Monte Carlo Methods

The general class of computational methods based on various forms of random sampling were called Monte Carlo methods, in honor of the famous principality whose raison d'être is gambling[1]. The method is called "deterministic" if the computation being performed is not inherently random: numerical quadrature schemes, based on evaluating a function at random abscissas; or solution of the Laplace equation with complicated boundary conditions; are deterministic problems. Conversely, if the problem has inherent random elements—for example, determining the distribution of particles that strike a screen after emerging from a magnetic spectrometer; or estimating the variance of the value of a stock portfolio in the face of market fluctuations—we call it "probabilistic".

We have already discussed numerical quadrature by Monte Carlo methods, as well as function minimization by simulated annealing. Both of these are deterministic problems. In this chapter we shall focus on problems for which the labor of simulating the underlying physics is no greater—and in some instances, much less—than that needed for an analytic solution. Our examples will be fairly simple—students who wish to delve further into this subject are directed to the classic work by Hammersley and Handscomb[2].

1. Pseudo-random numbers

The best contemporary discussion of methods for generating pseuo-random numbers on a digital computer is Knuth[3]. The liveliness of the subject is indicated by voluminous recent correspondence in computing journals[4].

We begin with a brief discussion of random numbers. A random sequence is a set of integers $\{a_k\}$ in the range [M, N] that exhibit no correlation from one to the next—in the sense that given that a_k has a certain value, the next number in the sequence, a_{k+1} is more likely to have one value than another (over the range of possible values). The absence of serial correlation can be generalized to an absence of multiple serial correlation: that is, instead of testing a_{k+1} we test a_{k+r} where $r > 1$. Such random sequences might be generated by perfectly fair roulette wheels, for example.

Of course, the techniques for generating "random" numbers on a digital computer are actually deterministic in nature. Thus given the same initial value a_0 , a function for generating "random" numbers will produce exactly the same sequence of numbers

1. When Von Neumann and Metropolis invented the method, Las Vegas, Nevada had not yet become the gambling Mecca it is today.
2. J.M. Hammersley and D.C. Handscomb, *Monte Carlo Methods* (Methuen & Co., Ltd., London, 1964).
3. Donald Knuth, Seminumerical Algorithms: The Art of Computer Programming, v. 2 (Addison-Wesley Publishing Co., reading, MA, 1981).
4. See, *e.g.* the letter by G. Marsaglia, *Commun. ACM* **36**, 7 (July 1993) 105-108, and various responses.

$$a_{n+1} = R(a_n)$$

every time such a sequence is generated. Moreover, by their very nature, even the least correlated random number generating functions will exhibit some degree of serial correlation. This is why such functions are called *pseudo-random number generators* (`prng`'s) in the literature of computation.

Another point to be aware of is that on computers whose word-size is 32 bits, many `prng`'s in common use produce integers in the range $[1, 2^{31} - 1] \equiv [1, 2147483647]$ so that after about 2×10^9 calls to the function the sequence will repeat itself. This may affect the simulation adversely[5].

The standard algorithm for generating prn's is the *linear congruential method* first proposed by Lehmer[6]. This is based on

$$x_{k+1} = (a\,x_k + c) \mod M\ .$$

With proper choices of a, c and M this can be a very good generator. Poor choices will lead to short cycles and lots of correlation. One good choice is $a = 16807$, $c = 0$ and $M = 2^{31} - 1$. There are other choices that are as good or better, but somewhat harder to implement.

Here is a program in Forth that implements this algorithm and produces a uniformly distributed floating point number in the range (0, 1) :

```
\ Pseudo random number generator in ANS Forth
\
\       Leaves a pseudo random number in the range (0,1)
\           on fp stack.
\
\ -------------------------------------------------
\     (c) Copyright 1998  Julian V. Noble.          \
\       Permission is granted by the author to      \
\       use this software for any application pro-  \
\       vided this copyright notice is preserved.   \
\ -------------------------------------------------
\
\   Based on GGUBS algorithm:  s' = 16807*s mod (2^31-1)
\   P. Bratley, B.L. Fox and L.E. Schrage, A guide to simulation
\   (Springer, Berlin, 1983).
\
\   To simplify transport to 16-bit machines the 32-bit
\   modular division is performed by synthetic division:
\
\   Let  b = d * m1 + m2 (= 2^31 - 1)
\
\   so that ( [n] means "largest integer <= n" )
\
```

5. For example, a system-supplied `prng` I once used had a cycle length of only 2^{15}-1= 32767, which led to very unreliable results.
6. D.H. Lehmer, *Ann. Comp. Lab. Harvard Univ.* **26** (1951) 141-146; M. Greenberger, *Journal of the ACM* **8** (1961) 163-167.

```
\       s' = (s*m1) MOD b = s*m1 - [s*m1/b]*b
\                         = m1 * (s - [s/d]*d) - m2 * [s/d]
\
\   Environmental dependence:
\
\       1. assumes at least 32-bit DOUBLEs
\       2. needs FLOATING and DOUBLE wordsets
\       3. assumes separate floating point stack
\

\ MARKER -rand

2VARIABLE     seed

21474.83647   D>F   FCONSTANT  bigdiv            \ 2^31-1
1277.73       D>F   FCONSTANT  divis
16807.        D>F   FCONSTANT  m1
2836.         D>F   FCONSTANT  m2

: (rand)    ( adr --)   ( f: -- seed')
         DUP  2@  D>F  divis FOVER FOVER        ( f: s d s d)
         F/  F>D  2DUP   D>F        ( [s/d])    ( f: s d [s/d])
         F*   F-                    ( f: s-d*[s/d])
         m1   F*                    ( f: m1*[s mod d])
         D>F  m2  F*  F-  FDUP  F>D  ( adr d)  ROT 2! ;

: prng      ( f: -- random#)
    seed  (rand)  bigdiv            ( f: -- seed 2**31-1)
    FSWAP  FDUP  F0<        ( -- f) ( f: -- 2**31-1 seed)
    IF   FOVER  F+   THEN   FSWAP  F/  ;

: test  0.1  seed 2!  1000 0 DO  prng  FDROP  LOOP  seed 2@ D. ;
\ TEST 522329230  ok
```

The reader should convince himself of the identities

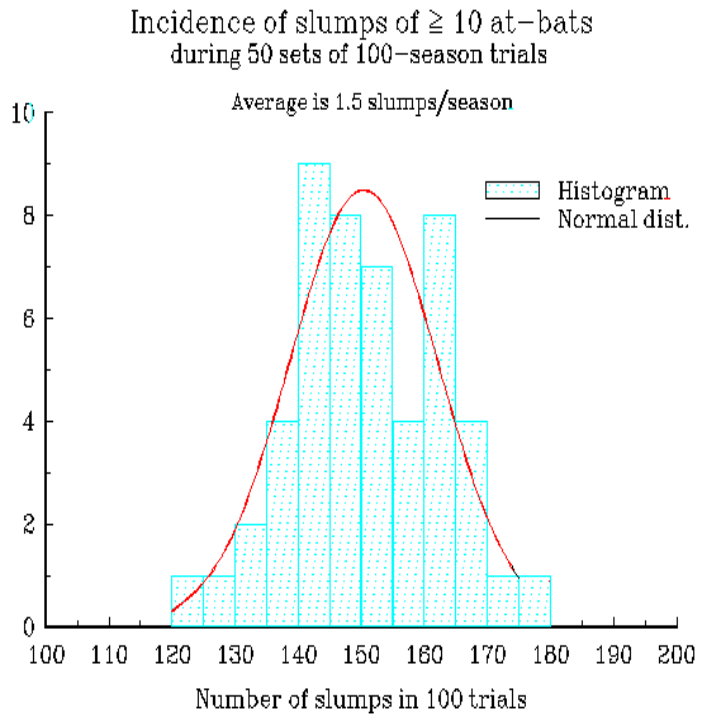
$$s' = (s \cdot m_1) \bmod b \equiv m_1 \cdot (s \bmod d) - m_2 \cdot \left[\frac{s}{d}\right]$$

$$\equiv m_1 \cdot \left(s - \left[\frac{s}{d}\right] \cdot d\right) - m_2 \cdot \left[\frac{s}{d}\right] .$$

where the square brackets mean "largest integer less than", *i.e.* round fractions down.

2. Batting practice

Having constructed a `prng` we are now in a position to apply it to some simulations. Consider a baseball player who plays N games per season. Assume his batting average is 0.300 (it could be anything.) If his chance of getting a hit, whenever he is at bat, is independent of any other time at bat, chance will lead to strings of hits or strikeouts—that is, to streaks or slumps. We night like to know what is the incidence of batting slumps—defined as 10 or more times at bat without a hit—for such a batter. This is a rather difficult problem to analyze using probability theory, but it is perfectly straightforward to simulate.



Basically, each time at bat is represented by generating a random number ξ. If ξ is between 0.0 and 0.3, a hit is recorded. If ξ is greater than 0.3, the trial is recorded as a strikeout. Next we need to check the strikeout counter to see if it has exceeded 10 strikeouts without an intervening hit. If so the event is recorded as a slump. The results from this simulation are shown to the left. The program that generated them is given below:

```
\ Simulation of batting slumps
\
\ ------------------------------------------------
\     (c) Copyright 1999  Julian V. Noble.          \
\       Permission is granted by the author to      \
\       use this software for any application pro-  \
\       vided this copyright notice is preserved.   \
\ ------------------------------------------------
\
\ This is an ANS Forth program requiring the
\   FLOAT, FLOAT EXT, FILE and TOOLS EXT wordsets.
\
\ Environmental dependencies:
\       Assumes independent floating point stack
```

```
MARKER -slump

INCLUDE prng.f      \ load the random number generator

\ ------------------------------------------- data structures
0.3 FCONSTANT  p0   \ batting average
250 VALUE N         \ # times @ bat/season

0 VALUE slump_len   \ place to hold the current

10 VALUE max_len    \ a slump is at least max_len strikeouts

0 VALUE #slumps

: initialize   ( 0.1  seed 2!)
               0 TO slump_len
               0 TO #slumps   ;

: got_a_hit?      ( -- f)
    prng  ( f: -- xi)   p0  F<=   ; \ its a hit if xi <= p0

: is_slump?   ( -- f)
    max_len  slump_len  <  ;

: end_slump!   is_slump?        \ was there a slump going?
          IF   #slumps  1+  TO #slumps   THEN
          0 TO slump_len  ;

: measure   ( -- )
    N 0  DO     got_a_hit?
                IF     end_slump!
                ELSE   slump_len  1+  TO  slump_len  THEN
    LOOP  ;

: stats     ( ncases --)
    initialize  DUP
    0 DO    measure   LOOP
    CR           . ."  trials of "        \ print results
        N        . ."  at-bats, giving "
        #slumps  . ."  slumps."
    ;
```

3. Non-uniformly distributed random variates

It often happens that we wish to simulate a process for which the random events are not representible by uniformly distributed real numbers in the interval (0, 1). For example we might wish to represent normally distributed random variables with mean $\bar{x}$ and standard deviation σ. There are several well-known algorithms for this. The Box-Muller algorithm uses the fact that

$$\int dx \int dy\, e^{-(x^2+y^2)/2} = \int d\theta \int d\left(e^{-\rho^2/2}\right)$$

Hence if we choose θ uniformly distributed on $[0, 2\pi]$ and $\eta = e^{-\rho^2/2}$ uniformly distributed on $(0, 1)$ we find

$$a = \sqrt{-2 \ln \eta}\ \cos\theta$$

and

$$b = \sqrt{-2 \ln \eta}\ \sin\theta$$

to be normally distributed random variables.

The problem with the algorithm in this form is that it requires that we evaluate three transcendental functions (`ln`, `sin` and `cos`) and a square root. However, we note that when x and y are independent random variables, uniformly distributed on the interval $(-1, 1)$, the random variable

$$u = \left(x^2 + y^2\right)\theta(1 - x^2 + y^2)$$

is a random variable that is uniformly distributed on $(0, 1)$. This allows us to implement the following procedure:

- pick two random numbers using the uniform `prng`;
- transform them to the interval $(-1, 1)$;
- compute u as above, rejecting it and starting over if it exceeds unity;
- compute the two random variables a and b given above, using the fact that

  $$a = x\sqrt{(-2 \ln u)/u} \qquad \text{and} \qquad b = y\sqrt{(-2 \ln u)/u}$$

  are independent normally distributed random numbers (on the interval $(-\infty, \infty)$) with mean 0 and standard deviation $\sigma = 1$.

Note that if we compute both numbers when the subroutine is called, we can save one of them so that we need only evaluate a logarithm and a square root every other time.

A standard FORTRAN function for the Box-Muller transformation is[7]

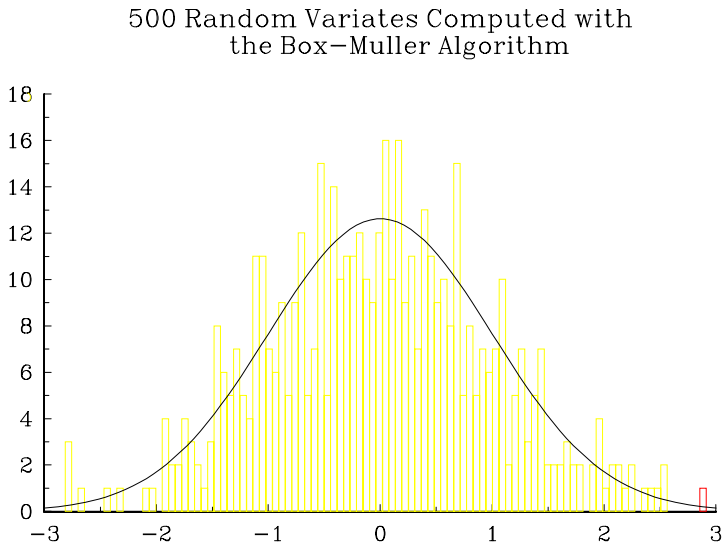
```
      FUNCTION GASDEV(IDUM)
      DATA ISET/0/
      IF (ISET.EQ.0) THEN
1       V1=2.*RAN1(IDUM)-1.
        V2=2.*RAN1(IDUM)-1.
        R=V1**2+V2**2
        IF(R.GE.1.)GO TO 1
        FAC=SQRT(-2.*LOG(R)/R)
        GSET=V1*FAC
        GASDEV=V2*FAC
        ISET=1
      ELSE
        GASDEV=GSET
        ISET=0
      ENDIF
      RETURN
      END
```

A Forth version appears below, to illustrate the difference in style between the two languages.

7. Press, *et al.*, *Numerical Recipes* (Cambridge University Press, New York, 1986).

```
\ Box-Muller algorithm for normal deviates
\ with mean 0 and sigma = 1.

\ This is an ANS Forth program requiring the
\   CORE EXT, FLOAT, FLOAT EXT, FILE and TOOLS EXT wordsets.
\
\ Environmental dependencies:
\       Assumes independent floating point stack

\ ---------------------- conditional compilation of non-Standard words
: undefined    BL WORD  FIND  NIP  0=  ;
undefined F2*       [IF]  : F2*    2.e0  F*  ;          [THEN]
undefined f^2       [IF]  : f^2   FDUP  F*   ;          [THEN]
undefined prng      [IF]  include  prng.f               [THEN]

\ -------------------- data structures
FVARIABLE   xi
0 VALUE  use_last

: get_x   prng  f2*  1e0  f-   ( f: -- x)  ;

: u_test    ( -- f) ( f: x y -- x y u)    \ true if u > 1
    FOVER  F^2  FOVER  F^2  F+  FDUP  1e0  F>  ;

: box_mul   ( f: -- a)
    use_last  IF   xi F@    FALSE  TO use_last   EXIT   THEN
        BEGIN   get_x  get_x   u_test
        WHILE   FDROP  FDROP  FDROP     \ if u > 1 start over
        REPEAT
    FDUP   FLN  FSWAP  F/   F2*         ( f: x y  -2*ln u / u )
    FABS   FSQRT                        ( f: x y sqrt[-2*ln u /u])
    FDUP   FROT  F*  xi  F!   F*   ;    ( f: -- a)
```

Output from this program is shown below in histogram form—the normal distribution corresponding to the number of samples is also shown.



Next let us consider how we would sample from an arbitrary probability distribution. We note that if the cumulative distributions for two sets of random variables are

$$P(x) = \int_{-\infty}^{x} dx' \, p(x')$$

$$Q(\xi) = \int_{-\infty}^{\xi} d\xi' \, q(\xi')$$

and we set

$$P(x) = Q(\xi)$$

then our problem is to find the function $x(\xi)$. If we suppose this is a known function and differentiate both sides with respect to ξ we find

$$p(x) \frac{dx}{d\xi} = q(\xi) \, .$$

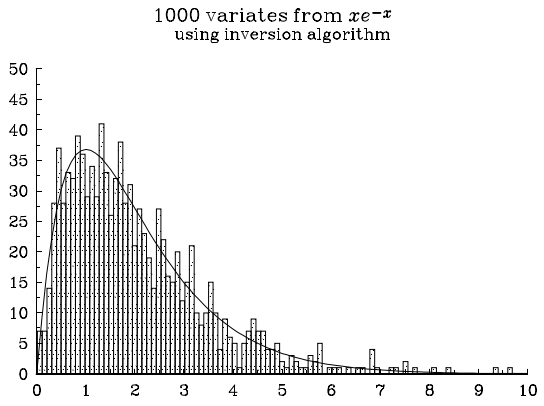
In general, since it is easy to generate uniformly distributed random variables we set

$$q(\xi) = \theta(\xi)\, \theta(1 - \xi)$$

and invert the relation

$$\int_{-\infty}^{x(\xi)} dx' \, p(x') = \xi \, .$$

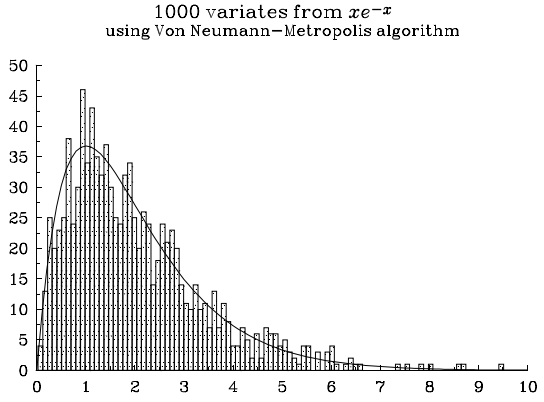
For some distributions this is difficult or impossible to accomplish in closed form, hence choosing random variates requires solving a transcendental equation. The figure to the right is a histogram of 1,000 variates chosen in this way from the distribution $p(x) = xe^{-x}$.



Alternatively, one may sample from the distribution $p(x)$ using the rejection method of Von Neumann: choose a random variable η from a uniform `prng` and also (uniformly) choose a value of x in the domain of $p(x)$ by scaling from $(0, 1)$. Then if

$$\eta \cdot \max p \leq p(x)$$

we keep the point, otherwise we reject it. A histogram of 1,000 such variates is shown to the right.



4. Young's two-slit experiment

Thomas Young is known for many contributions to physics but principally for having demonstrated the interference of light in 1800. He allowed coherent light to fall upon parallel slits in an opaque sheet, producing interference fringes on a screen. As is well known, the two-slit interference pattern is approximated by the intensity distribution

$$I(x) \propto 4\cos^2\left(\frac{a\pi x}{L\lambda}\right)$$

where a is the separation between the slits, L the distance to the viewing screen, and x the distance along the screen measured from the central maximum.

Young's result stood for more than a century as proof that light consisted of waves; in 1905, however, Albert Einstein introduced the idea that light consisted of energy quanta. Sir J.J. Thomson immediately recognized that this would imply the statistical nature of the preceding intensity distribution[8]. That is, one would expect the diffraction pattern to build up from individual photons striking the screen, the minima being places where photons had zero probability of arriving. One of Thomson's assistants, G.I. Taylor, thence attempted to photograph diffraction patterns in exceedingly dim light, hoping to see individual photons[9]. This proved impossible, however, with the available technology so the (negative) outcome led only to an upper bound on Planck's constant (which was, of course, much greater than the value predicted either by Einstein's photoelectric formula or by the blackbody radiation spectrum).

To simulate the buildup of the diffraction pattern on the screen, we choose points whose distribution function is

$$p(x) = \frac{\cos^2 x}{\int_{-X}^{X} du\, \cos^2 u}$$

on the rectangle

$$x \in (-X, X)$$

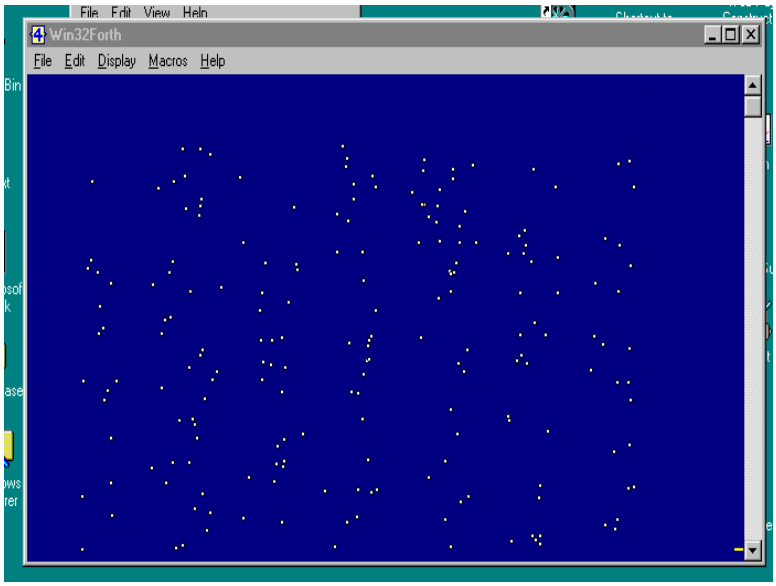
$$y \in (-Y, Y) .$$

A Forth program that does this by inverting the distribution (using the equation solver from Chapter 1) is shown on the next page, together with a screen shot of the output.

8. J.J. Thomson, *Proc. Camb. Phil. Soc.* **xiv** (1907) 417.
9. G.I. Taylor, *Proc. Camb. Phil. Soc.* **xvi** (1909) 415.

```
\ Simulation of Young 2-slit experiment with
\   individual photons
\
\ ************************************************
\     (c) Copyright 1998  Julian V. Noble.          *
\       Permission is granted by the author to      *
\       use this software for any application pro-  *
\       vided this copyright notice is preserved.   *
\ ************************************************
\
\ This is an ANS Forth program requiring the
\   FLOAT, FLOAT EXT, FILE and TOOLS EXT wordsets.
\
\ Environmental dependencies:
\       Assumes independent floating point stack
\       Assumes non-Standard graphics words from Win32Forth

MARKER -2slit
include prng.f
include ansfalsi.f
    0.1 seed 2!

    10e FCONSTANT x1
     5e FCONSTANT y1

   300e FCONSTANT y_lim
   500e FCONSTANT x_lim

    x1 F2*  FSIN  F2/  x1 F+  FCONSTANT  c

    FVARIABLE zeta

\ non-Standard Win32Forth graphics words
    WinDC theDC

: set_plot  ( – )
    CONDC PutHandle: theDC          \ initialize DC to the console
    0 0 800 600 BLUE  FillArea: theDC
    WHITE           BrushColor: theDC   ;

: offset    ( x y n – x+n y+n)
     TUCK  +  R  +  R  ;

: scale   ( f: x y – )  ( – x' y')
    y1  F/  1e  F-  F2/   y_lim  F*   FNEGATE   F>S
    x1  F/  1e  F+  F2/   x_lim  F*             F>S  SWAP  ;

: fplot    ( f: x y –)     \ plot a point
    scale  50 offset  2     FillCircle: theDC ;

\ continued on next page
```

```
\ Simulation
: new_zeta   prng  F2*  1e0  F-   c  F*  zeta  F!  ;

: new_y      prng  F2*  1e0  F-   y1 F*  ;

: f1    ( f: x - -  f1 = x+sin[2x]/2 - zeta )
        FDUP   F2*  FSIN  F2/  F+   zeta  F@  F-  ;

: new_x     ( f: - - x)      \ invert cos^2 distribution
            new_zeta            use( f1 x1 fnegate x1 1.0e-4 )falsi   ;

: 2slits    ( Npoints –)
        0.1 seed 2!                 \ initialize prng
        500 500 gotoxy              \ move cursor away
        set_plot                    \ make background
        0 DO
            new_x  new_y  fplot     \ plot a point
        LOOP
        BEGIN  KEY?  UNTIL          \ loop until key pressed
        KEY DROP    CLS  ;          \ clean up
```



Graphical output from 2slit.f -- 200 photons

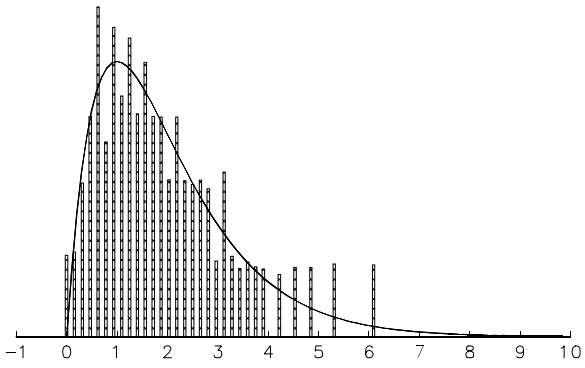
5. Random look-up tables

In our example above, the bottleneck was solving the transcendental equation for the random variables distributed according to the $\cos^2 x$ law. It is easy to see that if the integral representing the cumulative distribution cannot be expressed in closed form, in terms of well-known functions, the time to compute each point will become unacceptably large. Under such conditions even the Von Neumann rejection algorithm may be too slow.

There exists a remedy for this problem, based on the observation that for many simulations a histogram representation of the non-uniform distribution is perfectly adequate. Depending on the available storage, the histogram may be made sufficiently fine-grained that no loss of "physical significance" will be incurred. To construct such a representation it is enough to create a table of N variates by solving the equation

$$\int_{-\infty}^{x_k} dx\, p(x) = \frac{k}{N}, \quad k = 0, \ldots, N-1$$

Now, in what sense does such a table represent the desired random process? If we sample it using a pseudo-random number generator to generate random integers in the range [0, N–1], the resulting variates will generate a histogram that approximates the original distribution. An example of this is shown to the right, for the distribution $p(x) = x\,e^{-x}$, using a table of only 64 elements. If one employs a uniform random number generator with cycle length K, the table will be traversed in approximately N/K different orders, before repeating. This should be adequate for most uses.



The advantage of the random look-up table is its speed. All the computation is performed when the table is generated. Thus the run-time speed is simply the time to generate a random integer and fetch the appropriate variate from the table. In languages like FORTRAN or C we must create each such table and invoke it "by hand".

One advantage of Forth and other extensible languages[10] such as Lisp or Scheme, in this context, is that they permit us to define a generic constructor—a subroutine that creates packaged subroutines—of random lookup tables. Thus a large simulation that needs several such tables, each with its own distribution function and random number generator, can be much briefer to program. In the Forth program on the next pages, the subroutine `)DIST:` is the generic constructor.

10. "Object-oriented" languages like C++ and Smalltalk are similarly advantageous.

```
\ Forth tools for random lookup tables
\
\ ------------------------------------------------
\     (c) Copyright 1998  Julian V. Noble.          \
\       Permission is granted by the author to      \
\       use this software for any application pro-  \
\       vided this copyright notice is preserved.   \
\ ------------------------------------------------
\
\ This is an ANS Forth program requiring the
\   FLOAT, FLOAT EXT, FILE and TOOLS EXT wordsets.
\
\ Environmental dependencies:
\       Assumes independent floating point stack

FALSE [IF]

 Algorithm:
    Random lookup tables are described in
    Noble, "Scientific Forth", pp.

    If we need to sample a given random distribution, but have
    only a prng producing numbers y[k] evenly distributed on [0,1],
    we must solve the (generally transcendental) equation

        P(x<X[k]) = y[k]

    where P(x<X) is the cumulative distribution. In some cases this is
    not known in closed form and must be obtained by numerical quad-
    rature. Thus the inversion process can become a serious bottleneck
    in a large simulation, where many samples are required.

    The defining word DIST: creates a table with N entries

        x[k] = P^{-1} ( k/N ), k=0, 1, ... , N-1

    The lookup code defined by DOES> employs a prng with a long
    cycle to traverse the table in random order. This is equivalent
    to approximating the distribution function p(x) by a set of step
    functions (i.e. a histogram) of uneven width.


    IMPORTANT NOTE:
        Make sure the version of prng.f that you load is the version of
        November 24th, 1998 - 21:21. Earlier versions WILL NOT WORK!

[THEN]

MARKER -dist

: undefined    BL WORD  FIND  NIP  0=  ;

undefined s>f   [IF] : s>f   S>D  D>F  ;    [THEN]
undefined f>s   [IF] : f>s   F>D  D>S  ;    [THEN]
```

```
\ Conditionally define vectoring: for using function names as arguments
undefined  use( [IF]
: use(      ’       \ state-smart ’ for syntactic sugar
    STATE @  IF  POSTPONE LITERAL  THEN  ;  IMMEDIATE
’ NOOP  CONSTANT  ’noop
: v:   CREATE  ’noop  ,  DOES> PERFORM  ;   \ create dummy def’n
: ’dfa   ’ >BODY  ;                         ( -- data field address)
: defines    ’dfa   STATE @
             IF   POSTPONE  LITERAL    POSTPONE  !
             ELSE   !   THEN  ;  IMMEDIATE
\ end vectoring
[THEN]

include prng.f
include ansfalsi.f

\ program begins here

: SF,   \ "comma in" an IEEE 32-bit number
    HERE  SF!  1 SFLOATS  ALLOT  ;

: random_seed   ( -- d)
    TIME&DATE                   ( sec min hour day month year)
    2DROP 2DROP  60 *  +  S>D  ;

: random_ndx    ( adr -- index)
    DUP  CELL+                      ( -- adr adr+cell)
    (rand) bigdiv                   ( f: -- seed 2**31-1)
    FSWAP  FDUP  F0<        ( -- f) ( f: -- 2**31-1 seed)
    IF   FOVER  F+   THEN
    FSWAP  F/               ( adr)  ( f: prng)
    @  s>f  F*  f>s  ;      ( index)

: )DIST:    ( xt n --)              \ dist_name converts xi to X(xi)
    CREATE                          \ make entry under dist_name
        TUCK  ,                     \ save size
        random_seed  , ,            \ initialize seed
        SWAP  DUP  s>f              ( -- xt n)  ( f: -- n)
        0e  SF,                     \ first entry is 0.0e
        1 DO                        \ table will have n entries
            I s>f   FOVER   F/      ( xt)  ( f: -- n I/n)
            DUP  EXECUTE            ( xt)  ( f: -- n X[I/n])
            SF,                     \ store in table
        LOOP   DROP  FDROP          ( --)  ( f: --)
    DOES>   DUP  random_ndx  ( dup >R )
            SFLOATS  +  3 CELLS +   SF@   ( R> .  f.)  ;

\ Say:   use( inverse_dist_name  Table_size )DIST:  dist_name
\ auxiliary words with distribution
```

```
\ Example: inverse Poisson distribution

FVARIABLE xi

\ Want to solve P(X) = 1 - (1+X)*e^(-X) = xi ;  let u = e^(-X) or X = -ln(u)
: f1   ( f: u -- 1-[1-ln[u]]*u - xi )
    1e0  FOVER  FLN  F-  F*  FNEGATE  1e0  F+   xi F@  F-   ;

: invP      ( f: xi -- X[xi])   xi F!
    use( f1  1e-10  1e0  1e-6 )falsi    ( f: -- e^{-X})
    FLN  FNEGATE  ;
```

6. Stochastic differential equations

We now consider what happens when coefficients or driving forces of a system of differential equations are random or *stochastic*. An example is the damped harmonic oscillator bombarded by molecules at non-zero temperature:

$$m\ddot{x} + \gamma\dot{x} + m\omega^2 x = Q(t) .$$

With random driving forces the behavior of a system becomes unpredictable. There is no point in our considering the temporal behavior of a particular set of observations, since if we repeat the experiment next week we will get an entirely different set of observations. Random processes limit the useful information we can derive from a system to the distribution function of the solutions, or to ensemble averages of interesting quantities.

First let us consider what is an ensemble average. We imagine that a given experiment is run many different times, always with the same initial conditions; and that averages of coordinates $x(t)$ or velocities $\dot{x}(t)$—or of powers thereof—can be computed in the usual fashion.

If $x_k(t)$ is the value of position measured in the k'th experiment, at time t after the experiment began, then

$$\langle x(t) \rangle \overset{df}{=} \frac{1}{N} \sum_{k=1}^{N} x_k(t) ,$$

$$\mathrm{Var}\Big(x(t)\Big) \overset{df}{=} \Big\langle \Big(x(t) - \langle x(t) \rangle \Big)^2 \Big\rangle = \frac{1}{N} \sum_{k=1}^{N} \Big(x_k(t) - \langle x(t) \rangle \Big)^2 ,$$

etc. Assuming there is no steady component of the force $Q(t)$, we may say

$$\langle Q(t) \rangle \equiv 0 .$$

Since for any driving force we may express the solution as

$$x(t) = \frac{1}{m} \int_0^t ds\, K(t-s)\, Q(s) + x_0(t) ,$$

where

$$K(\tau) \stackrel{df}{=} \frac{1}{\Omega} e^{-\gamma\tau/2m} \sin(\Omega\,\tau)$$

and

$$\Omega^2 \stackrel{df}{=} \omega^2 - \frac{\gamma^2}{4m^2},$$

the ensemble average $\langle x(t)\rangle$ is just the homogeneous solution, $x_0(t)$.

To determine the mean-square position, however, we need to know something more about the random force than its (ensemble) mean of zero. From the above definitions we see that the variance,

$$\mathrm{Var}\Big[x(t)\Big] \equiv \Big\langle \Big[x(t) - \langle x(t)\rangle\Big]^2 \Big\rangle ,$$

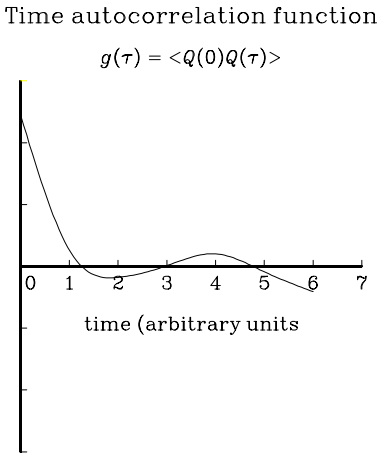
may be written as a double integral

$$\mathrm{Var}\,[x(t)] = \int_0^t du \int_0^t ds\;\; K(t-u)\,K(t-s)\,\langle Q(u)\,Q(s)\rangle .$$

The function

$$g(u-s) = \langle Q(u)\,Q(s)\rangle$$

is called the *time autocorrelation function* of the random process $Q(t)$. We have written it as a function of the difference between the two times, u and s because it is manifestly time-translation invariant.



The figure to the right displays the autocorrelation function of a pseudorandom number generator. We notice $g(u-s)$ is strongly peaked at 0, and falls off rapidly thereafter. Since the natural time scales, $1/\Omega$ and m/γ , for a noise–driven damped harmonic oscillator, are likely to be long compared to the time in which the autocorrelation function drops to zero, it is not unreasonable to approximate the latter by a δ–function:

$$g(\tau) \approx \sigma^2\delta(\tau) .$$

The variance of position is

$$\mathrm{Var}\Big[x(t)\Big] = \frac{\sigma^2}{m^2}\int_0^t ds\,\Big[K(s)\Big]^2$$

where we have used the obvious identity

$$\int_0^t ds\, f(t-s) \equiv \int_0^t ds\, f(s) .$$

Similarly, the velocity is given by the time derivative of $x(t)$,

$$\dot{x}(t) = \frac{1}{m}\int_0^t ds\,\Lambda(t-s)Q(s) + \dot{x}_0(t) - \frac{\gamma}{2m}x(t) ,$$

where

$$\Lambda(\tau) \stackrel{df}{=} e^{-\gamma\tau/2m}\cos(\Omega\tau) .$$

Therefore the mean velocity is

$$\langle \dot{x}(t) \rangle = \dot{x}_0(t) - \frac{\gamma}{2m} x_0(t) ,$$

and the corresponding variance is

$$\mathrm{Var}\left[\dot{x}(t)\right] = \sigma^2 \int_0^t ds \left[\Lambda(s)\right]^2 + \frac{\gamma^2\sigma^2}{4m^2}\int_0^t ds \left[K(s)\right]^2 + \frac{\gamma\sigma^2}{2m}\int_0^t ds\, K(s)\Lambda(s) .$$

We are interested in the behavior of the particle once it has reached equilibrium and forgotten its initial conditions. For large times,

$$\int_0^t ds \left[K(s)\right]^2 \to \frac{m}{2\gamma\omega^2}$$

$$\int_0^t ds \left[\Lambda(s)\right]^2 \to \frac{\Omega^2}{2\gamma\omega^2}$$

and

$$\int_0^t ds\, K(s)\Lambda(s) \to \frac{\gamma}{4m\omega^2} .$$

Combining these we find that after a long time the mean energy,

$$\langle H \rangle = \frac{m}{2}\left[\langle \dot{x}(t)^2 \rangle + \omega^2 \langle x(t)^2 \rangle\right] ,$$

has the limit

$$\langle H \rangle \to \frac{m^2\sigma^2}{2\gamma} .$$

That is, in equilibrium, an under-damped harmonic oscillator subject to a random driving force has mean energy independent of the oscillator frequency[11].

Now, suppose the random force represents collisions of molecules in a liquid or gas with the oscillator. (Of course, in that case the damping constant also arises from molecular collisions, *i.e.* viscosity.) Following Einstein, we can relate the constants σ^2 and γ to the temperature of the bath. A free particle in such a bath will, by the equipartition theorem, have average kinetic energy $\frac{1}{2}k_B T$ per degree of

11. Obviously any (damped) solution of the homogeneous equation contribute to the energy a term proportional to $e^{-\gamma t/m}$ that vanishes for large times.

freedom. It is easy to see (for example, take the limit of the over-damped solution of the harmonic oscillator) that in the limit $\omega^2 \to 0$, the average kinetic energy of a (damped) free particle is

$$\frac{1}{2} m\langle \dot{x}(t)^2 \rangle = \frac{m\sigma^2}{4\gamma} \equiv \frac{1}{2} k_B T .$$

In Brownian motion—the diffusion of small objects suspended in a liquid at (absolute) temperature T—there are no restoring forces, harmonic or otherwise. If one represents the density of objects as $\rho(x,t)$ then, as Einstein showed, ρ obeys the diffusion equation

$$\frac{\partial}{\partial t} \rho(x,t) = D \frac{\partial^2}{\partial x^2} \rho(x,t) .$$

where D is the *diffusion constant*. Since ρ represents the probability for finding a particle between x and $x+dx$ at time t, the mean position of the particles is

$$\langle x(t) \rangle = \int_{-\infty}^{\infty} dx \, x \, \rho(x,t) ;$$

it is then easy to see (integrate by parts) that

$$\langle x(t) \rangle = \text{constant} .$$

Similarly, the variance of the position is

$$\text{Var}\left[x(t)\right] = \int_{-\infty}^{\infty} dx \left(x^2 - \langle x(t) \rangle^2 \right) \rho(x,t) = 2D\,t$$

Comparing this with the variance (in position) of a free particle in a liquid or gas (the limit $\omega^2 \to 0$ of the damped harmonic oscillator with a random driving force),

$$\text{Var}\left[x(t)\right] = \frac{2k_B T}{m\gamma} t ,$$

we obtain Einstein's expression

$$D = \frac{k_B T}{6\pi\eta a} ,$$

for spherical particles of radius a, in a liquid of viscosity η, obeying Stokes' Law:

$$\gamma = 6\pi\eta a .$$

That is, measurement of the diffusion constant of a suspension of spheres allows direct measurement of the Boltzmann constant—or equivalently, of Avogadro's number. Perrin won the 1928 Nobel Prize in Physics for this and related measurements.

Returning to the one-dimensional, damped harmonic oscillator in a thermal bath, we see its average energy is $k_B T$. This was a key result used in deriving the Planck black body radiation formula before the modern formulation of quantum mechanics.