omega = 0.5, 1, 2$ for the comparison. Our $I_9 = g(0.5), I_{10} = g(1), I_{11} = g(2)$. Also as in the earlier integrals, we choose the limits of integration as $[10^{-10}, 10000]$ for the actual limits $[0, \infty]$.

Tables 1 and 2 provide us the MC integration (MCI) value of each of the integrals when pseudo-random generator Matlab rand, quasi-random generator halton, RSS's pi (π) and golden ratio (φ) are used. Wherever the limit of integration zero causes division by zero in a numerical computation, zero(0) is taken as 10^{-10}, otherwise zero is kept as such. For the limit of integration ∞, we have taken 10000. This is expected to provide four significant digit accuracy which is, in general, acceptable in most applications. It may be noted in this connection that a measuring device cannot measure usually an accuracy greater than 0.005% [38]. However, for an intermediate integration value used for further computation, we should compute the value with higher accuracy/precision so that the final output to be used in real world implementation has a 0.005% accuracy.

Table 1 Monte Carlo integration (MCI) along with time complexity using random generators *rand* and *halton*, random sequence sources pi and phi for the integrals $I_1 = \int_0^1 x^2 dx = \frac{1}{3}$ and

$$I_2 = \int_{10^{-10}}^{10000} \frac{\sin x}{x} dx \approx 1.5707963267949$$

Integral with True Value	*RSS*	*n*	*MCI value*	*Relative Error*	*Time(sec)*
I_1 =0.3333333333	*rand*	*1000*	*0.3313420241*	*0.0059739277*	*0.421*
		10000	*0.3331328144*	*0.0006015569*	*3.813*
		100000	*0.3333133046*	*0.0000600861*	*135.948*
	halton	*1000*	*0.3313370970*	*0.0059887089*	*0.656*
		10000	*0.3331333680*	*0.0005998961*	*3.891*
		100000	*0.3333133335*	*0.0000599996*	*132.504*
	pi	*1000*	*0.3313377003*	*0.0059868991*	*0.438*
		10000	*0.3331334104*	*0.0005997687*	*4.034*
		100000	*0.3333133374*	*0.0000599877*	*152.418*
	phi	*1000*	*0.3313357881*	*0.0059926357*	*0.359*
		10000	*0.3331332186*	*0.0006003443*	*3.970*
		100000	*0.3333133182*	*0.0000600453*	*143.128*
I_2 =1.5707963268	*rand*	*1000*	*1.6210367932*	*–0.0319840743*	*1.494*
		10000	*1.5737041052*	*–0.0018511492*	*14.089*
		100000	*1.5705198953*	*0.0001759818*	*235.874*
	halton	*1000*	*1.5741008831*	*–0.0021037459*	*2.415*
		10000	*1.5708375869*	*–0.000262670*	*14.147*
		100000	*1.5708984346*	*–0.0000650039*	*239.821*
	pi	*1000*	*1.558832916*	*0.0075840738*	*1.483*
		10000	*1.5704023188*	*0.0002508333*	*14.440*
		100000	*1.5708585727*	*–0.0000396270*	*235.542*
	phi	*1000*	*1.5905901066*	*–0.0126011116*	*1.498*
		10000	*1.5723190078*	*–0.0009693688*	*14.119*
		100000	*1.5710505371*	*–0.0001618353*	*244.999*

To conserve space we include in Table 2 Monte Carlo integration values for the following single integrals I_3, I_4, I_5, I_6, I_7, I_8, $g(0.5)$, $g(1)$, and $g(2)$ along with relative error in each of rand, halton, pi, and phi for the number of subintervals $n = 10000$ only.

Table 2 Monte Carlo integration (MCI) along with time complexity using random generators *rand* and *Halton*, random sequence sources pi and phi for the integrals $I_3 = \int_{10^{-10}}^{10000} (\frac{\sin x}{x})^2 dx \approx \frac{\pi}{2}$,

$$I_4 = \int_0^{10000} \frac{1}{1+x^2} dx \approx \frac{\pi}{2},\ I_5 = \int_{10^{-10}}^{1} x \sin\frac{1}{x} dx \approx 0.3785283345,\ I_6 = \int_0^{10000} \frac{\cos 3x}{x^2+4} dx \approx \frac{\pi}{4e^6},$$

$$I_7 = \int_0^{10000} \frac{x \sin x}{1+x^2} dx \approx \frac{\pi}{2e},\ I_8 = \int_{10^{-10}}^{1} \ln x dx \approx -1,\ I_9 = g(0.5), I_{10} = g(1),\ I_{11} = g(2)$$

Integral with True Value	*RSS*	*n*	*MCI value*	*Relative Error*	*Time(sec)*
I_3 =1.5707963267949	*rand*	*10000*	*1.5659602830*	*0.0030787211*	*13.704*
	halton	*10000*	*1.5709818944*	*-0.0001181360*	*14.141*
	pi	*10000*	*1.5703525786*	*0.0002824989*	*13.954*
	phi	*10000*	*1.5722664771*	*-0.0009359268*	*13.875*
I_4=1.5707963267949	*rand*	*10000*	*1.5763960089*	*-0.0035648684*	*4.797*
	halton	*10000*	*1.5707451016*	*0.0000326110*	*4.719*
	pi	*10000*	*1.5701990990*	*0.0003802070*	*4.828*
	phi	*10000*	*1.5721051426*	*-0.0008332180*	*4.562*
I_5=0.3785283345	*rand*	*10000*	*0.3783614909*	*0.0004407690*	*10.968*
(true value is	*halton*	*10000*	*0.3783617259*	*0.0004401483*	*11.078*
unknown; it is found	*pi*	*10000*	*0.3783617763*	*0.0004400152*	*10.845*
with n=1 million)	*phi*	*10000*	*0.3783616031*	*0.0004404728*	*11.813*
I_6=0.00389361481	*rand*	*10000*	*0.0035024365*	*-0.7990667613*	*15.953*
(sensitive)	*halton*	*10000*	*0.0019306096*	*0.0083202139*	*15.935*
	pi	*10000*	*0.0019617511*	*-0.0076760232*	*15.888*
	phi	*10000*	*0.0021763239*	*-0.1178937972*	*15.827*
I_7= 0.577863674895461	*rand*	*10000*	*0.5777126672*	*0.0002613207*	*14.046*
	halton	*10000*	*0.5778474094*	*0.0000281477*	*14.187*
	pi	*10000*	*0.5779883744*	*-0.0002157940*	*13.812*
	phi	*10000*	*0.5780012634*	*-0.0002380986*	*13.687*
I_8= -0.999999997597415	*rand*	*10000*	*-0.9999905104*	*0.0000094872*	*12.798*
	halton	*10000*	*-0.9999998752*	*0.0000001224*	*12.829*
	pi	*10000*	*-0.9999996467*	*0.0000003509*	*12.782*
	phi	*10000*	*-1.0000019957*	*-0.0000019981*	*13.016*
I_9 = 1.5707963267949	*rand*	*10000*	*1.5725361655*	*-0.0011076157*	*21.954*
	halton	*10000*	*1.5707490321*	*0.0000301087*	*21.938*
	pi	*10000*	*1.5704613149*	*0.0002132752*	*22.078*
	phi	*10000*	*1.5723761285*	*-0.0010057330*	*21.734*
I_{10} = 0.785398163397448	*rand*	*10000*	*0.7897715914*	*-0.0055684215*	*21.969*
	halton	*10000*	*0.7856054911*	*-0.0002639778*	*21.563*
	pi	*10000*	*0.7850052219*	*0.0005003087*	*21.938*
	phi	*10000*	*0.7869130037*	*-0.0019287546*	*21.672*
I_{11} =0	*rand*	*10000*	*0.0002371015*	*Inf.(since true=0)*	*22.625*
	halton	*10000*	*-0.0000065119*	*Inf*	*22.485*
	pi	*10000*	*-0.0005259715*	*Inf*	*23.016*
	phi	*10000*	*0.0013499859*	*Inf*	*22.563*

We now extract the relative errors and represent these errors in Table 3 for all the eleven typical integrals for the sake of ranking the pseudo-random Matlab generator *rand*, the random sequence sources *pi* and *phi*, and the quasi-random generator *halton* in a statistical sense.

Table 3 (an extraction of Tables 1 and 2) Ranking of pseudo-random generator rand, random sequence sources phi and pi, and quasi-random generator halton in terms of relative error for the integrals considered here.

Integral	rand rel. error	phi rel. error	pi rel. error	halton rel. error	Best
I_1	0.0006015569	0.0006003443	0.0005997687	0.0005998961	pi
I_2	-0.0018511492	-0.0009693688	0.0002508333	-0.000262670	pi
I_3	0.0030787211	-0.0009359268	0.0002824989	-0.0001181360	halton
I_4	-0.0035648684	-0.0008332180	0.0003802070	0.0000326110	halton
I_5	0.0004407690	0.0004404728	0.0004400152	0.0004401483	pi
I_6	-0.7990667613	-0.1178937972	-0.0076760232	0.0083202139	pi
I_7	0.0002613207	-0.0002380986	-0.0002157940	0.0000281477	halton
I_8	0.0000094872	-0.000001998	0.0000003509	0.0000001224	halton
I_9	-0.0011076157	-0.0010057330	0.0002132752	0.0000301087	halton
I_{10}	-0.0055684215	-0.0019287546	0.0005003087	-0.0002639778	halton
I_{11}	infinity	infinity	infinity	infinity	halton

Since the true value in integral I_{11} is zero (Table 2), the concerned relative error defined as [(true value – computed value)/true value] will be evidently infinity as shown in both Tables 2 and 3. However, in the absolute sense, the best result was produced by the halton random generator (see Table 2).

From the foregoing result (Table 3), it is clear that the performance of pi as an RSS is consistently always better than phi (the golden ratio φ) without any exception. The transcendental number pi (π) is even comparable to the quasi-random generator halton in terms of uniformity. A quasi-random generator is known to produce more uniformly distributed random numbers (with less discrepancy) than those produced by a pseudo-random generator such as the generator rand. As a matter of fact, the digits of pi are more uniformly distributed than those of phi as depicted by the test statistic χ^2, viz., the value of tsc (Table 4) and are nearly as uniform as those produced by *halton*. When we consider larger number (more than 10000) of digits, the test statistic χ^2 (tsc) is expected to be better (smaller) for π than that for φ. Of course, digits fewer than 7500 (say, 2500, 5000) may not be sufficient to compare uniformity of digits of these two numbers φ and π.

In the ranking for the purpose of uniform (at least) one dimensional scanning and producing good accuracy, we have halton (rank 1), pi (rank 2), phi (rank 3), and rand (rank 4). It can be observed that pi has performed significantly better than both phi and rand in all the foregoing typical examples; it has even performed better than halton (and hence others) in about 40% of the problems considered here.

Visualization of the integrands The integrands of some of the integrals considered here are interesting and can be visualized so as to appreciate the numerical integration using the Monte Carlo methods. We present the graph of these integrands along with the respective Matlab commands.

Integral I_2**:** Visualization of the integrand of I_2, viz., $\int_0^\infty \frac{\sin x}{x} dx = \frac{\pi}{2} =$ 1.5707963267949 is achieved through the Matlab command >>fplot('sin(x)/x', [10^-10, 100]) as in Fig. 2.

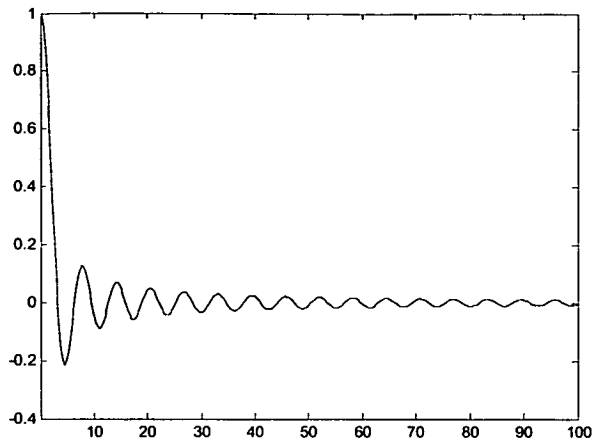

Fig. 2 Graph of $\frac{\sin x}{x}$, *Limits* : $[10^{-10},100]$

Integral I_4 Visualization of the integrand of I_4, viz., (a) $\int_0^\infty \frac{1}{1+x^2} dx = \frac{\pi}{2}$; True value= 1.5707963267949 is performed by the Matlab command >>fplot('1/(1+x^2)', [0,100]) as in Fig. 3.

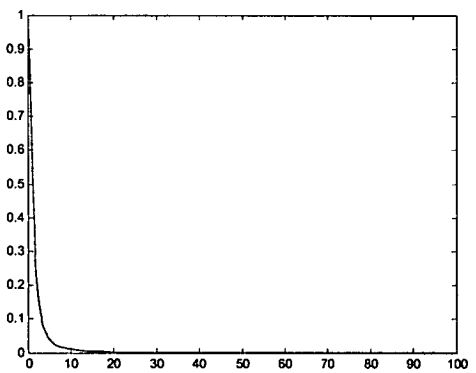

Fig. 3 Graph of $\frac{1}{1+x^2}$, *Limits* : [0,100]

Integral I_4 Visualization of the integrand of I_4, viz., (b) $\int_0^\infty \frac{1}{1+x^2} dx = \frac{\pi}{2}$ near origin is done using the command >>fplot('1/(1+x^2)', [0,5]) as in Fig. 4.

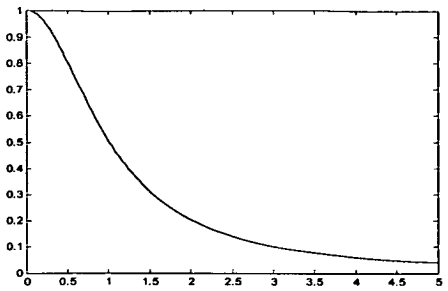

Fig. 4 Graph of $\frac{1}{1+x^2}$, *Limits* : [0,5]

Integrand of I_5 The graph of the integrand (a) $x\sin(\frac{1}{x})$, *Limits* : [0,1] is produced by the command >>fplot('x*sin(1/x)', [0,1]) as in Fig. 5.

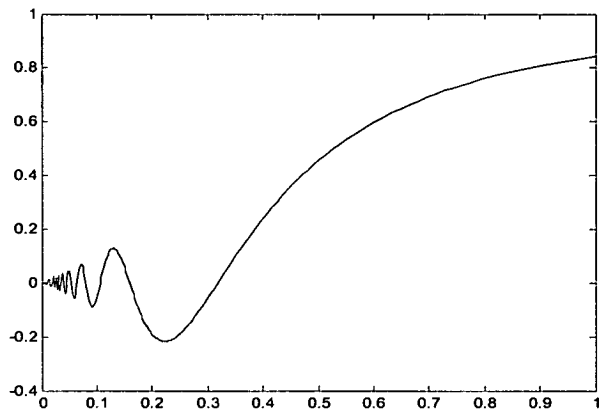

Fig. 5 Graph of $x\sin(\frac{1}{x})$, *Limits* : [0,1]

Integrand of I_5 The integrand (b) $x\sin(\frac{1}{x})$, *Limits* : [0,0.1], when expanded near the origin using the command >> fplot('x*sin(1/x)', [10^-10,.1]) outputs the graph as in Fig. 6.

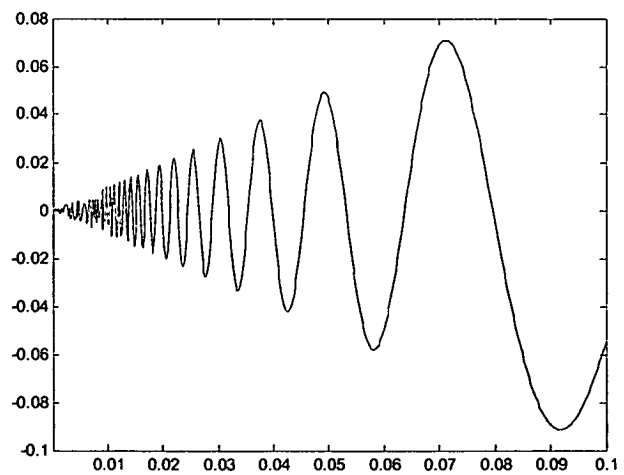

Fig. 6 Graph of $x\sin(\frac{1}{x})$, *Limits* : [0,0.1]

Integrand of I_6 The integrand (a) $\frac{\cos 3x}{x^2+4}$, *Limits* : $[-\infty,\infty]$; (Integration value = $\frac{\pi}{2e^6}$; Even integrand; True numerical value= 0.00389361481414237) has the graphical representation as in Fig. 7 produced by the command fplot('cos(3*x)/(x^2+4)', [-100,100]) is

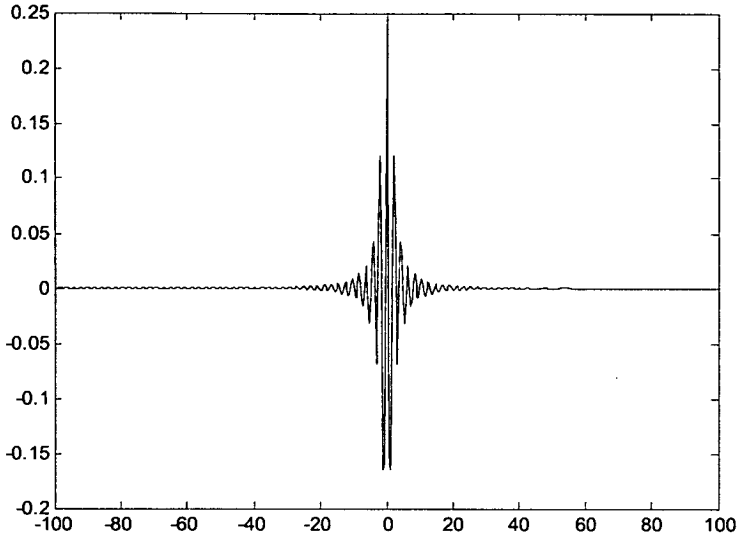

Fig. 7 Graph of $\frac{\cos 3x}{x^2+4}$, *Limits* : $[-100,100]$

Integrand of I_6 The graph of the integrand (b) $\frac{\cos 3x}{x^2+4}$, *Limits* : $[-\infty,\infty]$ expanded near the origin and produced by the command >>fplot('cos(3*x)/(x^2+4)', [-10,10]) is shown in Fig. 8.

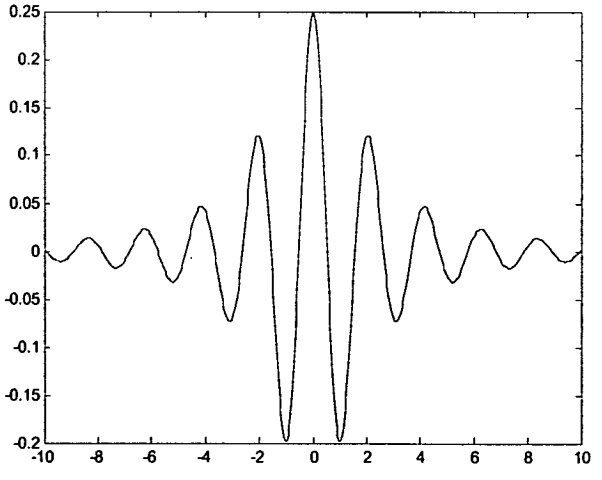

Fig. 8 Graph of $\frac{\cos 3x}{x^2+4}$, *Limits* : $[-10,10]$

Integral I_7 The integrand of the integral I_7, viz., (a) $\int_{-\infty}^{\infty} \frac{x\sin x}{1+x^2}dx = \frac{\pi}{e}$, (Even integrand; True value= 1.15572734979092) is visualized as in Fig. 9 using the command >>fplot('x*sin(1/x)/(1+x^2)', [-100,100])

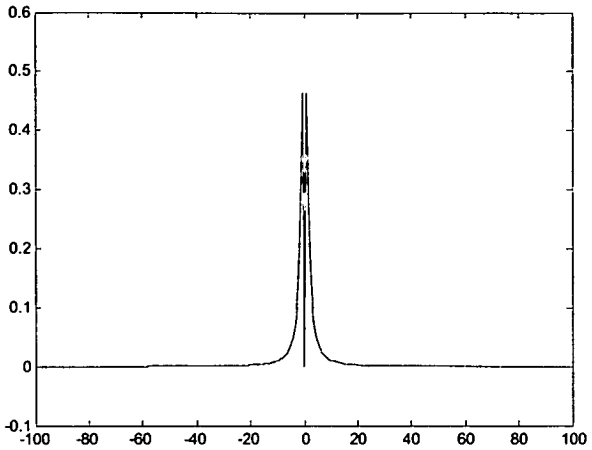

Fig. 9 Graph of $\frac{x\sin x}{1+x^2}$, *Limits* : [−100,100]

Integral I_7 The integrand of the integral I_7, i.e., (b) $\int_{-\infty}^{\infty} \frac{x\sin x}{1+x^2}dx = \frac{\pi}{e}$, (Even integrand) in expanded form near the origin when plotted by the command >>fplot('x*sin(1/x)/(1+x^2)', [-0.1,0.1]) would appear as in Fig. 10.

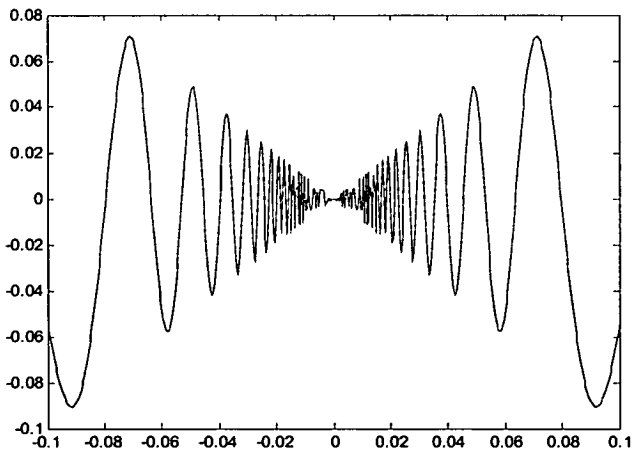

Fig. 10 Graph of $\frac{x\sin x}{1+x^2}$, *Limits* : [−0.1,0.1]

Integral I_8 The integrand of I_8, viz., $\int_{0.01}^{1} \ln x \, dx = x \ln x - x = \ln 1 - 1 - 0.01 \ln 0.01 + 0.01 =$ –0.943948298140119 has the graph produced by the command >>fplot('log(x)', [0.01, 1]) as in Fig. 11.

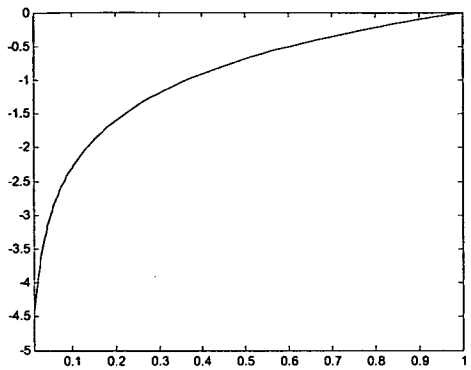

Fig. 11 Graph of $\ln x$, *Limits* : [0.01,1]

Integrals I_9, I_{10}, and I_{11} The integrands of the integrals $g(\omega) = \int_0^{\infty} \frac{\cos \omega x \sin x}{x} dx = \begin{cases} \pi/2, 0 < \omega < 1 \\ \pi/4, \omega = 1 \\ 0, w > 1 \end{cases}$

Have the graphs shown in Figs. 12, 13, 14. The command >>fplot('cos(.5*x)*sin(x)/x', [10^-10,100]) for $\omega = 0.5$ for I_9 depicts the graph as in Fig. 12.

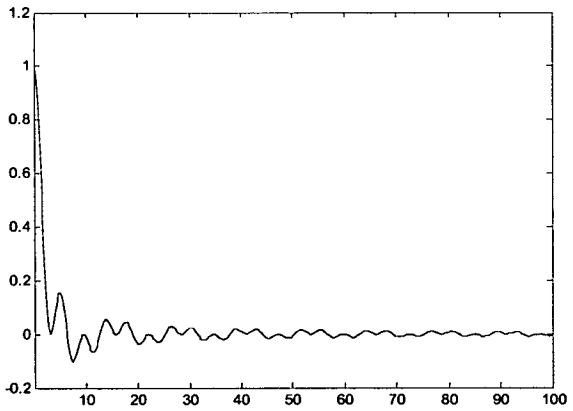

Fig. 12 Graph of $\frac{\cos 0.5x \sin x}{x}$, *Limits* : [$10^{-10}$,100]

The command >>fplot('cos(x)*sin(x)/x', [10^-10,100]) for $\omega = 1$ for I_{10} produces the graph as in Fig. 13.

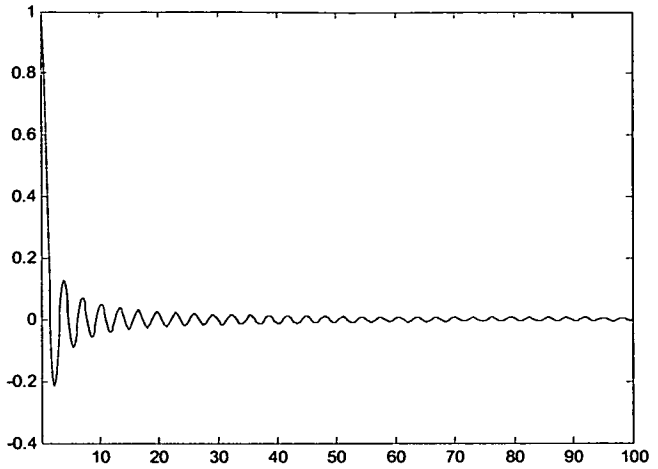

Fig. 13 Graph of $\frac{\cos x \sin x}{x}$, *Limits* : $[10^{-10},100]$

The command >>fplot('2*cos(x)*sin(x)/x', [10^-10,100]) for $\omega = 2$ for I_{11} generates the graph shown in Fig. 14.

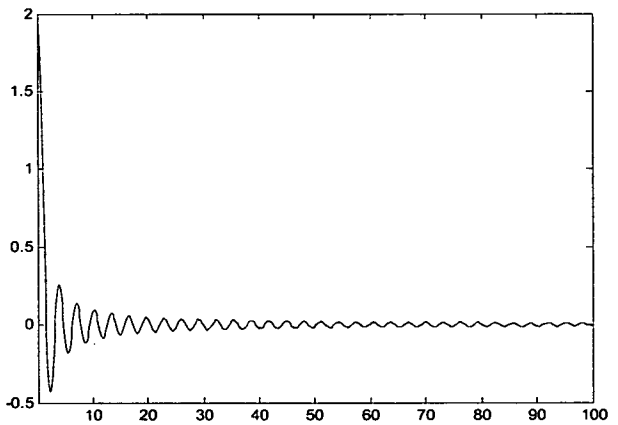

Fig. 14 Graph of $\frac{\cos 2x \sin x}{x}$, *Limits* : $[10^{-10},100]$

Table 4 Distribution of digits of φ and π with test statistic χ^2 (tsc), critical value of χ^2 (cvc= 14.6840) at 10% significance level and expected frequency (ef) of each digit

	φ (number of	leftmost digits)	π (number of	leftmost digits)
	7500	10000	7500	10000
0	758	1020	701	968
1	795	1062	782	1026
2	775	994	741	1021
3	784	1039	744	975
4	736	976	761	1013
5	741	988	784	1046
6	672	918	762	1021
7	760	1025	745	969
8	733	987	719	947
9	746	991	761	1014
	tsc=14.1813	tsc=14.12	tsc=8.0933	tsc=9.4580
	ef=750	ef=1000	ef=750	ef=1000

4. Conclusions

Study of transcendental and algebraic numbers for random sequence sources for randomized algorithms Our numerical experiment shows that pi (π) has always scored over phi (the golden ratio) in one dimensional Monte Carlo integration. This implies that other transcendental numbers as well as algebraic numbers could also be investigated to determine the uniformity/discrepancy of distribution of random numbers generated out of them. This has not only academic interest but also possibly significant real world applications. However, while quasi-random sequences are better suited for uniform scanning of a space such as that required in Monte Carlo integration, these need not fare better than the pseudo-random sequences for other kinds of problems such as the traveling salesman problem (TSP) [2]. These are because pseudo-random numbers are a bit too random unlike quasi-random numbers. Quasi-random numbers, on the other hand, are more uniformly distributed (with less discrepancy) than pseudo-random numbers.

Ready generation of random numbers versus storage-and-retrieval of random numbers Our time complexity experiment in Monte Carlo integration demonstrates that unless we have a more efficient storage and retrieval system, there is no appreciable advantage of storage and retrieval over the instant generation of random numbers. Although storing all the random numbers before the execution of an algorithm needs considerable storage space, it may be considered as not an important issue since (i) we have sufficient disk space available to us now and (ii) the problem space may be appropriately divided into subspaces so that we can always use fixed number (rather small set of) of random numbers for each subspace. However, we have experienced significant improvement in time complexity by pre-storage and retrieval of random numbers over certain random number generators while solving the TSP.

Use of Pi and Phi in Multiple integration Does pi fare better than phi for double, triple, and higher dimensional integrations? Although it might appear that pi should fare better than phi in higher dimensional integration as it has done for single integration, we are yet to explore this aspect. This

involves significant research mainly on the method of using the random sequence as well as the technique of implementation. A detailed study on this will appear elsewhere.

Connection of pi with Monte Carlo method in history Buffon posed and solved the following problem in 1777. Suppose a needle of length L is thrown randomly onto a horizontal plane marked with equi-spaced parallel straight lines having a distance $d > L$ between any two consecutive lines. If the distance of the center of the needle from the nearest line is x and the angle that its orientation makes with the line (or for that matter with any other line) is α, then the needle will intersect a line if and only if $x < 0.5L\sin\alpha$. What is the probability $P(x < 0.5L\sin\alpha)$ that the needle will intersect a line [39]? Buffon derived the probability of intersection as

$$P = L\int_0^{\pi}\sin\alpha d\alpha)/(\pi d) = \frac{2L}{\pi d} \Rightarrow \pi = \frac{2L}{dP}.$$

This was an entirely new method of computing π in 1777. Here to calculate the probability P, the needle needs to be thrown onto the ruled plane for a large number (H) of times and the number (h) of times it intersects a line. The probability $P = h/H$ is then computed.. However, during those days when today's modern computer was not even possibly imagined, determining π even to two decimal places in this way was not at all attractive; it had definitely an academic interest though. Today with the advent of super-high speed computers (over one billion floating point operations per second), such a calculation of probability to an acceptable accuracy is not unrealistic. Similar historical relation between the golden ratio and the Monte Carlo method is not known to us.

Deterministic mathematicallydirec methods versus Monte Carlo methods for integration Can the deterministic methods such as the closed quadrature formula Trapezoidal rule/Simpson's 1/3 rule and the Gauss-Legendre open quadrature always excel the Monte Carlo methods? The answer is usually 'yes' from both accuracy and computational complexity point of view. The MC procedure is sometimes preferred over a numerical quadrature when the integrand is violently fluctuating and/or involves long trigonometric/special functions. However, our focus is on the comparison of pi and phi — which one is a better source for a random sequence for MC integration/for a uniform scanning of given a domain.

Complexity Issues For most real world applications where the mathematical model consists of a single integral, computational/time complexity is often not an important issue since the time of computation for four significant digit (0.005%) accuracy is often of the order of seconds. It may be noted that the present day (2007) computer can execute over a billion floating operations per second and both Monte Carlo and quadrature methods are polynomial-time. From Tables 2 and 3, it appears that there are no significant differences in time complexities among the Monte Carlo integrations using any of the four RSS's, viz., *rand, halton, pi,*and *phi*. However, for multiple integration, complexity could become an issue. We will explore this in our future work.

Parallel implementation For the MC integration where we divide the region of integration into sub-regions, parallel implementation is straightforward since integration for each sub-region can be carried out independently and parallely/simultaneously using available processors in a multiprocessor system. The outputs are then added in a parallel mode to get the required integration value.

Appendix

The following Matlab program *mc1integrationrhpiphi* stored in the file named "mc1integrationrhpiphi" is the Monte Carlo one dimensional integration for all the integrals presented in section 3. This program uses another Matlab program *halton* stored in a separate file. It computes the value of an integral for varied number of subintervals with a fixed number N (=5000) of random numbers used in each subintervals. It includes the four random number sources, viz., rand, halton, pi, and phi and is self-explanatory. Observe that there are four appropriate lines of code from which the percentage symbol "%" need to be removed for the execution of the program. To conserve space we reproduce, in the Matlab program *mc1integrationrhpiphi,* π denoted by pi0 as well as φ denoted phi partially. We have removed the decimal point immediately after the first digits 3 and 1 in pi0 and phi as this is of no concern to us in this random sequence source (SSR) context. However, in the actual Matlab program pi0 and phi must be completely included. We have demonstrated later how all the 50000 digits used here for each of pi0 and phi could be generated using the Matlab command *vpa* and the Matlab program *insertblanks*.

```
%mclintegrationrhpiphi
function [true, I, comp_accuracy]=mclintegrationrhpiphi(fun);
%50000 digits of pi is in the following location pi0; each row has 10 elements.
%Middle two elements are not included so as not to exceed the line length.
pi0=[3141592653 5897932384 6264338327 9502884197 . . . 4592307816 4062862089 9862803482 5342117067
9821480865 1328230664 7093844609 5505822317 . . . 2841027019 3852110555 9644622948 9549303819
6442881097 5665933446 1284756482 3378678316 . . . 2346034861 0454326648 2133936072 6024914127
3724587006 6063155881 7488152092 0962829254 . . . 0011330530 5488204665 2138414695 1941511609
 .
 .
 .

8788053304 2714630119 4158989632 8792678327 . . . 0466587100 5000832851 7731177648 9735230926
6612345888 7310288351 5626446023 6719966445 . . . 1511493409 3934475007 3025855814 7561908813
9875235781 2331342279 8665035227 2536717123 . . . 6007956982 7626392344 1071465848 9578024140
8158405229 5369374997 1066559489 4459246286 . . . 5339439142 1112718106 9105229002 4657423604];

phi=[1618033988 7498948482 0458683436 5638117720 . . . 8622705260 4628189024 4970720720 4189391137
4847540880 7538689175 2126633862 2235369317 . . . 3890865959 3958290563 8322661319 9282902678
8067520876 6892501711 6962070322 2104321626 . . . 4975870122 0340805887 9544547492 4618569536
4864449241 0443207713 4494704956 5846788509 . . . 6478091588 4607499887 1240076521 7057517978
.
.
.
9670572589 3974080648 3175935357 8603833221 . . . 6229114328 2590723291 9935637853 2927911544
0043613420 1605741421 0103382425 7773560259 . . . 2438503757 7755389867 5571298529 3795171358
9130860163 5581160937 2610924981 1785171527 . . . 9680890339 9517122420 3675938612 4862023009
5411334656 3608193864 3097371691 1136979496 . . . 4373853556 2868657173 5845963512 3774246170];

t0=clock;
%*************************************************
%FIRST CHANGE: Supply values for alpha, beta, n
%alpha and beta are lower and upper limits of integration.
%alpha=0.000005;beta=.01; n=1000; a(1)=alpha;
alpha=10^-10; beta=10000;n=10000; a(1)=alpha; %Interval (beta-alpha) is divided into n subintervals
%alpha=10^-10; beta=1; n=10000; a(1)=alpha; %Interval (beta-alpha) is divided into n subintervals.
%alpha=0; beta=1;n=10000; a(1)=alpha; %Interval (beta-alpha) is divided into
%n subintervals. This is for integral x^2 from 0 to 1.
%alpha=0.000005;beta=1; n=100000; a(1)=alpha; %Interval (beta-alpha) is divided into n subintervals.
%alpha=0;beta=10000; n=10000; a(1)=alpha; %Interval (beta-alpha) is divided into n subintervals.
%alpha=0; beta=200; a(1)=alpha; n=10000; %Interval (beta-alpha) is divided into n subintervals.
%*************************************************
for i=2:n-1, a(i)=a(i-1)+(beta-alpha)/n; end;
%*************************************************
N=5000;S1=0;
%*************************************************
%SECOND CHANGE: Remove '%" from one of rand, halton, pi0, and phi
%x=rand(1, 5000); fprintf('Matlab rand used as random number source\n') %Matlab rand
%-------------------------
%x=halton(1,N); fprintf('halton used as random number source\n') %halton
%-------------------------
%reshape(pi0,1,5000);x=10^-10*pi0;;fprintf('pi used as random number source\n') %pi
%-------------------------
```

```
reshape(phi,1,5000);x=10^-10*phi; fprintf('phi used as random number source\n') %phi
%************************************************
fprintf('   # of subintervals n  # of random nos. N  TrueValue    IntegrationValue     RelativeError
Time_in_sec\n')

for i=2:n-1, S=0;for j=1:N,
t=x(j);
%*****************************************************************************
p=a(i)-a(i-1);q=a(i-1); x1=p*t+q;
%THIRD CHANGE: Write fun = p*(integrand in terms of x1)
%fun=p*x1^2; %Integrand is x^2
%fun=p*sin(x1)/x1; %Integrand is sin x/x
%fun=p*(sin(x1)/x1)^2; %Integrand is (sin x /x)^2
%fun=p*(1/(1+x1^2)); %Integrand is 1/(1+x^2)
%fun=p*x1*sin(1/x1); % Integrand is x sin(1/x)
%fun=p*cos(3*x1)/(x1^2+4); %Integrand is cos 3x /(x^2+4)
%fun=p*x1*sin(x1)/(1+x1^2); %Integrand is x sin x/(1+x^2)
%fun=p*log(x1); %Integrand is ln x
%fun=p*cos(0.5*x1)*sin(x1)/x1; %Integrand is (cos 0.5x sin x)/x
%fun=p*cos(x1)*sin(x1)/x1; %Integrand is (cos x sin x)/x
fun=p*cos(2*x1)*sin(x1)/x1; %Integrand is (cos 2x sin x)/x
%*****************************************************************************
S=S+fun;
%S=S+(a(i)-a(i-1))*((a(i)-a(i-1))*t+a(i-1))* sin(1/((a(i)-a(i-1))*t+a(i-1)));
end;
S=S/N; S1=S1+S;I=S1;
end;
%*****************************************************************************
%FOURTH CHANGE: Write true (True integration value)
%true = 1/3; %True value of integral x^2 from 0 to 1
%true= 0.3785283345; % Probably true value of integral x sin (1/x) from 10^-10 to 1
%true=pi/2; %True value of integral 1/(1+x^2)or sin x/x or (sin x/x)^2 from 0 to inf.
%true=-.943948298140119; %True value of integral ln x from .01 to 1
%true=pi/(4*exp(6)); %True value of integral cos 3x/(x^2+4) from 0 to inf.
%true=pi/(2*exp(1)); %True value of integral x sin x/(1+x^2) from 0 to inf.
%true=1*log(1)-1-10^-10*log(10^-10)+10^-10;%True value of integral ln x from 10^-10 to 1.
%true=pi/2; %True value of the integral cos(0.5x)sin x/x from 10^-10 to inf.
%true=pi/4; %True value of integral cos x sin x / x from 0 to inf.
true=0; %True value of integral cos 2x sin x / x from 0 to inf.
%-----------------
comp_accuracy =(true - I)/true; format long g;
fprintf('%16i %20i %17.10f %17.10f %17.10f %13.6f \n', i+1, N, true, I, comp_accuracy, etime(clock,
t0));
```

The Matlab program "halton" presented below needs to be saved in the Matlab file called halton — a file different from the file "mc1integrationrhpiphi".

```
function r = halton(p,q) % where p = 1 and q = n = # of RNs
 dim = 2; persistent seed seed2 base
if isempty(seed2),prm_numbers=primes(300);base=prm_numbers(dim);seed2=base^4-1;
 end
 if nargin < 1, p = 1; end, if nargin < 2, q = 1; end
 r = zeros(p,q); seed = seed2;
 for k = 1:p*q, x = 0.0; base_inv = 1.0/base;
 while ( any ( seed ~= 0 ) )
 digit = mod ( seed, base ); x = x + digit * base_inv;
 base_inv = base_inv / base; seed = floor ( seed / base );
 end
 r(k) = x; seed2 = seed2 + 1; seed = seed2;
 end
```

Inserting blanks in a string of blankless digits/characters We have used the Matlab program *insertblanks* to convert a .txt file containing consecutive characters/digits without any blank anywhere. Consider, for instance, the first 50000 digits of pi or those of phi produced by the Matlab variable precision arithmetic command vpa described later in this appendix. The output, viz., the string of 50000 digits will be obtained/shown on the Matlab command window in a single line without any blank in the string. The insertblanks program takes this string in the form of a .txt file as its input and then outputs the string in a matrix with desired number of columns and with desired blanks after each fixed-sized block of, say, 10 digits. Insertion of such blanks is necessary to make

use of the consecutive blocks as random numbers. The program which is self-explanatory is as follows.

```
%Inserting blanks in .txt file: Blanks are inserted among fixed blocks of (alphanumeric) characters.
function f = insertblanks()
clc;
% show .txt files
system('dir *.txt');
% system('dir *.asv'); % system('dir *.*');

disp(sprintf('\n'));
inputfile=input('Enter input file name: ','s'); %Input file has extension .txt
disp(sprintf('\n\n'));
outputfile=input('Enter output file name: ','s');
disp(sprintf('\n'));
columns=input('Enter no. of columns for display matrix: ');
disp(sprintf('\n\n\n'));
elementsize=input('Enter no. of digits in each element: ');

fid1 = fopen(inputfile);
fid2 = fopen(outputfile,'w');
  St = fscanf(fid1,'%s'); loopCount =length(St)/ elementsize;
  S=length(St); initial=1; final=elementsize; Columns = 0; Line = '';

  for i=1:elementsize:S
     Temp=St(1,initial:final); initial = initial+elementsize; final = final+elementsize;
     if final > S % last number is not K-digits
 final = S;
end

     Columns = Columns +1; Line = strcat(Line, sprintf(' %s', Temp));

     if Columns == columns
        disp(sprintf('%s\n',Line)); fprintf(fid2,'%s\n',Line);
Columns = 0; % Reset column counter
        Line='';
     end
  end

  disp(sprintf('%s\n',Line)); % Display left-over
  fprintf(fid2,'%s\n',Line);  % Store left-over in file

fclose(fid1);
fclose(fid2);

disp(sprintf('\nInput File Name: %s',inputfile));
disp(sprintf('\nOutput File Name: %s',outputfile));

disp(sprintf('\nNumber of coulumns of display matrix: %d',columns));
disp(sprintf('\nNumber of digits in an element: %d', elementsize));
```

To generate, say, 50000 digits of golden ratio as well as those of pi, one may use the Matlab variable precision arithmetic command *vpa* as follows.

>>vpa((sqrt(5)+1)/2,50000), >> vpa(2*asin(1), 50000)

These will produce 50000 digits each of golden ratio and pi in the Matlab command window. The 50000 digits will be displayed in one line without any blank anywhere. The blanks are then required to be appropriately inserted between fixed blocks of digits. Inserting blanks is achieved by the Matlab program *insertblanks*. Each block will be used as a random number. It is often mapped onto an interval, say, [0, 1] in most applications.

Creating digits and inserting blanks If we, for instance, want to insert a blank after each block of 10 digits of 50000-digit π, then the procedure could be written as

S.1 Execute in Matlab command window the command >>pi50000=vpa('2*asin(1)', 50000)
S.2 Copy from the Matlab command window the 50000 digits by clicking "Edit" and then "Copy".

S. 3 Open a new M-file and paste these 50000 digits. Remove the decimal point '.'
S. 4 Click on "File" situated at the left top corner. Go to "Save as", click and save in pi50000.txt.
S.5 Now go to Matlab command window and execute the command >>insertblanks. At the following four prompts provide the requisite file names, number of columns in each line, and number of digit in each element of a column. If we want 10 columns in each line and 10 digits in each element in a column, and the output file name as pi50000outcol10block10, then the required inputs will be provided as follows.

Prompt	Input
Enter input file name:	pi50000.txt
Enter output file name:	pi50000outcol10block10
Enter number of columns for display matrix:	10
Enter number of digits in each element:	10

Thus we obtain the required output which must be plugged in the program mc1integrationrhpiphi before we use pi as an RSS.

References

[1] S.K. Sen and R.P. Agarwal, Golden ratio in science, as random sequence source, its computation, and beyond, to appear in *Computers and Mathematics with Applications*.
[2] T. Samanta and S.K. Sen, Pseudo- versus Quasi-random Generators in Heuristics for Traveling Salesman Problem, to appear.
[3] T. Samanta, Random Number Generators: MC Integration and TSP-solving using Simulated Annealing, Genetic and Ant System Approaches, Ph.D. Thesis, Department of Mathematical Sciences, Florida Institute of Technology (2006).
[4] H. J. Karloff and P. Raghavan, Randomized algorithms and pseudorandom numbers, *Journal of the ACM*, 40, No.3, 454-476, (1993).18.
[5] M. M. Meysenburg and J.A. Foster, The quality of pseudorandom number generators and simple genetic algorithm performance, In Proceedings of the Seventh International Conference on Genetic Algorithms, Morgan Kaufmann, 276-281, (1997).
[6] M. M. Meysenburg and J.A. Foster, Randomness and ga performance, revisited, Proceedings of the Genetic and Evolutionary Computation Conference. Morgan Kaufmann, (1999).
[7] E. Cantu-Paz, On Random Numbers and the Performance of Genetic Algorithms, In Proc. of Genetic and Evolutionary Computation Conference (GECCO) 2002, 754-761, (2002).
[8] S. Kimura and K. Matsumura, Genetic Algorithms using Low Discrepancy Sequences, Proceedings of the 2005 conference on Genetic and evolutionary computation, Washington DC, USA, 1341-1346, (2005).
[9] The TSP Symmetric Traveling Salesman Problem Instances, http://elib.zib.de/pub/Packages/mp-testdata/tsp/tsplib/tsp/
[10] S. Galanti, and A. Jung, Low-Discrepancy Sequences: Monte Carlo Simulation of Option Prices, The Journal of Derivatives, 5, 63-83, (1997).
[11] S.K. Sen, T. Samanta and A. Reese, Quasi- Versus Pseudo-Random Generators: Discrepancy, Complexity and Integration-Error based Comparison, International Journal of Innovative Computing, Information and Control, 2, No. 3, (2006).
[12] D. E. Knuth, The Art of Computer Programming; Volume 2: Seminumerical Algorithms; Addison-Wesley; Reading, MA, 2nd Edition, (1981).
[13] P. L'Ecuyer, Software for uniform random number generation: distinguishing the good

and the bad, Winter Simulation Conference, Proceedings of the 33nd conference on Winter simulation, Arlington, Virginia, 95 – 105, (2001).
[14] V. Lakshmikantham, S.K. Sen, and T. Samanta, Comparing Random Number Generators Using Monte Carlo Integration, International Journal of Innovative Computing, Information and Control, 1, No. 2, 143-165, (2005).
[15] M.Matsumoto and T. Nishimura, Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator, ACM Trans. on Modeling and Computer Simulations, 8, No. 1, 3-30, (1998).
[16] J. H. Halton, and G. B. Smith, Algorithm 247: Radical-Inverse Quasi-Random Point Sequence, Communications of the ACM, 7, 701-702, (1964).
[17] I. M. Sobol, On the Distribution of Points in a Cube and the Approximate Evaluation of Integrals, U.S.S.R. Computational Mathematics and Mathematical Physics, 7, 86-112, (1967).
[18] H. Faure, Discrepance de suites associees a un systeme de numeration (en dimension s), Acta Arithmetica XLZ, 337-351, (1982).
[19] H. Niederreiter, Low Discrepancy and Low-Dispersion Sequences, Journal of Number Theory, 30, 51-70, (1988).
[20] J. G. Van der Corput, Verteilungsfunktionen I, II, Nederl. Akad. Wetensch. Proc. Ser. B, 38, 813-821 and 1058-1066, (1935).
[21] M. Junger, G. Reinelt, and G. Rinaldi, The Traveling Salesman Problem, Operations Research and Management Sciences, 7, 225-330, 1995.
19
[22] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, The Traveling Salesman Problem-A Guided Tour of Combinatorial Optimization, JohnWiley and Sons, (1985).
[23] G. Reinelt, The Travelling Salesman: Computational Solutions for TSP Applications, Springer-Verlag, (1994).
[24] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, Equation of State Calculations by Fast Computing Machines, J. Chem. Phys., 21, 6, 1087-1092, (1953).
[25] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, Optimization by Simulated Annealing, Science, 220, 4598, 671-680, (1983).
[26] V. Cerny, A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm, Journal of Optimization Theory and Applications, 45, 41-51, (1985).
[27] C. C. Skicism and B. L. Golden, Solving k-shortest and constrained shortest path problems efficiently, Ann. Operations Research, 20, 249-282, (1989).
[28] R. W. Eglese, Simulated Annealing: A Tool for Operational Research, European Journal of Operational Research, 46, 271-281, (1990).
[29] M. Fleischer, Simulated Annealing: Present, Past and Future, Proceedings of Winter Simulation Conference, (1995).
[30] P. J. M. van Laarhoven and E. H. L. Aarts, Simulated Annealing: Theory and Applications, Kluwer Academic Publishers: Norwell MA, USA, (1987).
[31] E. H. L. Aarts and J. Korst, Simulated Annealing and Boltzmann Machines : A Stochastic Approach to Combinatorial Optimization and Neural Computing, Wiley: Chichester [England], New York, (1989).
[32] E. H. L. Aarts and J. K. Lenstra, Local Search in Combinatorial Optimization, Wiley: Chichester [England] ; New York, (1997).
[33] M. M. Flood, The Traveling–Salesman Problem, Operations Research, 4, 61-75, (1956).
[34] A. Croes, A Method for Solving Traveling Salesman Problems, Operations Research, 5, 791-812, (1958).

[35] S. Lin and B. W. Kernighan, An effective heuristic algorithm for the traveling salesman problem, Operations Research, 21, 498-516, (1973).
[36] Z. Michalewisz and D. B. Fogel, How to Solve It: Modern Heuristics, Springer-Verlag, Berlin, Germany, (2000).
[37] E.V. Krishnamurthy and S.K. Sen, Numerical Algorithms: Computations in Science and Engineering, Affiliated East-West Press, New Delhi,(2001).
[38] V. Lakshmikantham and S.K. Sen, Computational Error and Complexity in Science and Engineering, Elsevier, Amsterdam, (2005).
[39] P. Beckmann, A History of π (Pi), Barnes and Noble Books, New York, (1971).