Random Processes

Monte Carlo Simulation

Random or Stochastic processes

You cannot predict from the observation of one event, how the next will come out

Examples:

Coin: the only prediction about outcome – 50% the coin will land on its tail

Dice: In large number of throws – probability 1/6

Question: What is the most probable number for the sum of two dice?

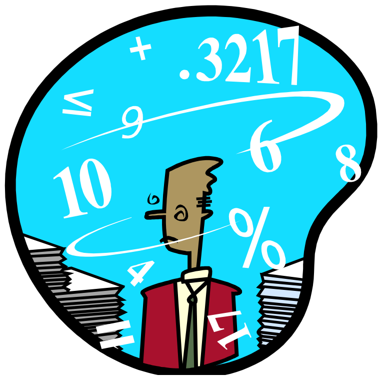
	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

36 possibilities

6 times – for **7**

Applications for MC simulation

- Stochastic processes
- Complex systems (science)
- Numerical integration
- Risk management
- Financial planning
- Cryptography
- …



How do we do that?

- You let the computer to throw “the coin” and record the outcome
- You need a program that generates randomly a variable

  … with relevant probability distribution

Part 1

Random number generators

Sources of Random Numbers

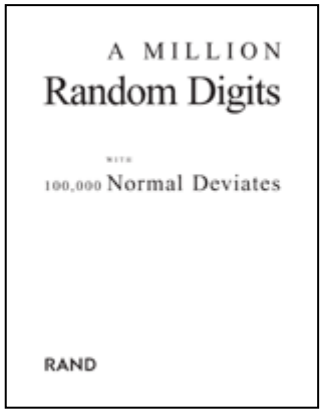
- Tables
- Hardware (external sources of random numbers – generates random numbers from a physics process.
- Software (source of pseudorandom numbers)

Tables

Most significant

A Million Random Digits with 100,000 Normal Deviates

by RAND



A MILLION
Random Digits
WITH
100,000 Normal Deviates

RAND

00000	10097	32533	76520	13586	34673	54876	80959	09117	39292	74945
00001	37542	04805	64894	74296	24805	24037	20636	10402	00822	91665
00002	08422	68953	19645	09303	23209	02560	15953	34764	35080	33606
00003	99019	02529	09376	70715	38311	31165	88676	74397	04436	27659
00004	12807	99970	80157	36147	64032	36653	98951	16877	12171	76833
00005	66065	74717	34072	76850	36697	36170	65813	39885	11199	29170
00006	31060	10805	45571	82406	35303	42614	86799	07439	23403	09732
00007	85269	77602	02051	65692	68665	74818	73053	85247	18623	88579
00008	63573	32135	05325	47048	90553	57548	28468	28709	83491	25624
00009	73796	45753	03529	64778	35808	34282	60935	20344	35273	88435
00010	98520	17767	14905	68607	22109	40558	60970	93433	50500	73998
00011	11805	05431	39808	27732	50725	68248	29405	24201	52775	67851
00012	83452	99634	06288	98083	13746	70078	18475	40610	68711	77817
00013	88685	40200	86507	58401	36766	67951	90364	76493	29609	11062
00014	99594	67348	87517	64969	91826	08928	93785	61368	23478	34113
.....										

Software - Random Number Generators

- There are no true random number generators but pseudo RNG!
- Reason: computers have only a limited number of bits to represent a number
- It means: the sequence of random numbers will repeat itself (period of the generator)

Good Random Number Generators

Two important issues:

1. randomness
2. knowledge of the distribution.

Other (still important) issues

1. independent of the previous number
2. long period
3. produce the same sequence if started with same initial conditions
4. fast

Two basic techniques for RNG

The standard methods of generating pseudorandom numbers use modular reduction in congruential relationships.

Two basic techniques for generating uniform random numbers:

1. congruential methods
2. feedback shift register methods.

For each basic technique there are many variations.

Linear Congruent Method for RNG

Generates a random sequence of numbers $\{x_1, x_2, \ldots x_k\}$ of length M over the interval $[0,M-1]$

$$\boxed{x_i = \mathrm{mod}(ax_{i-1} + c, M) = remainder\left(\frac{ax_{i-1} + c}{M}\right)} \quad 0 \le x_{i-1} < M$$

- starting value x_0 is called "seed"
- coefficients *a* and *c* should be chosen very carefully

note:

$$\mathrm{mod}(b, M) = b - \mathrm{int}(b/M) * M$$

the method was suggested by D. H. Lehmer in 1948

Example:

$$x_i = \mathrm{mod}(ax_{i-1} + c, M)$$

$$\mathrm{mod}(b, M) = b - \mathrm{int}(b / M) * M$$

a=4, c=1, M=9, x_1=3

x_2 = 4

x_3 = 8

x_4 = 6

x_{5-10} = 7, 2, 0, 1, 5, 3

interval: 0-8, i.e. [0,M-1]

period: 9 i.e. M numbers (then repeat)

Random Numbers on interval [A,B]

- Scale results from x_i on [0,M-1] to y_i on [0,1]

$$y_i = x_i / (M - 1)$$

- Scale results from x_i on [0,1] to y_i on [A,B]

$$y_i = A + (B - A)x_i$$

Magic numbers for Linear Congruent Method

- M (length of the sequence) is quite large
- However there is no overflow
  (for 32 bit machines $M=2^{31} \approx 2*10^9$)
- Good “magic” number for linear congruent method:

$$\boxed{x_i = \mathrm{mod}(ax_{i-1} + c, M)}$$

a = 16,807, c = 0, M = 2,147,483,647

for c = 0 “multiplicative congruential generator”:

Other Linear Congruential Generators

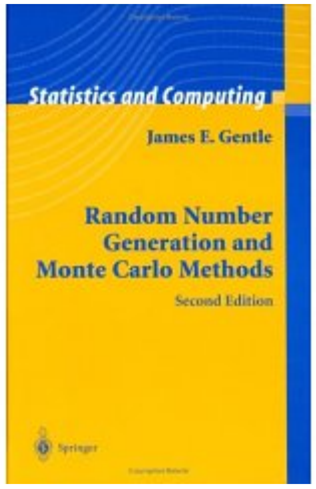
- ✓ Multiple Recursive Generators
  many versions including “Lagged Fibonacci”
- ✓ Matrix Congruential Generators
- ✓ Add-with-Carry, Subtract-with-Borrow, and Multiply - with-Carry Generators

Other Generators

- ✓ Nonlinear Congruential Generators
- ✓ Feedback Shift Register Generators
- ✓ Generators Based on Cellular Automata
- ✓ Generators Based on Chaotic Systems
- ✓ …

James E. Gentle – “Random Number Generation and Monte Carlo Methods

Second edition - 2004



Statistics and Computing

James E. Gentle

Random Number Generation and Monte Carlo Methods

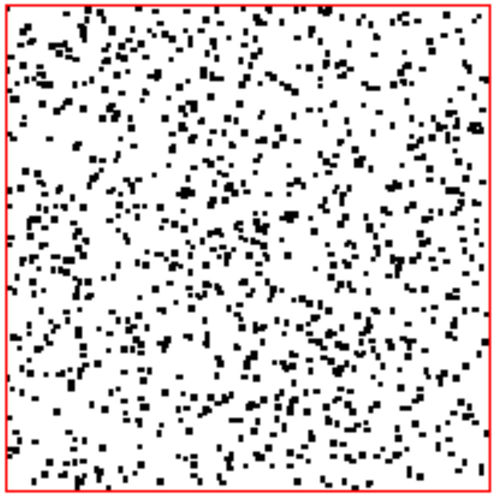
Second Edition

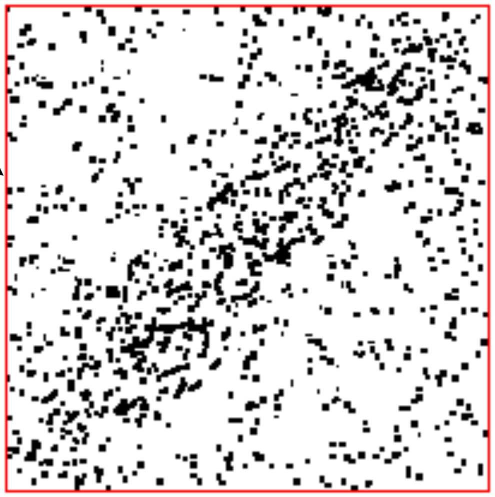
Springer

How can we check the RNG?

Plots:

- 2D figure, where x_i and y_i are from two random sequences (parking lot test)
- 3D figure (x_i, y_i, z_i)
- 2D figure for correlation (x_i, x_{i+k})





How can we check the RNG?

Example of other assessments

Uniformity. A random number sequence should contain numbers distributed in the unit interval with equal probability. Use bins.

k-th momentum $\left\langle x^k \right\rangle = \frac{1}{N}\sum_{i=1}^{N} x_i^k \approx \frac{1}{k+1}$

near-neighbor correlation $\frac{1}{N}\sum_{i=1}^{N} x_i x_{i+k} \approx \frac{1}{4}$

Software for RNG

C/C++ and Fortran (90,95) provide built-in uniform random number generators,

but … except for small studies, these built-in generators should be avoided.

A number of Fortran and C/C++ programs are available in

StatLib: http://lib.stat.cmu.edu/

NetLib: http://www.netlib.org/liblist.html

GAMS: http://gams.nist.gov/

GNU Scientific Library (GSL) http://www.gnu.org/software/gsl/

IMSL (International Mathematics and Statistics Library) libraries contain a large number of RNGs

“Industrial” methods in C/C++ and Fortran

- rand
- random
- drand48
- rn
- drand
- srand
- …

1. call SEED
   Changes the starting point of the pseudorandom number generator.
2. call RANDOM
   Returns a pseudorandom number greater than or equal to zero and less than one from the uniform distribution.

Standard RNG in C++

`#include <cstdlib>`	library
srand(*seed*)	is used to initialize the RNG
rand()	returns a pseudo random integer in the range 0 to RAND_MAX. RAND_MAX = 32767

Generating integer random numbers in a range i1 – i2:

```
random_i = i1 + (rand()%(i2-i1+1));
```

a better method to do the same

```
random_i = i1 + int(1.0*(i2-i1+1)*rand()/(RAND_MAX+1.0));
```

Generating real random numbers between 0.0 and 1.0

```
drandom = 1.0*rand()/(RAND_MAX+1);
```

Example: srand and rand in C++

```
// generate integer random numbers between i1 and i2
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <ctime>
using namespace std;

int main ()
{
  int nmax=10;             /* generate 10 random numbers*/
  int i1=1, i2=6, irandom;
  srand (123);             /* initial seed */
//srand(time(NULL)); // better to "randomize" seed values

  for (int i=0; i < nmax; i=i+1)
  {
   irandom = i1+rand()%(i2-i1+1);/* number between i1 & i2*/
   cout << " " << irandom << endl;
  }
  system("pause");
  return 0;
}
```

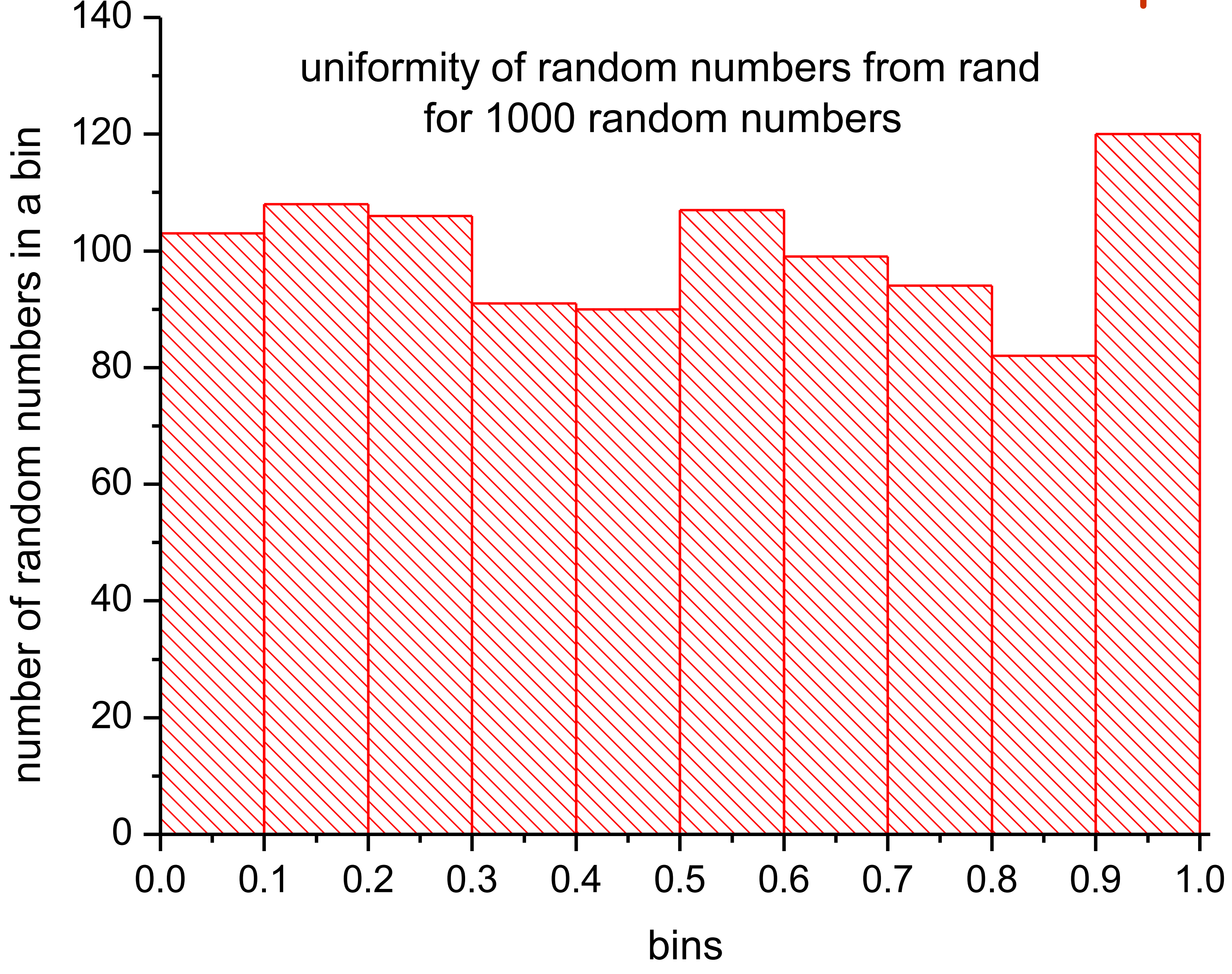
```
3
4
6
1
6
2
6
3
5
3
```

Example: cont. for float

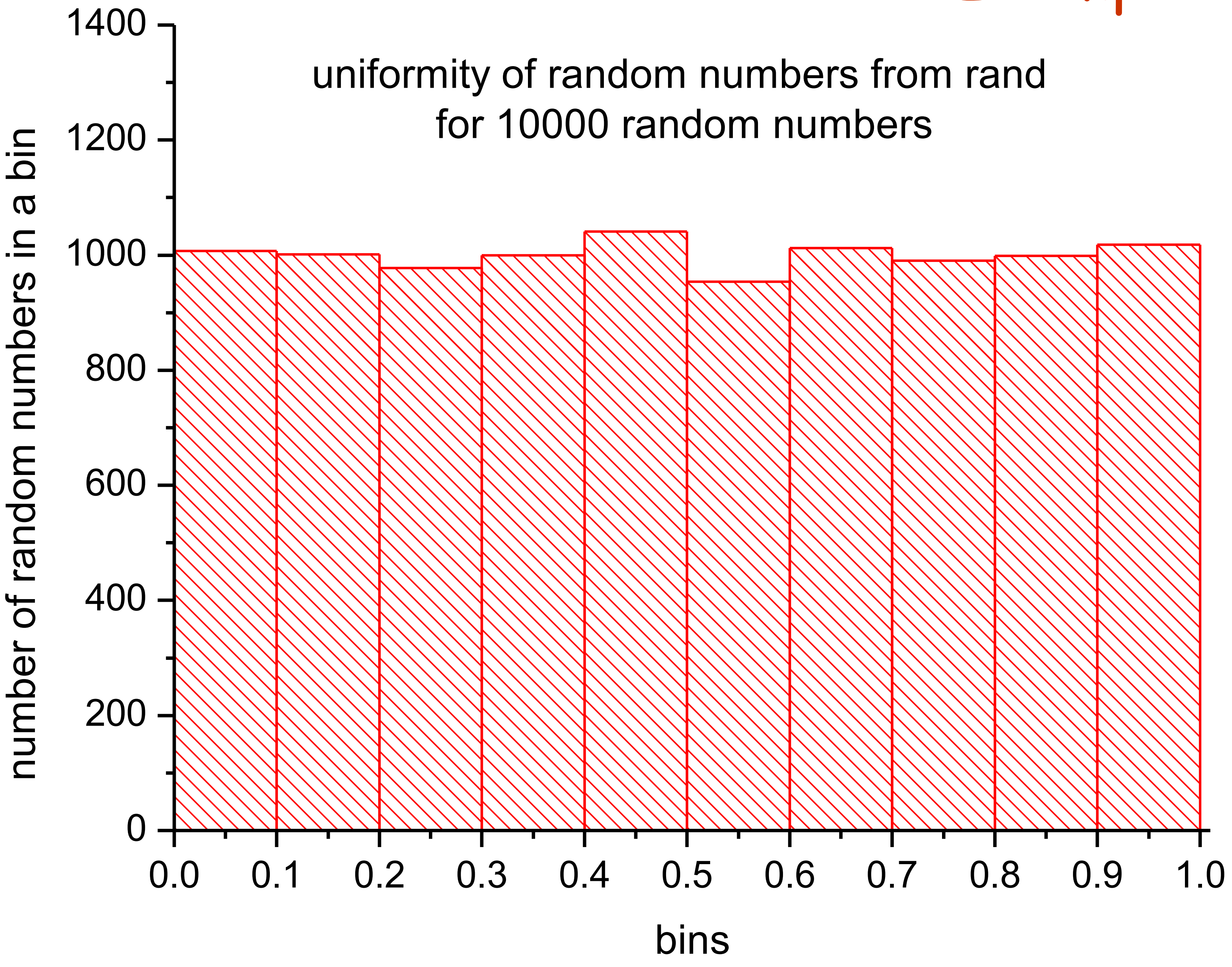
```
/* generate random numbers between 0.0 and 1.0 */
#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <cmath>
#include <ctime>
using namespace std;
int main ()
{
  int nmax = 10;    /*generate 10 random number*/
  double drandom;
  cout.precision(4);
  cout.setf(ios::fixed | ios::showpoint);

  srand(4567); /* initial seed value */
  for (int i=0; i < nmax; i=i+1)
  {
      drandom = 1.0*rand()/(RAND_MAX+1);
      cout << "d = " << drandom << endl;
  }
  system("pause");
  return 0;
}
```

```
0.4563
0.2816
0.4452
0.8693
0.8514
0.6432
0.0493
0.9999
0.6017
0.0548
```

Example


uniformity of random numbers from rand
for 1000 random numbers
number of random numbers in a bin
bins
0
20
40
60
80
100
120
140
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

Example

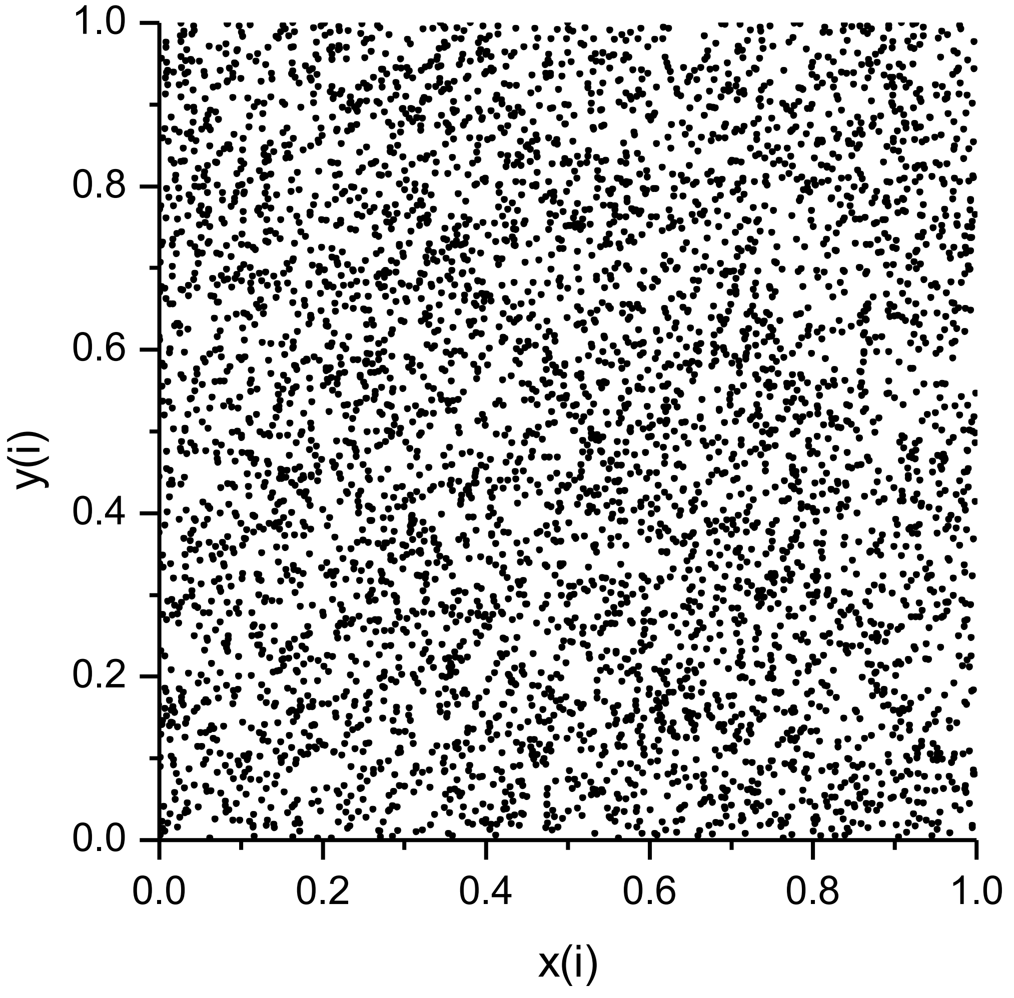

uniformity of random numbers from rand
for 10000 random numbers
number of random numbers in a bin
bins
0
200
400
600
800
1000
1200
1400
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

Example:

2D distribution for two random sequences x_i and y_i

k-th moment of the random number distribution

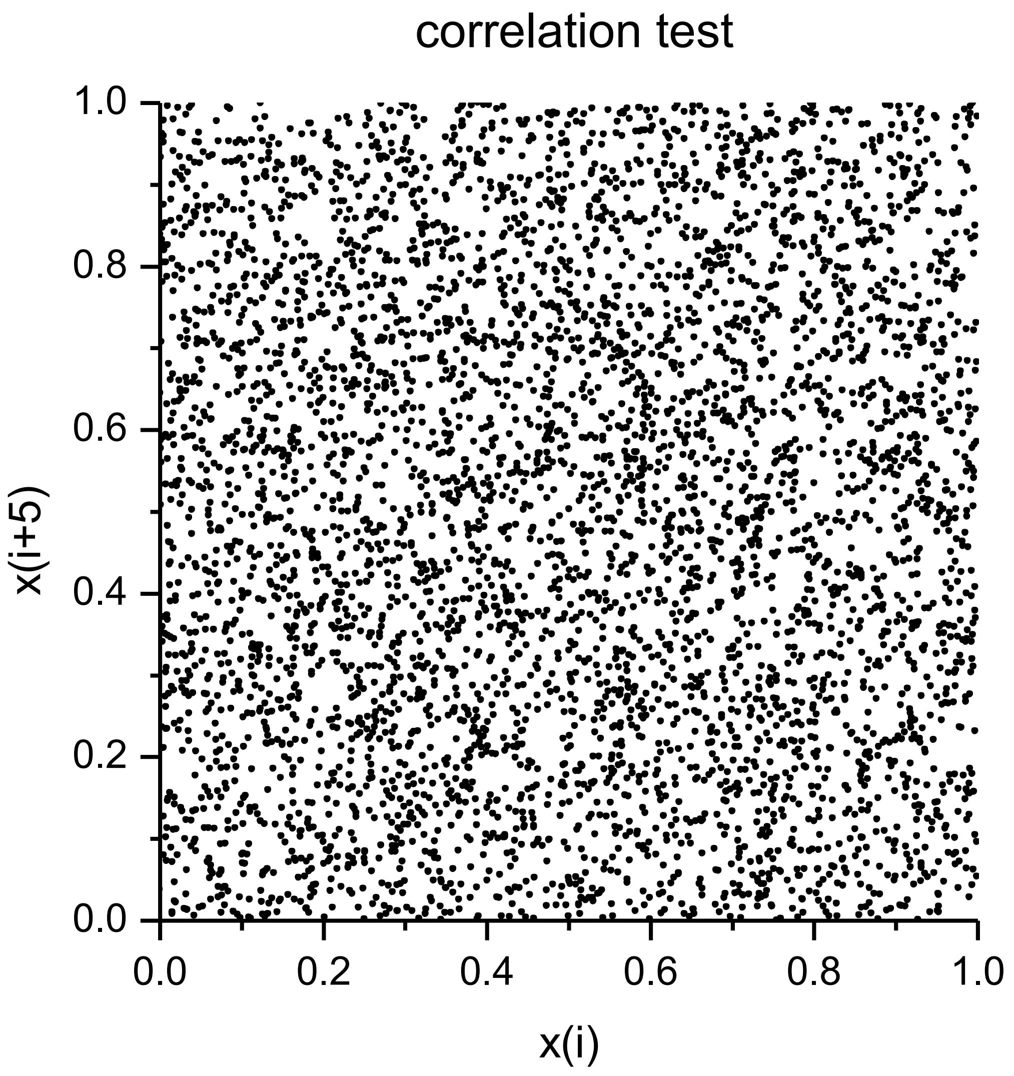
two random sequences (parking lot test)



5000 points,
k-th momentum $<x^4>=0.1991$
near-neighbor correlation = 0.2507

Example:

2D distribution for correlation (x_i, x_{i+5})



5000 points,
k-th momentum $<x^4>=0.1991$
near-neighbor correlation = 0.2507

Comment to rand in C++

“The version of rand() that comes with your C++ compiler will in all probability be a pretty simple generator and wouldn't be appropriate for scientific use. … It may well be random enough for use in simple programs and games.”

Jacobs, B. C++ Random Numbers. A tutorial for beginners, introducing the functions srand() and rand()

see also http://www.netlib.org/random/

Source codes for various random number generators in C and Fortran, including the RANLIB library

Practice 1 (homework)

1. Write a program to generate random numbers using the linear congruent method
2. Plot 2D distribution for two random sequences x_i and y_i
3. Plot 2D distribution for correlation (x_i, x_{i+4})
4. Evaluate 5-th moment of the random number distribution
5. Use some built-in RNG for problems 2-4.

Part 2

Monte Carlo Integration

Monte Carlo Integration

- There are very many methods for numerical integration
- Can MC approach compete with sophisticated methods?
- Can we gain anything from integration by “gambling”?

Problem: High-Dimensional Integration

Example: Integration for a system with 12 electrons.

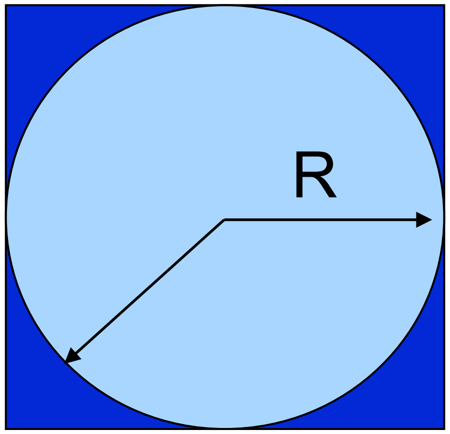
- 3*12=36 dimensional integral
- If 64 points for each integration then $=64^{36}$ points to evaluate
- For 1 Tera Flop computer = 10^{53} seconds
- That is … 3 times more then the age of the universe!

Integration by rejection hit and miss method

Example: area of a circle

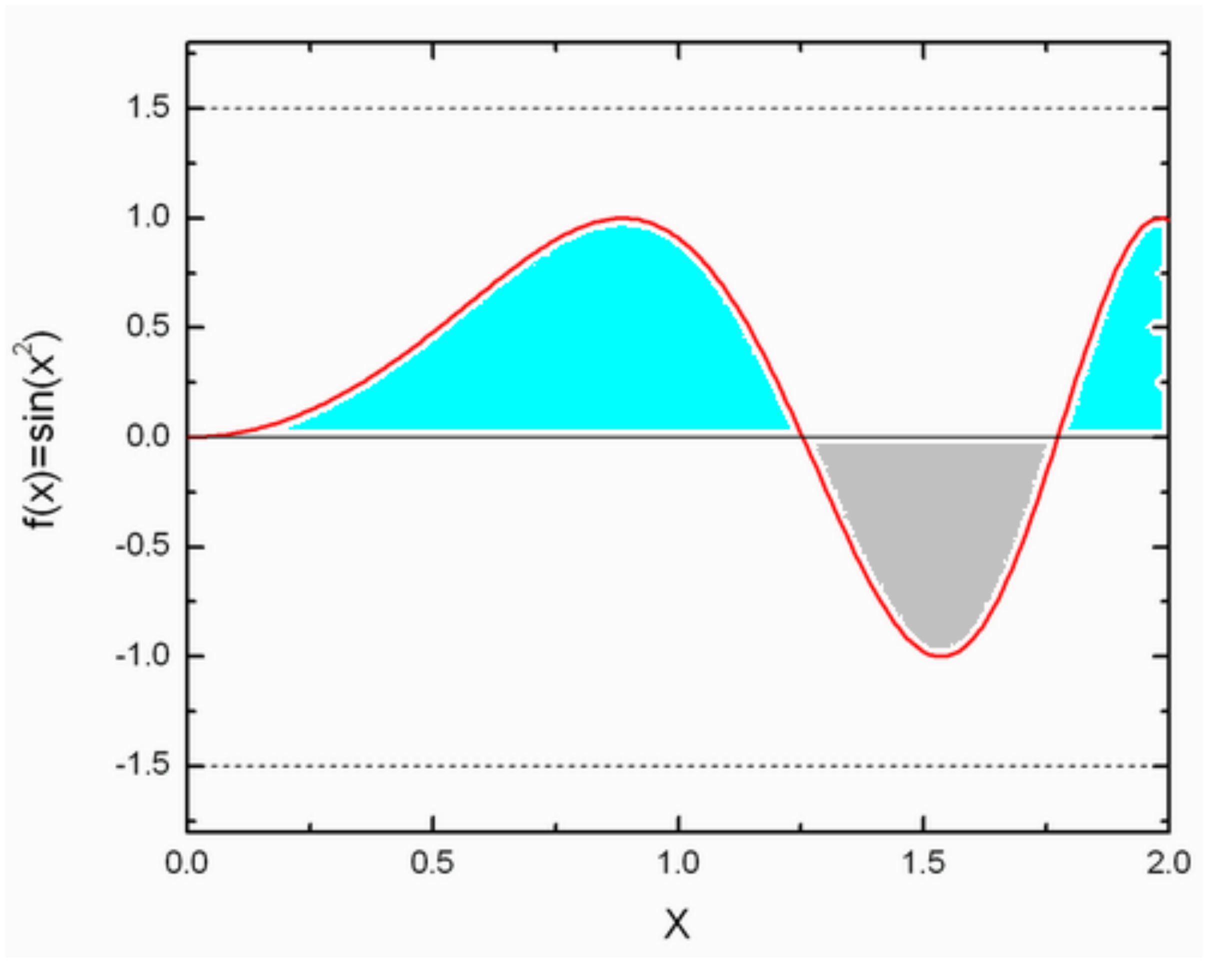
Radius: R

Area of the square: $4R^2$



1. loop over N
2. generate a pair of random numbers x and y on [-1,1]
3. if $(x*x+y*y) < 1$ then $m=m+1$
4. since $A_{circle}/A_{square} = m/N$
5. $A_{circle} = m/N*A_{square} = (m/N)*4R^2$

One more example



Compute N pairs of random numbers x_i and y_i with $0.0 \leq x \leq 2.0$ and $-1.5 \leq y \leq 1.5$.

$$F_n = A\left(\frac{n_+ - n_-}{N}\right)$$

Integration by mean value

$$I = \int_a^b f(x)dx = (b-a)\langle f \rangle$$

$$I = \int_a^b f(x)dx \approx (b-a)\frac{1}{N}\sum_{i=1}^{N} f(x_i) \pm \Delta S$$

$$\Delta S = (b-a)\sqrt{\frac{\langle f^2 \rangle - \langle f \rangle^2}{N}}$$

$$\langle f \rangle = \frac{1}{N}\sum_{i=1}^{N} f(x_i) \qquad \langle f^2 \rangle = \frac{1}{N}\sum_{i=1}^{N} f^2(x_i)$$

the error evaluation is based on the normal distribution

Traditional methods (midpoint, Simpson, …) – N points are chosen with equal spacing.
Monte Carlo method – random sampling

Midpoint vs Monte Carlo method error

Consider a one-dimensional integral: $\int_{x_l}^{x_h} f(x)\,dx$. We can evaluate this integral numerically by dividing the interval x_l to x_h into N identical subdivisions of width

$$h = \frac{x_h - x_l}{N}.$$

Let x_i be the midpoint of the ith subdivision, and let $f_i = f(x_i)$. Our approximation to the integral takes the form

$$\int_{x_l}^{x_h} f(x)\,dx \simeq \sum_{i=1}^{N} f_i\,h$$

This "midpoint method" is not particularly accurate, but is very easy to generalize to multi-dimensional integrals.
What is the error associated with the midpoint method?
The error is the product of the error per subdivision, which is $O(h^2)$, and the number of subdivisions, which is $O(h^{-1})$. The error per subdivision follows from the linear variation of $f(x)$ within each subdivision. Thus, the overall error is $O(h^2) \times O(h^{-1}) = O(h)$. Since, $h \propto N^{-1}$, we can write

$$\int_{x_l}^{x_h} f(x)\,dx \simeq \sum_{i=1}^{N} f_i\,h + O(N^{-1}).$$

Error in midpoint m-d for 2-dim integral

Let us now consider a two-dimensional integral. For instance, the area enclosed by a curve. We can evaluate such an integral by dividing space into identical squares of dimension h, and then counting the number of squares, N (say), whose midpoints lie within the curve. Our approximation to the integral then takes the form

$$A \simeq N h^2.$$

This is the two-dimensional generalization of the midpoint method. What is the error associated with the midpoint method in two-dimensions? The error is generated by those squares which are intersected by the curve. These squares either contribute wholly or not at all to the integral, depending on whether their midpoints lie within the curve. In reality, only those parts of the intersected squares which lie within the curve should contribute to the integral The error is the product of the area of a given square, which is $O(h^2)$, and the number of squares intersected by the curve, which is $O(h^{-1})$
$\Rightarrow$ the overall error is $O(h^2) \times O(h^{-1}) = O(h) = O(N^{-1/2})$.

$$\Rightarrow \quad A = N h^2 + O(N^{-1/2}).$$

Error in midpoint m-d for 3-dim integral

Let us now consider a three-dimensional integral represented by the volume enclosed by a surface. We can evaluate such an integral by dividing space into identical cubes of dimension h, and then counting the number of cubes, N (say), whose midpoints lie within the surface. Our approximation to the integral then takes the form

$$V \simeq N h^3.$$

This is the three-dimensional generalization of the midpoint method.

What is the error associated with the midpoint method in three-dimensions? The error is generated by those cubes which are intersected by the surface. These cubes either contribute wholly or not at all to the integral, depending on whether their midpoints lie within the surface. In reality, only those parts of the intersected cubes which lie within the surface should contribute to the integral. The error is the product of the volume of a given cube, which is $O(h^3)$, and the number of cubes intersected by the surface, which is $O(h^{-2})$
$\Rightarrow$ the overall error is $O(h^3) \times O(h^{-2}) = O(h) = O(N^{-1/3})$.

$$\Rightarrow \quad V = N h^3 + O(N^{-1/3}).$$

Error in midpoint m-d for d-dim integral

Finally, consider using the midpoint method to evaluate the volume, V, of a d-dimensional hypervolume enclosed by a $(d-1)$-dimensional hypersurface. It is clear, from the above examples, that

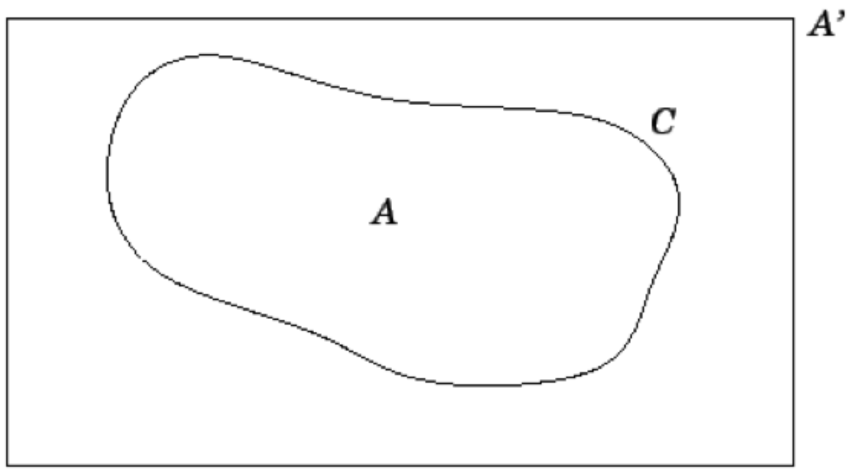
$$V = N\,h^d + O(N^{-1/d}),$$

where N is the number of identical hypercubes into which the hypervolume is divided. Note the increasingly slow fall-off of the error with N as the dimensionality, d, becomes greater. The explanation for this phenomenon is quite simple. Suppose that $N = 10^6$. With $N = 10^6$ we can divide a unit line into (identical) subdivisions whose linear extent is 10^{-6}, but we can only divide a unit area into subdivisions whose linear extent is 10^{-3}, and a unit volume into subdivisions whose linear extent is 10^{-2}. Thus, for a fixed number of subdivisions the grid spacing (and, hence, the integration error) increases dramatically with increasing dimension.

Error in Monte Carlo method

Let us now consider the Monte-Carlo method for evaluating multi-dimensional integrals. Consider, for example, the evaluation of the area, A, enclosed by a curve, C. Suppose that the curve lies wholly within some simple domain of area A', as illustrated below. Let us generate N' points which are randomly distributed throughout A'. Suppose that N of these points lie within curve C. Our estimate for the area enclosed by the curve is simply

$$A = \frac{N}{N'} A'. \qquad (*)$$



What is the error associated with the Monte-Carlo integration method? Well, each point has a probability $p = A/A'$ of lying within the curve. Hence, the determination of whether a given point lies within the curve is like the measurement of a random variable x which has two possible values:
1 (corresponding to the point being inside the curve) with probability p, and
0 (corresponding to the point being outside the curve) with probability $1 - p$.
If we make N' measurements of x (i.e., if we scatter N' points throughout A') then the number of points lying within the curve is

$$N = \sum_{i=1,N'} x_i,$$

where x_i denotes the ith measurement of x. Now, the mean value of N is

$$\bar{N} = \sum_{i=1,N'} \bar{x} = N' \bar{x},$$

where

$$\bar{x} = 1 \times p + 0 \times (1 - p) = p.$$

Hence,

$$\bar{N} = N' p = N' \frac{A}{A'},$$

which is consistent with Eq. $(*)$. We conclude that, on average, a measurement of N leads to the correct answer.

But, what is the scatter in such a measurement? Well, if σ represents the standard deviation of N then we have

$$\sigma^2 = \overline{(N - \bar{N})^2},$$

which can also be written

$$\sigma^2 = \sum_{i,j,=1,N'} \overline{(x_i - \bar{x})(x_j - \bar{x})}.$$

However, $\overline{(x_i - \bar{x})(x_j - \bar{x})}$ equals $\overline{(x - \bar{x})^2}$ if $i = j$, and equals zero, otherwise, since successive measurements of x are uncorrelated. Hence,

$$\sigma^2 = N' \, \overline{(x - \bar{x})^2}.$$

Now,

$$\overline{(x - \bar{x})^2} = \overline{(x^2 - 2\,x\,\bar{x} + \bar{x}^2)} = \overline{x^2} - \bar{x}^2,$$

and

$$\overline{x^2} = 1^2 \times p + 0^2 \times (1 - p) = p.$$

Error in the MC method

Thus,

$$\overline{(x-\bar{x})^2} = p - p^2 = p\,(1-p),$$

giving

$$\sigma = \sqrt{N'\,p\,(1-p)}.$$

Finally, since the likely values of N lie in the range $N = \bar{N} \pm \sigma$, we can write

$$N = N'\,\frac{A}{A'} \pm \sqrt{N'\,\frac{A}{A'}\left(1-\frac{A}{A'}\right)}.$$

It follows from Eq. $(*)$ that

$$A = A'\,\frac{N}{N'} \pm \frac{\sqrt{A\,(A'-A)}}{\sqrt{N'}} \qquad (**).$$

$\Rightarrow$ the error scales like $(N')^{-1/2}$.

Error in the MC method in d dimensions

The Monte-Carlo method generalizes immediately to d-dimensions. For instance, consider a d-dimensional hypervolume V enclosed by a $(d-1)$-dimensional hypersurface A. Suppose that A lies wholly within some simple hypervolume V'. We can generate N' points randomly distributed throughout V'. Let N be the number of these points which lie within A. It follows that our estimate for V is simply

$$V = \frac{N}{N'}\, V'.$$

Now, there is nothing in our derivation of Eq. $(**)$ which depends on the fact that the integral in question is two-dimensional $\Rightarrow$ we can generalize this equation to give

$$V = V'\,\frac{N}{N'} \pm \frac{\sqrt{V\,(V'-V)}}{\sqrt{N'}}.$$

We conclude that the error associated with Monte-Carlo integration always scales like $(N')^{-1/2}$, irrespective of the dimensionality of the integral.

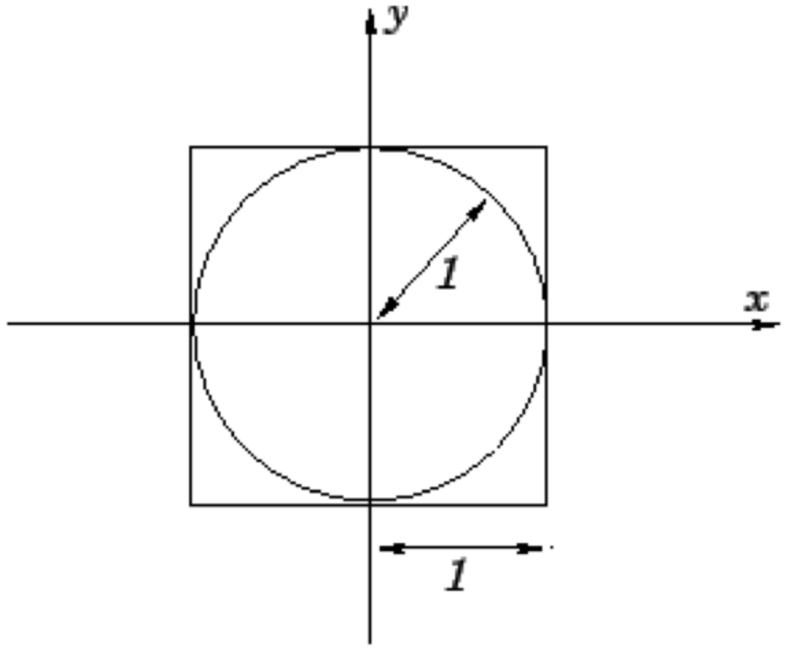
Comparison of midpoint and MC methods

In the midpoint method, we fill space with an evenly spaced mesh of N (say) points (i.e., the midpoints of the subdivisions), and the overall error scales like $N^{-1/d}$, where d is the dimensionality of the integral.
In the Monte-Carlo method, we fill space with N (say) randomly distributed points, and the overall error scales like $N^{-1/2}$, irrespective of the dimensionality of the integral.
For a one-dimensional integral ($d = 1$), the midpoint method is more efficient than the Monte-Carlo method, since in the former case the error scales like N^{-1}, whereas in the latter the error scales like $N^{-1/2}$. For a two-dimensional integral ($d = 2$), the midpoint and Monte-Carlo methods are both equally efficient, since in both cases the error scales like $N^{-1/2}$. Finally, for a three-dimensional integral ($d = 3$), the midpoint method is less efficient than the Monte-Carlo method, since in the former case the error scales like $N^{-1/3}$, whereas in the latter the error scales like $N^{-1/2}$.

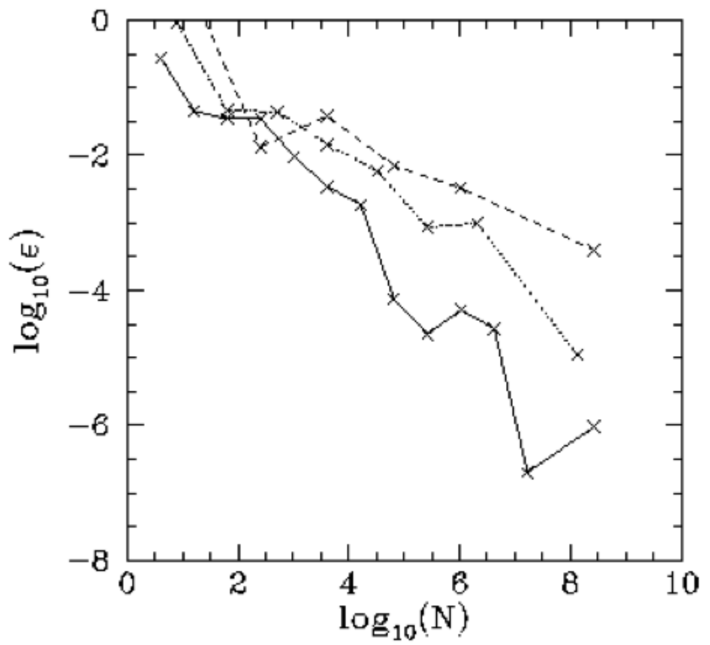
$\Rightarrow$ For a sufficiently high dimension integral the Monte-Carlo method is always going to be more efficient than an integration method (such as the midpoint method) which relies on a uniform grid.

Example: volume of a d-dim sphere

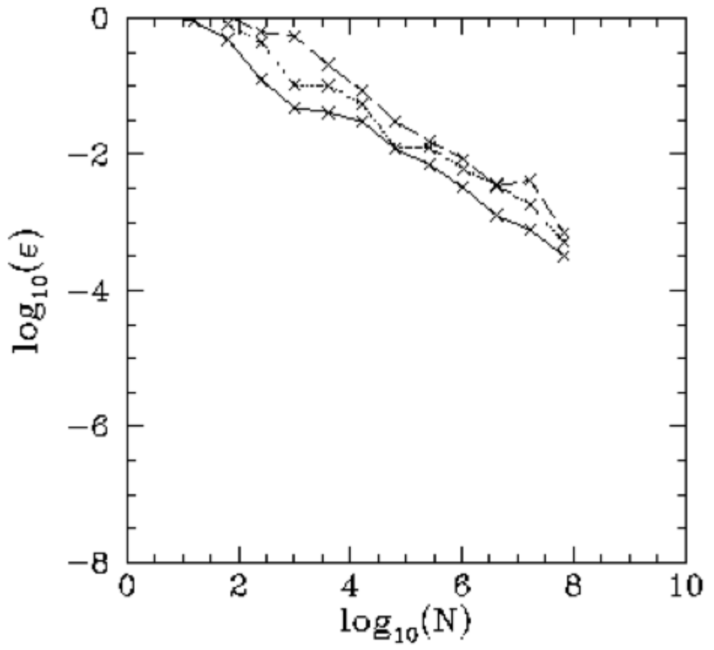
Let us evaluate the volume of a unit-radius d-dimensional sphere, where d runs from 2 to 4, using both the midpoint and Monte-Carlo methods. For both methods, the domain of integration is a cube, centred on the sphere, which is such that the sphere just touches each face of the cube, as illustrated below



The integrals are the area of a unit-radius circle (solid curve), the volume of a unit-radius sphere (dotted curve), and the volume of a unit-radius 4-sphere (dashed curve).



The integration error, ϵ, versus the number of grid-points, N, for three integrals evaluated using the midpoint method.



The integration error, ϵ, versus the number of points, N, for three integrals evaluated using the Monte-Carlo method.

Up to now, we have only considered how the Monte-Carlo method can be employed to evaluate a rather special class of integrals in which the integrand function can only take the values 0 or 1.
However, the Monte-Carlo method can easily be adapted to evaluate more general integrals.
Suppose that we wish to evaluate $\int f\,dV$, where f is a general function and the domain of integration is of arbitrary dimension. We proceed by randomly scattering N points throughout the integration domain and calculating f at each point. Let x_i denote the ith point. The Monte-Carlo approximation to the integral is simply

$$\int f\,dV = \frac{1}{N}\sum_{i=1,N} f(x_i) + O\left(\frac{1}{\sqrt{N}}\right).$$

Error in MC vs Simpson integration

Error in Monte - Carlo	1D case	$\frac{1}{\sqrt{N}}$
Error in Monte - Carlo	nD case	$\frac{1}{\sqrt{N}}$
Error in Simson	1D case	$\frac{1}{N^4}$
Error in Simson	nD case	$\left(\frac{1}{N^4}\right)^{1/n}$

at n 7 or 8 the error in Monte Carlo integration is similar to that of conventional scheme

Example: 1D integration (C++)

```
 double int_mc1d(double(*f)(double), double a, double b, int n)
/* 1D intergration using Monte-Carlo method for f(x) on [a,b]
input: f - Function to integrate (supplied by a user)
       a - Lower limit of integration
       b - Upper limit of integration
       n - number random points
output:r - Result of integration
Comments: be sure that following headers are included
    #include <cstdlib>
    #include <ctime>
*/
{
   double r, x, u;
   srand(time(NULL)); /* initial seed value (use system time) */

   r = 0.0;

   for (int i = 1; i <= n; i=i+1)
   {
     u = 1.0*rand()/(RAND_MAX+1); // random between 0.0 and 1.0
        x = a + (b-a)*u;           // random x between a and b
        r = r + f(x);
    }
    r = r*(b-a)/n;
    return r;
}
```

Example $\int_0^{\pi} \sin(x)\,dx = 2.0$

n	Trapez.	Simpson	Monte Carlo
2	1.570796	2.094395	2.483686
4	1.896119	2.004560	2.570860
8	1.974232	2.000269	2.140117
16	1.993570	2.000017	1.994455
32	1.998393	2.000001	2.005999
64	1.999598	2.000000	2.089970
128	1.999900	2.000000	2.000751
256	1.999975	2.000000	2.065036
512	1.999994	2.000000	2.037365
1024	1.999998	2.000000	1.988752
2048	2.000000	2.000000	1.989458
4096	2.000000	2.000000	1.991806
8192	2.000000	2.000000	2.000583
16384	2.000000	2.000000	1.987582
32768	2.000000	2.000000	1.991398
65536	2.000000	2.000000	1.997360

Example $\int_0^{\pi} \frac{x}{x^2+1}\cos(10x^2)dx = 0.0003156$

n	Trapez.	Simpson	Monte Carlo
64	0.004360	-0.013151	0.081207
128	0.001183	-0.001110	0.155946
256	0.000526	-0.000311	0.071404
512	0.000368	0.000006	0.002110
1024	0.000329	0.000161	-0.004525
2048	0.000319	0.000238	-0.010671
4096	0.000316	0.000277	0.000671
8192	0.000316	0.000296	-0.009300
16384	0.000316	0.000306	-0.009500
32768	0.000316	0.000311	-0.005308
65536	0.000316	0.000313	-0.000414
131072	0.000316	0.000314	0.001100
262144	0.000316	0.000315	0.001933
524288	0.000316	0.000315	0.000606
1048576	0.000316	0.000315	-0.000369
2097152	0.000316	0.000316	0.000866
4194304	0.000316	0.000316	0.000330

many methods to increase accuracy

Example: antithetic variates – using “mirror points”

$$I = \int_a^b f(x)dx \approx (b-a)\frac{1}{N}\sum_{i=1}^{N/2}\left(f(x_i) + f(a+(b-x_i))\right)$$

Antithetic variates have negative covariances, thus reducing the variance of the sum

more methods can be found in
James E. Gentle – “Random Number Generation and Monte Carlo Methods

Second edition - 2004

Multidimensional Monte Carlo

$$\int_a^b dx \int_c^d dy f(x,y) \cong (b-a)(d-c)\frac{1}{N}\sum_{i=1}^{N} f(x_i, y_i)$$

Example: nD integration (C++)

```
double int_mckd(double(*fn)(double[],int),double a[],
       double b[], int n, int m)
/* input is similar to 1D integration*/
{
   double r, x[n], p;
   int i, j;
   srand(time(NULL));/* initial seed value (use system time) */
   r = 0.0;
   p = 1.0;

// step 1: calculate the common factor p
    for (j = 0; j < n; j = j+1) p = p*(b[j]-a[j]);

// step 2: integration
    for (i = 1; i <= m; i=i+1)
    {
//       calculate random x[] points
        for (j = 0; j < n; j = j+1)
        {
            x[j] = a[j] + (b[j]-a[j])*rand()/(RAND_MAX+1);
        }
        r = r + fn(x,n);
    }
    r = r*p/m;
    return r;
}
```

$$\int_0^1 dx_1 \int_0^1 dx_2 \int_0^1 dx_3 \int_0^1 dx_4 \int_0^1 dx_5 \int_0^1 dx_6 \int_0^1 (x_1 + x_2 + \ldots + x_7)^2 \, dx_7 = 12.83333333$$

Example

```
           7D Integral
      8     11.478669
     16     12.632578
     32     13.520213
     64     13.542921
    128     13.263171
    256     13.178140
    512     12.850561
   1024     12.747383
   2048     12.745207
   4096     12.836080
   8192     12.819113
  16384     12.790508
  32768     12.765735
  65536     12.812653
 131072     12.809303
 262144     12.831216
 524288     12.832844
total elapsed time = 1 seconds
```

Practice: Integration

- Use Monte Carlo integration (both rejection and mean value methods) to evaluate

$$\int_0^3 \exp(-x)dx \quad \text{and} \quad \int_0^5 \sin(2x^2)dx$$

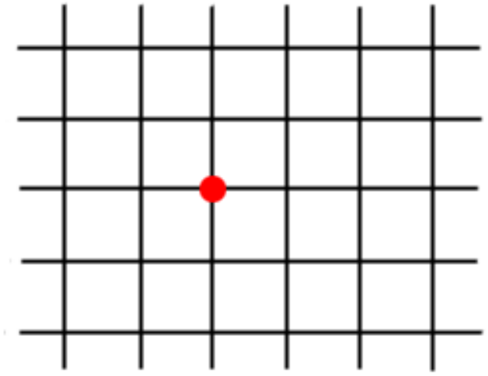
- Evaluate 7-D integral

$$\int_0^1 dx_1 \int_0^1 dx_2 \int_0^1 dx_3 \int_0^1 dx_4 \int_0^1 dx_5 \int_0^1 dx_6 \int_0^1 (x_1 + x_2 + \ldots + x_7)^2 dx_7$$

Part 3

Random Walk

Random Walk



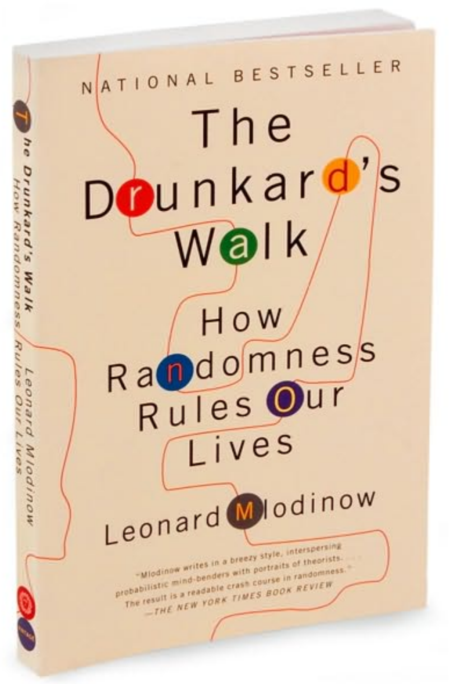
A simple random walk is a sequence of unit steps where each step is taken in the direction of one of the coordinate axis, and each possible direction has equal probability of being chosen.

Random walk on a lattice:

- In two dimensions, a single step starting at the point with integer coordinates (x,y) would be equally likely to move to any of one of the four neighbors (x+1,y), (x-1,y), (x,y+1) and (x,y-1).
- In one dimension walk there are two possible neighbors
- In three dimensions there are six possible neighbors.

Random Walk simulates:

- Brownian motion
  (answer the question - how many collisions, on average, a particle must take to travel a distance R).
- Electron transport in metals, …
- …

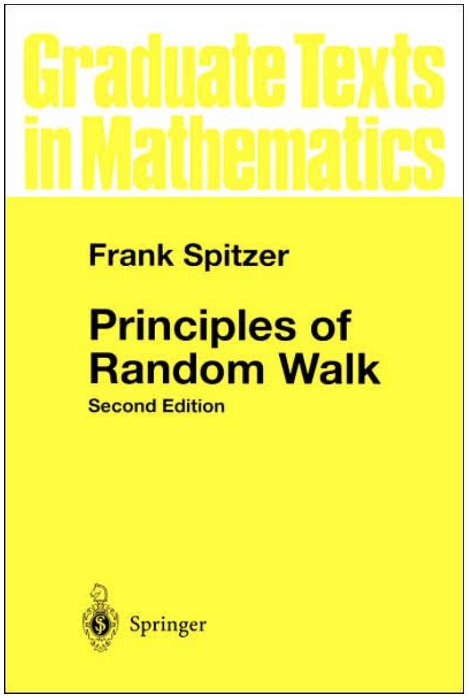

NATIONAL BESTSELLER

The Drunkard's Walk

How Randomness Rules Our Lives

Leonard Mlodinow

"Mlodinow writes in a breezy style, interspersing probabilistic mind-benders with portraits of theorists. The result is a readable crash course in randomness." —THE NEW YORK TIMES BOOK REVIEW

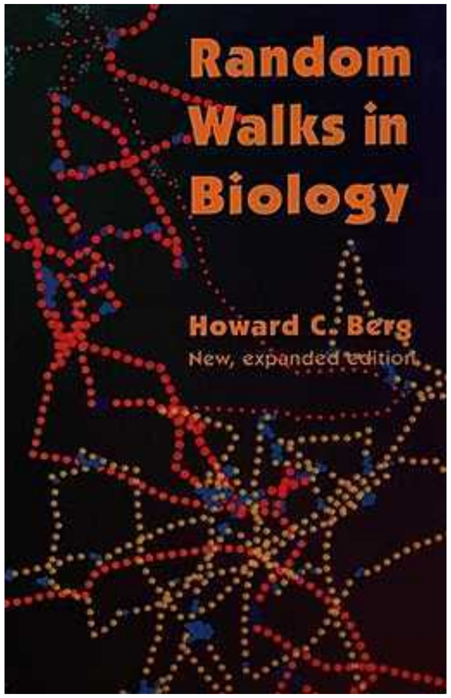

Graduate Texts in Mathematics

Frank Spitzer

Principles of Random Walk

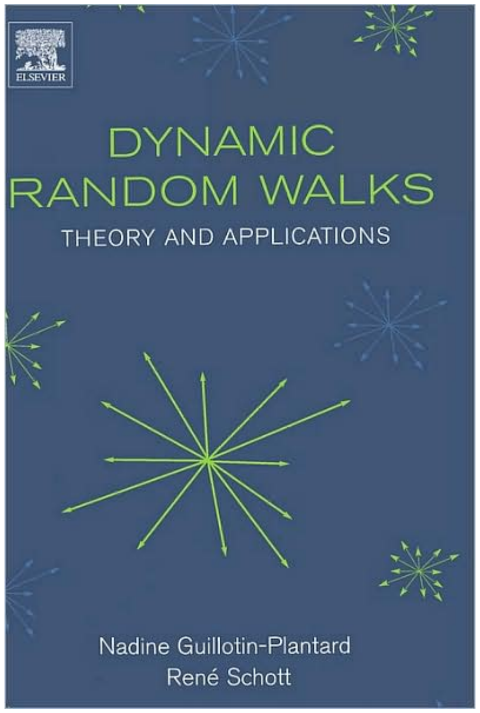
Second Edition

Springer


Random Walks in Biology

Howard C. Berg

New, expanded edition


ELSEVIER

DYNAMIC RANDOM WALKS

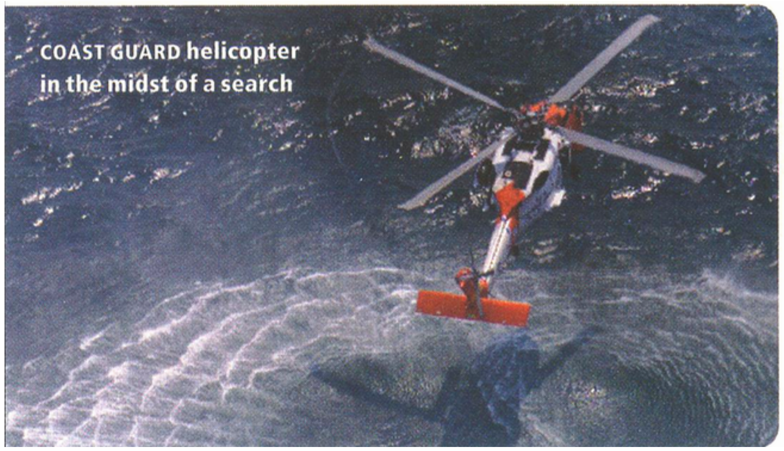
THEORY AND APPLICATIONS

Nadine Guillotin-Plantard
René Schott

SCIENTIFIC AMERICAN

November 2009

How does the Coast Guard find people lost at sea?

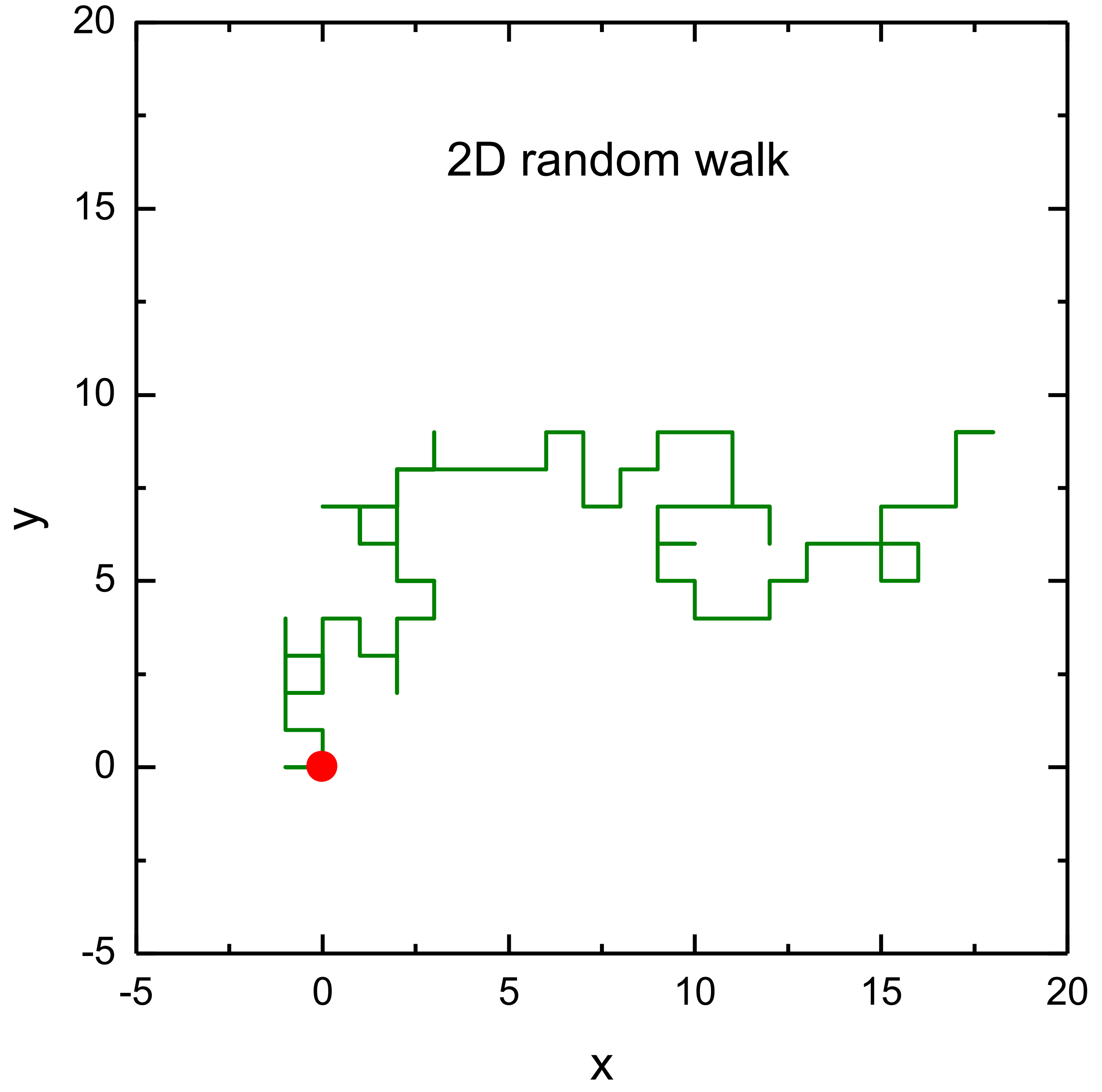


Then, based on that information, we build a strategy with the help of search-planning software called the Search and Rescue Optimal Planning System (SAROPS), which simulates the trajectory of various kinds of objects as they drift. SAROPS is a Monte Carlo–based system that simulates units called particles. Some particles will represent people in the water; others, the boat. They can all start drifting at different times and locales. With SAROPS, we can make more than 10,000 guesses about where boaters got in trouble and when and where they might end up. The program then assesses which scenario is most probable.

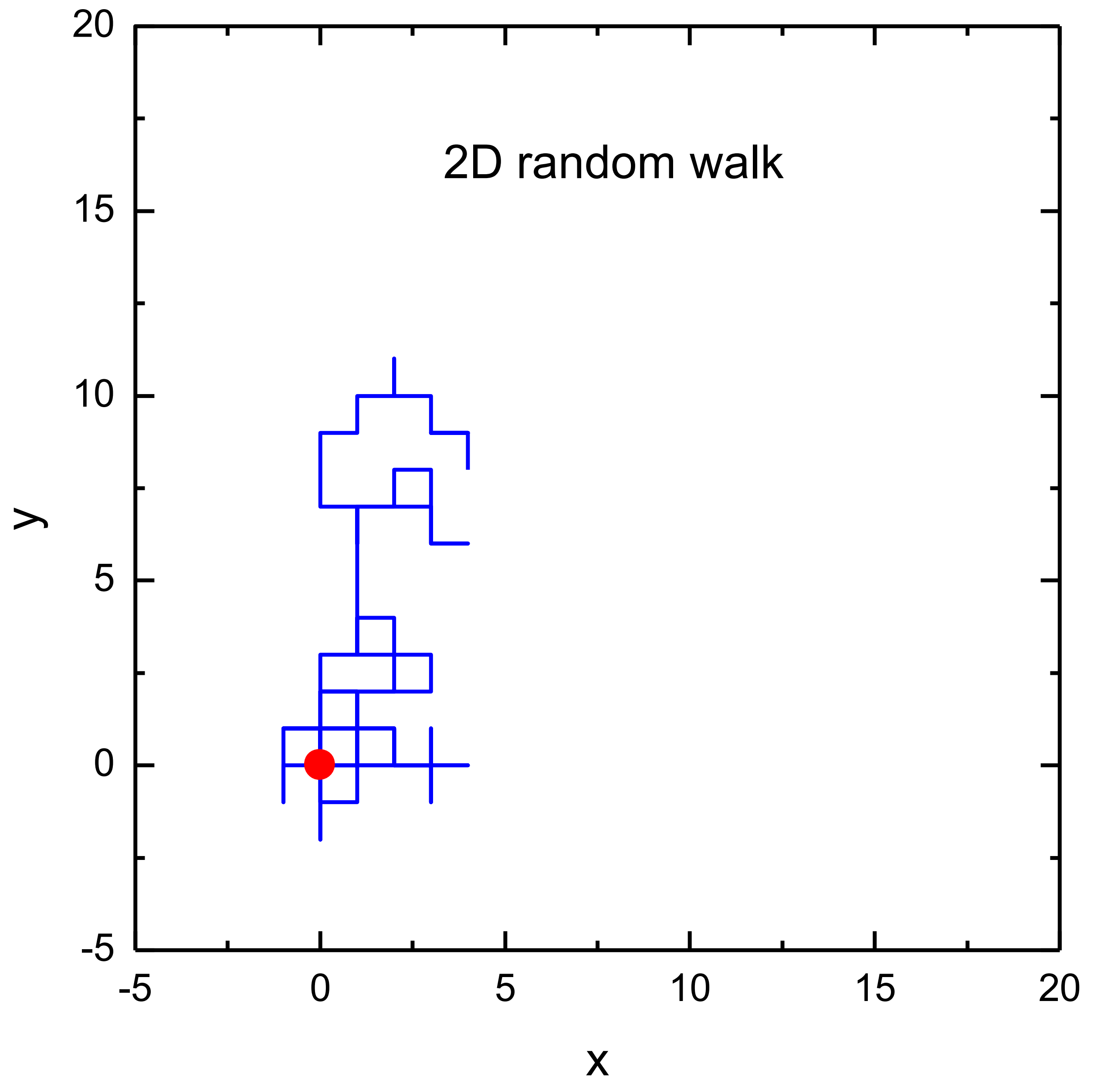
Practice 2 (random walk)

1. Write a program that simulate a random 2D walk with the same step size . Four directions are possible (N, E, S, W). Your program will involve two large integers, M = the number of random walks to be taken and N = the maximum number of steps in a single walk.
2. Find the average distance to be from the origin point after N steps
3. Is there any finite bound on the expected number of steps before the first return to the origin?

example



example



Various models of random walk

Persistent random walk

Restricted random walk

Self-avoiding random walk

…

Examples of applications:

- Spread of inflectional diseases and effects of immunization
- Spreading of fire

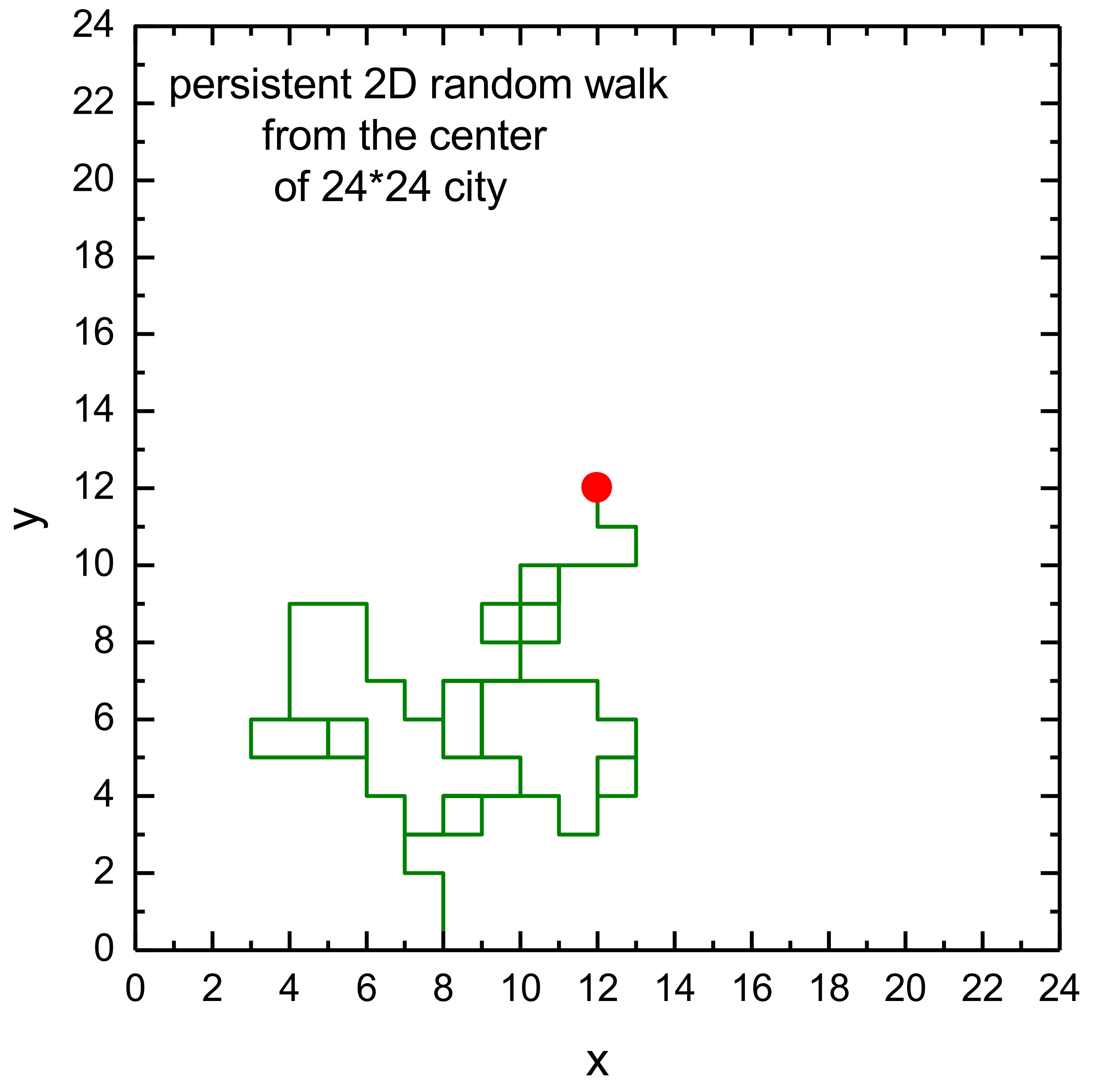
A persistent random walk

A persistent random walk in 2 dimensions in a city with n*n blocks

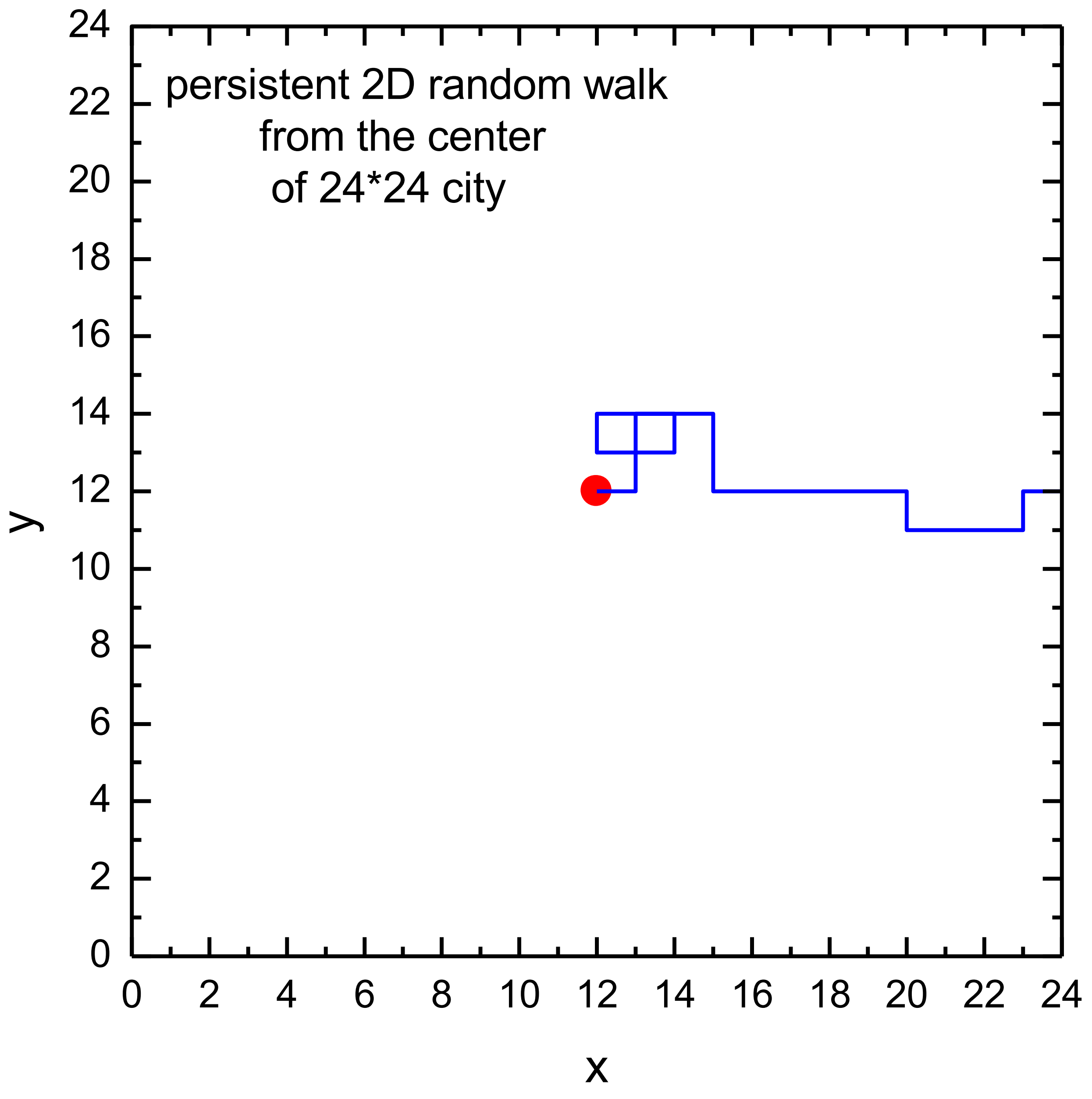
Condition: the walker can not step back

Goal: find average number of steps to get out the city

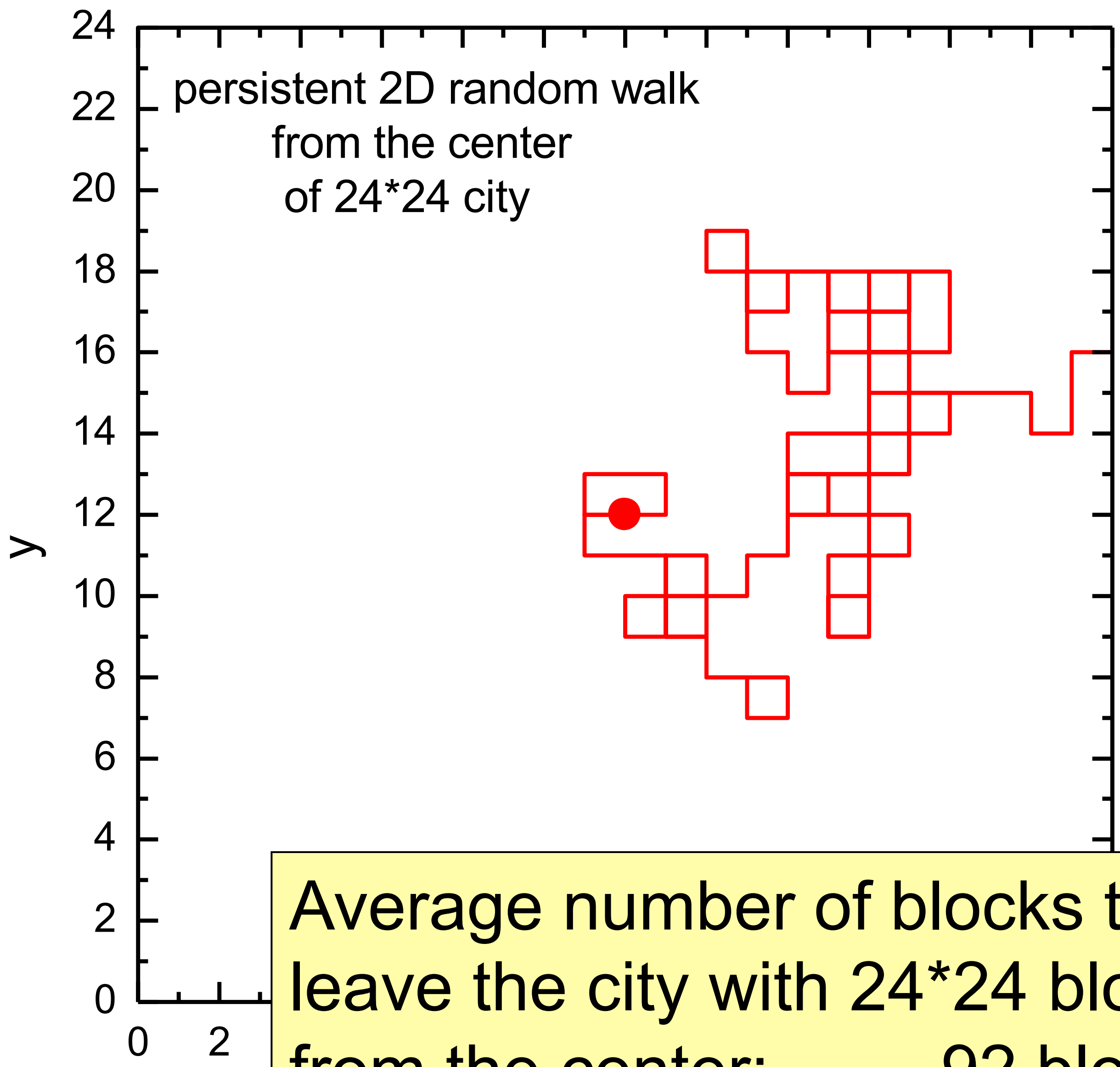
persistent random walk in a city



persistent random walk in a city



persistent random walk in a city



Average number of blocks to go to leave the city with 24*24 blocks

from the center: 92 blocks

from a random point: 47 blocks

The Metropolis algorithm (cont.)

The metropolis sampling is most efficient for multidirectional problems.

In a traditional random walk all visiting points are equal. What is we want the random walker to spend more time in a specific region, e.g. where for a 2D walk g(x,y) is larger.

$$x' = x + h(2u_i - 1)$$
$$y' = y + h(2u_{i+1} - 1)$$

$\Longrightarrow$

then consider

$q = \dfrac{g(x',y')}{g(x,y)}$ and generate

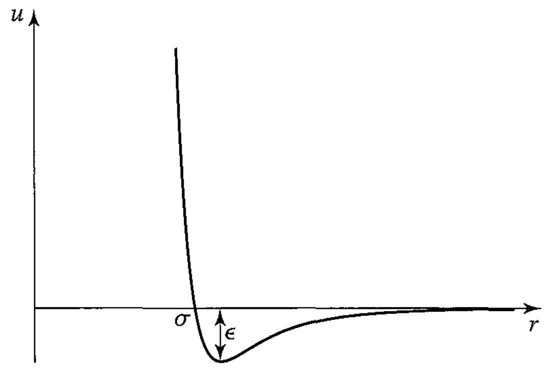
some random number α

if $q \geq \alpha$ the step is accepted

if $q < \alpha$ the step is rejected

Example

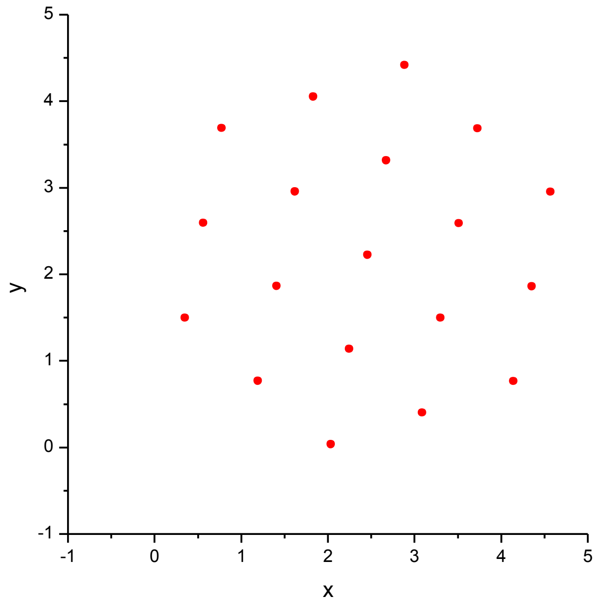
a group of atoms interact by Lennard-Jones potential

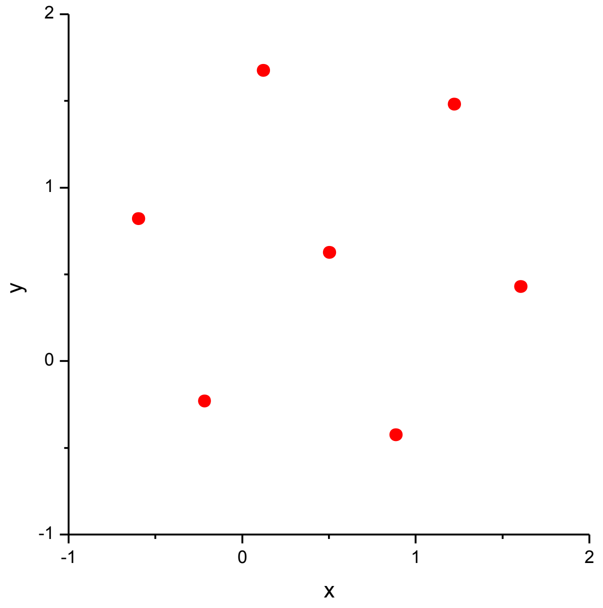


$$V(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$$

Find positions of n atoms that gives the min value of the total potential.
Method: Monte-Carlo variations

examples: n=19 n=7





Example

The French naturalist and mathematician Comte de Buffon showed that the probability that a needle of length L thrown randomly onto a grid of parallel lines with distance D≥L apart intersects a line is 2L/(D*π).

```
c*** loop over trials
      hit = 0
      do it=1,itests
        x0 = float(N)*D*rand()
        k = int(x0/D)
        x1 = x0 - D*float(k)
        x2 = D - x1
        x = min(x1,x2)
        dx = 0.5*abs(L*cos(1.0*pi*rand()))
        if(dx.ge.x) hit = hit + 1
      end do
c*** average number of hits
      ahit = float(hit)/float(itests)
      buffon = (2*L)/(pi*D)
```

```
Buffon problem for D=1
enter numbers of tests
10000
enter numbers of intervals in the grid
10
enter the needle size L<1
0.5
hit    =   3.157E-01
buffon =   3.183E-01
```

```
Buffon problem for D=1
enter numbers of tests
100000
enter numbers of intervals in the grid
50
enter the needle size L<1
0.9
hit    =   5.717E-01
buffon =   5.730E-01
```

Example

investigate a simple problem that generated much attention several years ago and for which many mathematicians obtained an incorrect solution. The problem was the analysis of the optimal strategy in a television game show popular at the time. The show was Let's Make a Deal with host Monty Hall. At some point in the show, a contestant was given a choice of selecting one of three possible items, each concealed behind one of three closed doors. The items varied considerably in value. After the contestant made a choice but before the chosen door was opened, the host, who knew where the most valuable item was, would open one of the doors not selected and reveal a worthless item. The host would then offer to let the contestant select a different door from what was originally selected. The question, of course, is should the contestant switch? A popular magazine writer Marilyn vos Savant concluded that the optimal strategy is to switch. This strategy is counterintuitive to many mathematicians, who would say that there is nothing to be gained by switching; that is, that the probability of improving the selection is 0.5. Study this problem by Monte Carlo methods. What is the probability of improving the selection by switching? Be careful to understand all of the assumptions, and then work the problem analytically also. (A Monte Carlo study is no substitute for analytic study.)

```
 Lets make a deal
 enter numbers of tests
10000
 win1 =   3.359E-01
 win2 =   6.641E-01
```

```
c*** loop over trials
      win1 = 0
      win2 = 0
      do it=1,itests
        a(1) = rand()
        a(2) = rand()
        a(3) = rand()
        choice = 1 + int(3.0*rand())
        b(1) = a(choice)
        if(choice.eq.1) b(2) = max(a(2),a(3))
        if(choice.eq.2) b(2) = max(a(1),a(3))
        if(choice.eq.3) b(2) = max(a(1),a(2))
        if(b(1).ge.b(2)) then
           win1 = win1 + 1
           else
           win2 = win2 + 1
        end if
      end do
c*** average number of games and wins
      awin1 = float(win1)/float(itests)
      awin2 = float(win2)/float(itests)
      write (*,101) awin1, awin2
```

Example

The gambler's ruin problem. Suppose that a person decides to try to increase the amount of money in his/her pocket by participating in some gambling. Initially, the gambler begin with \$m in capital. The gambler decides that he/she will gamble until a certain goal, \$n (n>m), is achieved or there is no money left (credit is not allowed). On each throw of a coin (roll of the dice, etc.) the gambler either win \$1 or lose \$1. If the gambler achieves the goal he/she will stop playing. If the gambler ends up with no money he/she is ruined.

What are chances for the gambler to achieve the goal as a function of k, where k=n/m?

How long on average will it take to play to achieve the goal or to be ruined?

```
 write (*,*)'enter numbers of tests, money and
goal'
      read  (*,*) itests, money1, money2

c*** loop over trials
      total = 0
      wins = 0
      do it=1,itests
        x=money1
        games=0
        do while(x.gt.0.and.x.lt.money2)
           games = games + 1
           luck = 1
           if(rand().le.0.5) luck=-1
           x = x+luck
        end do
        total = total+games
        if(x.gt.0) wins = wins+1
      end do
c*** average number of games and wins
      agames = float(total)/float(itests)
      awins = float(wins)/float(itests)
      aloose = 1.0-awins
      write (*,100) itests, money1, money2
      write (*,101) awins, aloose, agames
 The gambler`s ruin problem.
 Chances to reach certain goal
 enter numbers of tests, money and goal
10000
10
100

 tests:      10000
 initial:       10
 goal:         100
 win   =   1.026E-01
 loose =   8.974E-01
 games =   9.019E+02
```

chance to win in each bet 50/50

```
 The gambler`s ruin problem.
 Chances to reach certain goal
 enter numbers of tests, money and goal
100000
10
100

 tests:     100000
 initial:       10
 goal:         100
 win   = 9.44000E-03
 loose = 9.90560E-01
 games = 4.51806E+02
```

chance to win in each bet 49/51

Applications of Monte-Carlo simulations

- ✓ integration
- ✓ statistical physics
- ✓ aerodynamic
- ✓ quantum chromodynamics
- ✓ molecular dynamic simulation
- ✓ experimental particle physics
- ✓ cellular automata
- ✓ percolation
- ✓ radiation field and energy transport
- ✓ …
- ✓ Finance and business
- ✓ …

Good reference place for Quantum Monte Carlo

http://www.qmcwiki.org/index.php/Research_resources

Cellular automation

Cellular automata – dynamic computational models that are discrete in space, state and time.

Applications – physics, biology, economics, …

Random walk is an example of cellular automata.

see also “The Game of Life” is a cellular automaton devised by John Horton Conway in 1970. Life is an example of emergence and self-organization - complex patterns can emerge from the implementation of very simple rules.