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?

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

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 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

```
00005 10397 34533 78286 35805 15475 68569 50117 81829 74845
00001 37342 04905 64494 79286 28085 48337 20638 18615 05022 81465
00003 86452 84523 45820 12358 12543 50649 87456 75823 32648 76833
00003 99639 07539 09279 72715 36111 21165 86979 72497 64630 71489
00004 02878 98790 85317 04428 66482 68625 94831 14897 12415 74934
00005 68405 74177 36723 74985 36497 35178 65831 39889 11189 26470
00006 36380 13985 49275 63684 82693 62161 89799 71495 24363 38712
00007 85289 77402 02051 66032 69465 78418 73553 92847 19823 69797
00008 63573 26125 06125 67068 95253 57846 28448 27856 85841 56254
00009 73704 47753 03529 84379 59048 32648 69352 52164 35273 94845
00100 93301 17707 14095 88067 92209 40536 68970 36632 50500 73984
00111 11805 05414 39908 27732 50725 42624 79405 66201 52775 67651
00124 01652 98136 04088 39903 17746 70578 16874 86543 68711 77687
00137 88885 63200 38507 58481 36764 67981 95304 74693 29409 11096
00148 99547 67486 87917 64986 91685 89385 97985 61348 23678 46113
```
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)

Two basic techniques for generating uniform random numbers:

1. congruential methods
2. feedback shift register methods.

For each basic technique there are many variations.

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 \( \{x_1, x_2, \ldots, x_M\} \) of length \( M \) over the interval \([0, M-1]\)

\[
x_i = \text{mod}(a x_{i-1} + c, M) = \text{remainder} \left( \frac{a x_{i-1} + c}{M} \right) \quad 0 \leq x_{i-1} < M
\]

- starting value \( x_0 \) is called “seed”
- coefficients \( a \) and \( c \) should be chosen very carefully

Note:

\[
\text{mod}(b, M) = b - \text{int}(b / M) \times M
\]

The method was suggested by D. H. Lehmer in 1948.
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 \(M=2^{31} \approx 2 \times 10^9\))
- Good “magic” number for linear congruent method:
  \[ x_i = \text{mod}(ax_{i-1} + c, M) \]
- \(a = 16,807, c = 0, M = 2,147,483,647\)
  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

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
  \[ \langle x^k \rangle = \frac{1}{N} \sum_{i=1}^{N} x_i^k - \frac{1}{k+1} \]

- Near-neighbor correlation
  \[ \frac{1}{N} \sum_{i=1}^{N} x_ix_{i+k} = \frac{1}{4} \]

---

**Test Suites (most known) for RNG**

- the NIST Test Suite (NIST, 2000) includes sixteen tests

- "DIEHARD Battery of Tests of Randomness (eighteen tests)
  [http://www.stat.fsu.edu/pub/diehard/](http://www.stat.fsu.edu/pub/diehard/)

- TestU01: includes the tests from DIEHARD and NIST and several other tests that uncover problems in some generators that pass DIEHARD and NIST
  [http://www.iro.umontreal.ca/~simardr/testu01/tu01.html](http://www.iro.umontreal.ca/~simardr/testu01/tu01.html)

---

**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/](http://lib.stat.cmu.edu/)
- IMSL (International Mathematics and Statistics Library) libraries contain a large number of RNGs

---

**"Industrial" methods in C/C++ and Fortran**

- `rand` 1. call **SEED**
  Changes the starting point of the pseudorandom number generator.

- `random` 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++**

```cpp
// generate integer random numbers between i1 and i2
#include <cstdlib>
#include <iostream>
#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;
}
```

---

**Example: srand and rand in C++**

```cpp
// generate integer random numbers between i1 and i2
#include <cstdlib>
#include <iostream>
#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;
}
```
/* 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 numbers */
    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;
}
Standard RNG in Fortran

srand(iseed) seeds the random number generator
rand() Return real random numbers in the range 0.0 through 1.0.
Generating a real random number between 0.0 and 1.0

call srand(iseed)
x = rand()

There are very many good uniform and non-uniform random number generators written in Fortran.

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

- 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^{63} 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^2+y^2) < 1 then m=m+1
4. since A_{circle}/A_{square} = m/N
5. A_{circle} = m/N*A_{square} = (m/N)*4R^2
Compute $N$ pairs of random numbers $x_i$ and $y_i$ with $0.0 \leq x \leq 2.0$ and $-1.5 \leq y \leq 1.5$.

$$F_n = A \left( \frac{n_r - n_l}{N} \right)$$

Integration by mean value

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

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

$$\Delta S = (b-a)\sqrt{\frac{\sum_{i=1}^{N} (f_i - \left\langle f \right\rangle)^2}{N}}$$

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

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

### Example: 1D integration (C++)

```c
#include <cstdlib>
#include <ctime>

double int_mc1d(double(*f)(double), double a, double b, int n)
/* 1D intergration using Monte-Carlo method for f(x) on [a,b] */
{ 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: $\int \sin(x)dx = 2.0$

<table>
<thead>
<tr>
<th>$n$</th>
<th>Trapez.</th>
<th>Simpson</th>
<th>Monte Carlo</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.570796</td>
<td>2.094395</td>
<td>2.483686</td>
</tr>
<tr>
<td>4</td>
<td>1.896119</td>
<td>2.004560</td>
<td>2.570860</td>
</tr>
<tr>
<td>8</td>
<td>1.9743232</td>
<td>2.000269</td>
<td>2.140117</td>
</tr>
<tr>
<td>16</td>
<td>1.993570</td>
<td>2.000017</td>
<td>1.994455</td>
</tr>
<tr>
<td>32</td>
<td>1.999598</td>
<td>2.000001</td>
<td>2.005999</td>
</tr>
<tr>
<td>64</td>
<td>1.999900</td>
<td>2.000000</td>
<td>2.000751</td>
</tr>
<tr>
<td>128</td>
<td>1.999999</td>
<td>2.000000</td>
<td>2.000001</td>
</tr>
<tr>
<td>256</td>
<td>1.999999</td>
<td>2.000000</td>
<td>2.000000</td>
</tr>
<tr>
<td>512</td>
<td>1.999999</td>
<td>2.000000</td>
<td>2.000000</td>
</tr>
<tr>
<td>1024</td>
<td>1.999999</td>
<td>2.000000</td>
<td>2.000000</td>
</tr>
<tr>
<td>2048</td>
<td>1.999999</td>
<td>2.000000</td>
<td>2.000000</td>
</tr>
<tr>
<td>4096</td>
<td>1.999999</td>
<td>2.000000</td>
<td>2.000000</td>
</tr>
<tr>
<td>8192</td>
<td>1.999999</td>
<td>2.000000</td>
<td>2.000000</td>
</tr>
<tr>
<td>16384</td>
<td>1.999999</td>
<td>2.000000</td>
<td>1.999999</td>
</tr>
<tr>
<td>32768</td>
<td>1.999999</td>
<td>2.000000</td>
<td>1.999999</td>
</tr>
<tr>
<td>65536</td>
<td>1.999999</td>
<td>2.000000</td>
<td>1.999999</td>
</tr>
</tbody>
</table>

### Example: $\int \frac{x}{x^2 + 1}cos(10x^2)dx = 0.0003156$

<table>
<thead>
<tr>
<th>$n$</th>
<th>Trapez.</th>
<th>Simpson</th>
<th>Monte Carlo</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>0.004360</td>
<td>-0.013151</td>
<td>0.081207</td>
</tr>
<tr>
<td>128</td>
<td>0.001813</td>
<td>-0.001110</td>
<td>0.155946</td>
</tr>
<tr>
<td>256</td>
<td>0.000526</td>
<td>-0.000311</td>
<td>0.071404</td>
</tr>
<tr>
<td>512</td>
<td>0.000036</td>
<td>0.000006</td>
<td>0.002110</td>
</tr>
<tr>
<td>1024</td>
<td>0.000032</td>
<td>0.000016</td>
<td>-0.004525</td>
</tr>
<tr>
<td>2048</td>
<td>0.000031</td>
<td>0.000028</td>
<td>-0.010671</td>
</tr>
<tr>
<td>4096</td>
<td>0.000031</td>
<td>0.000038</td>
<td>-0.016871</td>
</tr>
<tr>
<td>8192</td>
<td>0.000031</td>
<td>0.000046</td>
<td>-0.023071</td>
</tr>
<tr>
<td>16384</td>
<td>0.000031</td>
<td>0.000054</td>
<td>-0.029271</td>
</tr>
<tr>
<td>32768</td>
<td>0.000031</td>
<td>0.000062</td>
<td>-0.035471</td>
</tr>
<tr>
<td>65536</td>
<td>0.000031</td>
<td>0.000070</td>
<td>-0.041671</td>
</tr>
</tbody>
</table>

### Example: antithetic variates – using "mirror points"

$$I = \int_{a}^{b} f(x)dx = (b-a)\left\langle f \right\rangle + \frac{1}{N} \sum_{i=1}^{N} \left( f(x_i) + f(a + b - x_i) \right)$$

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

\[
\int_{a}^{b} \int_{c}^{d} f(x, y) \, dx \, dy = (b - a)(d - c) \frac{1}{N} \sum_{i=1}^{N} f(x_i, y_i)
\]

**Example: nD integration (C++)**

```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;
}
```

**Practice: Integration**

- Use Monte Carlo integration (both rejection and mean value methods) to evaluate
  \[ \int _0^1 e^{-x} \, dx \] and \[ \int _0^1 \sin(2x) \, dx \]
- Evaluate 7-D integral
  \[
  \int_{x_0}^{x_1} \int_{x_0}^{x_2} \int_{x_0}^{x_3} \int_{x_0}^{x_4} \int_{x_0}^{x_5} \int_{x_0}^{x_6} (x_1 + x_2 + \ldots + x_7)^2 \, dx_1 \, dx_2 \, dx_3 \, dx_4 \, dx_5 \, dx_6 \, dx_7
  \]

**Part 3**

Non-uniform distributions
Non-uniform distributions

Most situations in physics – random numbers with non-uniform distribution
  - radioactive decay
  - experiments with different types of distributions
  - ...

Principal idea: Generating non-uniform random number distributions with a uniform random number generators

Example for \( w(x) = \exp(-x^2) \)

```c
double w(double); int main () {
  int nmax = 50000;
  double xmin=0.0, xmax=2.0, ymin, ymax;
  Double w, y;
  ymin = w(xmin);
  ymax = w(xmax);
  srand(time(NULL));
  for (double i=1; i <= nmax; i=i+1)
    {
      x = xmin + (xmax-xmin)*rand()/(RAND_MAX+1);
      y = ymin + (ymax-ymin)*rand()/(RAND_MAX+1);
      if (y > w(x)) continue;
      file_3  << " " << x << endl; /* output to a file */
    }
  system("pause");
  return 0;
}

/* Probability distribution \( w(x) \) */
double w(double x)
  {
    return exp(0.0-1.0*x*x);
  }
```

Example

Example

Method 1: von Neumann rejection

Generating non-uniform distribution with a probability distribution \( w(x) \)

- generate two random numbers
  - \( x \) on \([x_{min}, x_{max}]\)
  - \( y \) on \([y_{min}, y_{max}]\)
- if \( y < w(x) \), accept
- if \( y > w(x) \), reject
- The \( x \) so accepted will have the weighting \( w(x) \)

Method 2: Inversion method

Works if the function you are trying to use for a distribution has an inverse.

Let \( p(y) = f(y) \) is some arbitrary distribution

From the fundamental transformation law of probabilities

\[
[p(y)dy] = [p(x)dx] \quad \text{or} \quad p(y) = p(x)\frac{dx}{dy}
\]

Let \( p(x) \) is a uniform probability distribution

We need to solve the differential equation

\[
\frac{dx}{dy} = f(y) \quad \Rightarrow \quad x = \int f(y)dy = F(y)
\]

then \( y = F^{-1}(x) \) provides the non-uniform distribution
Example: exponential distribution

\[ p(y) = \exp(-y) \]
\[ \frac{dx}{dy} = \exp(-y) \]
\[ x = \int \exp(-y)dy \rightarrow \exp(-y) \quad \text{(take positive distribution)} \]
\[ y = -\ln(x) \]
y is a positive exponential distribution generated from a uniform distribution \( x \)

Quite often analytic solutions are not feasible. However, very many program libraries have most common non-uniform distributions

**Practice: non-uniform distribution**

Use the von Neumann rejection technique to generate a normal distribution of standard deviation 1.0

more on integration – importance of sampling

or more attention to regions corresponding to large values of the integrand

\[ I = \frac{1}{N} \sum_{i=1}^{N} f(x_i) \]

where \( p(x) \) is a probability density over \( x \)

The density \( p(x) \) is called the importance function

Then with \( x_i \) from the distribution with density \( p \)

\[ I = (b-a) \frac{1}{N} \sum_{i=1}^{N} f(x_i) \]

**The Metropolis algorithm**

In 1953 Metropolis introduced “the idea of importance sampling” that can considerably improve speed and quality of calculations.

\[ I = \frac{1}{N} \sum_{i=1}^{N} w(x_i) f(x_i) \]

In the simplest version, \( x_{i+1} = x_i + h(2u - 1) \) where \( h \) is a step and \( u \) is from a uniform random distribution

The step is accepted if

\[ w(x_{i+1}) \geq \alpha_i \]

where \( \alpha_i \) is a random number from a uniform distribution

**Part 4**

**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, ...

---

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 (or are carried by tides or currents). SAROPS uses a graph to model the search area and can rapidly search through long ranges of time. They can all start drifting at different times and locations. With SAROPS, we can make more than 10,000 guesses about where boaters got in trouble and when and where they might end up.

---

Practice 2 (random walk)

1. Write a program that simulates 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: random walk (Fortran)

```fortran
integer*4 iu, it, is, itests, isteps, iway, x, y
real*4 rand, d, dav

read (*,*) itests, isteps
    dav=0.0
    do it=1,itests
        x=0
        y=0
        do is=1,isteps
            iway= int(0.0+4.0*rand())
            if(iway.eq.0) x = x+1
            if(iway.eq.1) x = x-1
            if(iway.eq.2) y = y+1
            if(iway.eq.3) y = y-1
            c    write(7,101) x,y
        end do
        d = sqrt((float(x))**2+(float(y))**2)
        dav = dav + d
    end do
    dav = dav/float(itests)
    write(*,100) itests, isteps, dav
```

---

Example: 2D random walk

![Example: 2D random walk](image)
Various models of random walk
Persistent random walk
Restricted random walk
Self-avoiding random walk

Examples of applications:
- Spread of infecional 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

do while (out.eq.0)
skip=0
iway = int(1.0+4.0*rand())
c check can we use this iway or not
if(abs(iway - iold).eq.2) skip=1
c if step is allowed the walker goes
if(skip.eq.0) then
   nsteps = nsteps+1
   if(iway.eq.1) x = x+1
   if(iway.eq.3) x = x-1
   if(iway.eq.2) y = y+1
   if(iway.eq.4) y = y-1
   c check conditions to be out of n*n city
   if(x.le.0.or.x.ge.ncity) out=1
   if(y.le.0.or.y.ge.ncity) out=1
   iold=iway
end if
end do

Example: (Fortran)

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
Polymer simulation

Random walk in 2 dimensions
Condition: self-avoiding 2D random walk

Example: (from Fortran)

```fortran
    do while (out.eq.0)
        xold=x
        yold=y
        c = rand()
        if(c.le.0.25) x = x+1
        if(c.gt.0.25.and.c.le.0.50) x=x-1
        if(c.gt.0.50.and.c.le.0.75) y=y+1
        if(c.gt.0.75) y=y-1
        if(a(x,y).eq.0) then
            a(x,y)=1
            n = n+1
        else
            x=xold
            y=yold
        end if
    end do
```

---

Polymer simulation

- Average polymer size: 72
- End-to-end size: 12

---
The Metropolis algorithm (cont.)

The metropolis sampling is most efficient for multidirectional problems.

In a traditional random walk all visiting points are equal. What do 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.

\[
x' = x + k(2a_i - 1) \quad \text{then consider} \quad q = \frac{g(x',y')}{g(x,y)}
\]

and generate some random number $\alpha$ if $q \geq \alpha$ the step is accepted if $q < \alpha$ the step is rejected.

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 $D$ apart intersects a line is $\frac{2L}{\pi D}$.

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

Example

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.)
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=m/n?

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

Example
An industrious physics major finds a job at a local fast food restaurant to help him pay his way through college. His task is to cook 20 hamburgers on a grill at any one time. When a hamburger is cooked, he is supposed to replace it with uncooked hamburger. However, our physics major does not pay attention to whether the hamburger is cooked or not. His method is to choose a hamburger at random and replace it by an uncooked one. He does not check if the hamburger that he removes from the grill is ready.

What is the distribution of cooking times of the hamburgers that he removes?
What is a chance for a customer to get a well cooked hamburger if it takes 5 minutes to cook a hamburger.

Does the answers to the first two questions change if he cooks 40 hamburgers at any one time?

Comment: For simplicity, assume that he replaces a hamburger at a regular interval of 30 seconds and there is an indefinite supply of uncooked hamburgers.
nburger = 20
tsteps = 100000
bmin = 10
bmax = 20
c initialization (burgers)
do i = 1, nburger
    b(i) = 0
end do
c initialization (time distribution)
do i = 1, tsteps
    tcook(i) = 0
end do
c other variables
    cook1 = 0
    cook2 = 0
    cook3 = 0
    tmax = 0
c initial seed number calculated from current time
    call gettim(ihour,imin,isec,imsec)
    init = imsec*isec*imin+ihour
    call srand(init)
do i = 1, tsteps
    x = 1.0 + rand()*real(nburger)
    burger = int(x)
do j = 1, nburger
        if(j.eq.burger) then
            tcook(b(j)+1) = tcook(b(j)+1) + 1
            if(b(j)+1.gt.tmax) tmax = b(j)+1
            if(b(j).lt.bmin) cook1 = cook1 + 1
            if(b(j).ge.bmin.and.b(j).le.bmax) cook2 = cook2 + 1
            if(b(j).gt.bmax) cook3 = cook3 + 1
            b(j) = 0
        else
            b(j) = b(j) + 1
        end if
    end do
do i = 1, tsteps
    x = 1.0 + rand()*real(nburger)
    burger = int(x)
do j = 1, nburger
        if(j.eq.burger) then
            tcook(b(j)+1) = tcook(b(j)+1) + 1
            if(b(j)+1.gt.tmax) tmax = b(j)+1
            if(b(j).lt.bmin) cook1 = cook1 + 1
            if(b(j).ge.bmin.and.b(j).le.bmax) cook2 = cook2 + 1
            if(b(j).gt.bmax) cook3 = cook3 + 1
            b(j) = 0
        else
            b(j) = b(j) + 1
        end if
    end do
end if
write(*,102) tmax
write(*,101) rcook1, rcook2, rcook3
for 100,000 burgers
20 burgers on the grill
max cooking time = 237
undercooked = 0.39941001
well cooked = 0.25903001
over cooked = 0.34156001
40 burgers on the grill
max cooking time = 463
undercooked = 0.22596000
well cooked = 0.18769000
over cooked = 0.58635002
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
- ...
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.

What is a chance to encounter a bear?
What data do you need for your simulation?