Ruletka dla fizyka

(wprowadzenie do technik Monte-Carlo)

Jerzy Karczmarczuk
Zakład Informatyki, Uniwersytet w Caen, Francja

Generatory liczb losowych

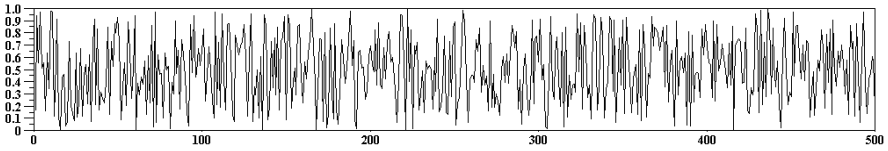
Aby modelować obiekty i procesy należące do świata rzeczywistego, komputer musi umieć generować zjawiska losowe, niepowtarzalne (z praktycznego punktu widzenia), gdyż Natura jest właśnie taka. Jest to oczywiście niezbędne w fizyce statystycznej czy w symulacji procesów jądrowych. Jednak losowe, albo raczej *pseudolosowe*, ciągi liczb znajdują zastosowanie także w rozmaitych algorytmach całkowicie deterministycznych, np. aby obliczyć objętość skomplikowanego obiektu, czasami wygodnie jest zapełnić go dość gęsto losowo wybranymi punktami; wtedy, przy zadanej średniej gęstości tych punktów, ich liczba pozwala oszacować wynik. Mamy nadzieję, że te losowe punkty rozłożą się równomiernie wewnątrz szacowanego obiektu. Wynik jest naturalnie obarczony błędem, ale często interesuje nas jakieś przybliżenie, a może ono nie być gorsze niż otrzymane innymi metodami, o ile ocena „gorsze/lepsze" bierze pod uwagę np. stosunek dokładności do kosztów obliczeniowych. Innym przykładem jest szukanie minimum funkcji wielu zmiennych przez wybór losowy (choć nie „ślepy") punktów-kandydatów. Jest to użyteczne w szukaniu punktów równowagi złożonych układów fizycznych. Wiemy, że układy termodynamiczne dążą do maksimum entropii, układy mechaniczne minimalizują tzw. całki działania itp. Są to dwa z licznych przykładów technik zwanych popularnie Monte-Carlo. Jak jednak wygenerować te losowo wybrane liczby czy punkty? W zasadzie można posłużyć się jakimś układem fizycznym, np. elektroniką generującą szum, substancją radioaktywną itp., ale dla fizyka istotna jest często możliwość *dokładnego* powtórzenia eksperymentu lub obliczeń, a prawdziwe układy losowe nie dają się kontrolować...

Tak więc, w typowych zastosowaniach, w generacji ciągów liczb losowych, wbrew nazwie nie ma *nic* losowego, algorytmy są całkowicie określone i powtarzalne. Nie używa się też z góry przygotowanych tablic liczb losowych. Tablice oraz pomiary fizyczne, np. odczyt zegara komputera, mogą służyć do wstępnego ustawienia generatorów, ale reszta jest z góry określona. Będące w codziennym użyciu algorytmy komputerowe posługują się ścisłą matematyką. Typowy generator jest zwykłą funkcją, która pobiera jako argument ostatnio wygenerowaną liczbę pseudolosową i dostarcza nową wartość. Funkcja jest tak „dziwaczna", że ten nowy wynik *wydaje się* nie mieć nic wspólnego z poprzednim, a kolejne wartości są równomiernie rozrzucone w jakimś przedziale.

Matematyczna precyzja stanowi właśnie siłę takich generatorów, które są oparte na teorii pozwalającej wykazać, że generator dostarczy np. $M = 2^{32} - 1$ niepowtarzających się liczb całkowitych między 0 a M, a następnie powtórzy cały ciąg. Oczywiście w użyciu są generatory znacznie lepsze... Profesjonalne generatory pracują zwykle na liczbach całkowitych między 0 a $M - 1$, gdzie M jest bardzo dużą liczbą, np. równą wspomnianym 2^{32} lub znacznie większą. Tak więc wszystko wygląda losowo, ale nie spotkają nas żadne nieprzyjemne niespodzianki.

Ułamek w zakresie $[a, b)$ otrzymuje się, dzieląc wynik całkowity przez M, a następnie mnożąc przez $b - a$ i dodając a. W podstawowych algorytmach generacji lepiej nie używać bezpośrednio liczb ułamkowych (zmiennoprzecinkowych), gdyż jest to wolniejsze, niepewne teoretycznie i zależne od komputera.

Jednak prosty, niedoskonały, ale wystarczający do prostych testów gene rator można skonstruować samemu, wystarczy wielokrotnie zastosować szybkozmienną, „mieszającą" funkcję do jakiejś wartości rzeczywistej. Przyjmijmy np. $x_0 = 0.16524961$, albo inną wartość między 0 a 1, oraz dość duży mnożnik, np. $R = 17652083.651$, a następnie powtarzajmy: $x_{n+1} = \mathrm{frac}(R \cdot x_n)$, gdzie frac oznacza część ułamkową liczby rzeczywistej, $\mathrm{frac}(z) \equiv z - [z]$. Nasz model zilustrowany jako funkcja x_n dla 500 wartości n przedstawia się następująco:



Generujemy w ten sposób liczby rozłożone *jednorodnie* w pewnym przedziale, liczby, których rozkład prawdopodobieństwa jest funkcją stałą. Ale fizyk potrzebuje i innych rozkładów, zgodnych z rozkładami niejednorodnymi. Np. jeśli chcemy utworzyć model gazu, często korzystamy z rozkładu Gibbsa, mówiącego, że prawdopodobieństwo stanu mikroskopowego, tj. zbioru położeń i pędów wszystkich cząstek $s(x, p)$, jest proporcjonalne do $e^{E(s)/kT}$, gdzie k jest stałą Boltzmanna: $1.3807 \cdot 10^{-23} J/K$, T oznacza temperaturę, a $E(s)$ jest energią danego stanu: sumą energii kinetycznych i ewentualnie, przy obecności oddziaływań, także sumą energii potencjalnych.

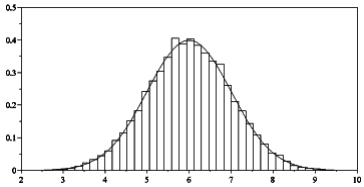
Nie miejsce tu, aby omówić szczegóły generowania liczb o dowolnych rozkładach, wrócimy do tego kiedy indziej, ale w powyższym przykładzie metoda jest prosta. Aby wygenerować liczbę losową x o rozkładzie prawdopodobieństwa $p(x) = e^{-x}$ między 0 a ∞, wystarczy wziąć losową wartość r z jednorodnego rozkładu między 0 a 1, a następnie utworzyć $x = \ln(1 - r)$. W ogólności zwykle zaczynamy od rozkładu jednorodnego, a potem kombinujemy i transformujemy wyniki. W ten sposób będziemy również generować *funkcje losowe*, np. szumy,

losowe trajektorie cząstek płynu itp. Zazwyczaj jest to o wiele trudniejsze niż generowanie pojedynczych liczb losowych, gdyż w funkcji losowej sąsiednie wartości są zwykle mocno skorelowane, np. zbliżone, więc to, co wygenerowaliśmy dla x, wpływa na wartości generowane dla $x + \Delta x$.

Często potrzebujemy również *liczb gaussowskich*, losowych między $-\infty$ a ∞, z rozkładem $p(x) = \exp\left((x - x_s)^2/2\sigma^2\right)/\sqrt{2\pi}\sigma$. Jak wiemy, tak zwykle są rozłożone wartości doświadczalne x wokół średniej x_s, z wariancją σ, więc takie rozkłady są potrzebne np. w symulacji aparatury pomiarowej, która dostarcza wyników z błędami statystycznymi.

Prosty, ale często wystarczający algorytm generacji jest małą perełką obliczeniową. Zaczniemy od pytania: dlaczego zwykle fluktuacje wartości wokół średniej mają kształt krzywej dzwonowej Gaussa? Okazuje się, z tzw. centralnego twierdzenia granicznego, że jeśli pewna zmienna losowa jest ***sumą***, wynikiem nałożenia się wielu składników o rozsądnych, ale dowolnych rozkładach prawdopodobieństwa – co ma zwykle miejsce w procesach naturalnych i technologicznych – to przy dostatecznej liczbie składników wynik jest z grubsza gaussowski, znacznie więcej próbek wykazuje małe odchylenia od średniej niż duże, co jest intuicyjnie oczywiste. Możemy ten fakt wykorzystać do symulacji. Wygenerujemy kilkadziesiąt lub nawet tylko kilkanaście, np. 12 (wygodne ze względu na normalizację, wtedy $\sigma = 1$), liczb losowych jednorodnych w [0, 1) i obliczymy ich sumę. Wynik będzie gaussowski wokół 6. Sprawdźmy to za pomocą programu rysującego histogram 20 000 liczb losowych oraz jego sprawdzenie: krzywa Gaussa. Zgodność jest doskonała. Program został napisany w polecanym przez nas w zeszłym odcinku języku pakietu Scilab, którego zaletą jest możliwość pisania krótkich i czytelnych programów operujących na tablicach.

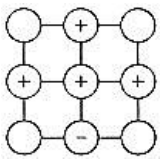
```
R=rand(12,20000);
histplot(40,sum(R,'r'));
x=0:0.1:12; u = x - 6;
plot2d(x,exp(-u.*u/2)/sqrt(2*%pi),
       strf="000");
```



Szczegóły programu są mało istotne, wystarczy wiedzieć, że `rand(m,n)` generuje tablice $m \times n$ jednorodnych liczb losowych, że `sum` pozwala wysumować tablice po wszystkich wierszach (albo kolumnach), a `plot2d` rysuje wykres dwuwymiarowy o określonych parametrach.

Przykład: przejścia fazowe w modelu ferromagnetyka

Przećwiczmy konkretne zastosowanie generacji liczb losowych w fizyce. Zajmiemy się tzw. modelem Isinga zjawiska magnetyzacji spontanicznej, napiszemy program symulacyjny i pokażemy, jakiego rodzaju „chwytów" dopuszcza się fizyk, wykorzystujący metody Monte-Carlo. Celem naszym jest wizualizacja, a także nieskomplikowane *pomiary* własności układu. Nigdy nie będziemy tracić z oczu różnic między podejściem algorytmicznym a procesem realnym. Symulowanym układem jest siatka (tu: dwuwymiarowa, można jednak prosto rozwinąć model w trzech wymiarach), w węzłach której znajdują się oddziałujące „magnesy", mogące przyjmować dwie wartości lub ustawienia: „w górę" lub „w dół". Ten model dość realistycznie odpowiada rzeczywistemu magnetykowi, zawierającemu oddziałujące kwantowe spiny.



Reprezentacją modelu jest tablica wypełniona liczbami $s = \pm 1$. Wartość +1 oznacza moment magnetyczny (spin) skierowany w górę, –1 – w dół. Sąsiednie spiny „lubią" ustawiać się równolegle, gdyż to zmniejsza energię oddziaływania. W tym modelu energia pary sąsiednich spinów wyraża się wzorem $E_{ij} = -J/2 \cdot s_i s_j$. J jest dowolną stałą określającą siłę oddziaływania, możemy ją przyjąć, równą 1, zresztą istotny będzie tylko iloraz J/kT. Stałą Boltzmanna również przyjmujemy równą jedności. W symulacjach komputerowych nie ma zwykle powodu, aby wszystkie stałe fizyczne miały swoje prawdziwe wartości, natomiast ważne jest, aby umieć przetransponować wyniki symulacji do świata rzeczywistego. Jest to osobny temat do dyskusji.

Na rysunku powyżej wkład spinu centralnego do energii oddziaływania jest równy $–3 + 1 = –2$. Spiny diagonalne, dla prostoty, nie liczą się jako najbliżsi sąsiedzi, ich wprowadzenie zmienia własności układu w nieznacznym stopniu. W ogóle najważniejszym parametrem układu jest jego *wymiar*. Spiny „nanizane na nitkę", tj. tworzące układ jednowymiarowy, nie wykazują żadnych ciekawych zachowań. W dwóch i więcej wymiarach układ może się spontanicznie magnesować w temperaturze niższej niż tzw. punkt Curie. Energia całości jest sumą po wszystkich węzłach energii jednostkowych, a prawdopodobieństwo danego stanu jest dane rozkładem Gibbsa: $p(\mathbf{s}) = 1/Z \cdot \exp(-\Sigma s_i s_j/T)$, gdzie Z jest stałą normalizującą prawdopodobieństwo, a **s** jest oznaczeniem stanu: ciągu wszystkich s_i. Do symulacji często używa się następującego algorytmu (Metropolis i inni), który wielokrotnie iteruje następujące akcje:

- Każdemu spinowi „proponujemy" nową wartość, losowaną spośród możliwych. Tutaj: dwie możliwości.
- Jeśli zmieniona energia byłaby mniejsza od wyjściowej, program akceptuje zmianę, a układ przechodzi do nowego stanu.

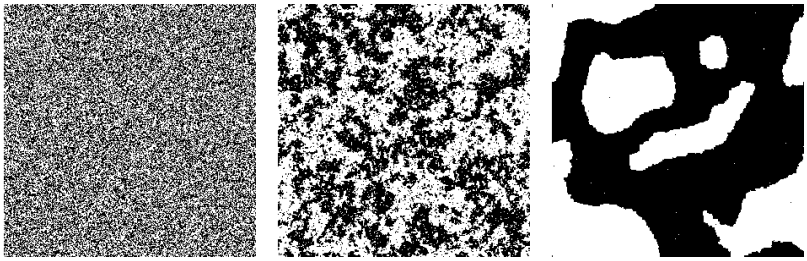
- Jeśli to spowodowałoby zwiększenie energii o ΔE, obliczamy wartość $q = \exp(-\Delta E/T)$ i generujemy liczbę losową r między 0 a 1. Jeśli $r < q$, zmiana zostaje przyjęta, jeśli $r \geq q$, spin pozostaje ze starą wartością.

Zauważamy, że w wysokich temperaturach q jest prawie równe jeden i prawie wszystkie propozycje zostaną akceptowane. Fluktuacje termiczne sprzyjają bałaganowi i oddziaływanie między spinami nie jest istotne. Gdy temperatura się zmniejsza, układ będzie widocznie preferował konfiguracje, w których spiny sąsiednie będą równoległe. Układ będzie „wolał" zmniejszać swoją energię przez porządkowanie sąsiednich spinów. Zaczną się pojawiać domeny magnetyczne. Możemy ustalić temperaturę i puścić symulację w „ruch", ale „czas", w jakim przebiega odwracanie spinów, jest artefaktem, nie ma w zasadzie nic wspólnego z rzeczywistym czasem, a co interesuje fizyka, to konfiguracja równowagowa, po długim okresie symulacji, gdy konfiguracja się ustali (w sensie ogólnym tego słowa, w wysokich temperaturach stan może się „ustalić" jako szybkozmienny bałagan...). Osobę zainteresowaną zagadnieniami symulacji i wizualizacji sam proces ewolucji układu może bardzo interesować, przy zachowaniu sporej ostrożności pewien związek między dynamiką symulacji a rzeczywistym dochodzeniem do stanu równowagi jednak można zaobserwować, ale chwilowo nie jest to dla nas ważne.

Możemy zauważyć tutaj pewien konflikt między realizmem symulowanego procesu, pożytecznym przy wizualizacji, a jego sprawnością. Fizyk zainteresowany pomiarami magnetyzacji, ciepła właściwego itp. a nie obrazkami, zauważy, że nie ma sensu proponować spinowi jego poprzedniej wartości, gdyż ta *zawsze* zostanie zaakceptowana, a to spowolni ewolucję układu. Propozycje można zredukować do odwracania. Ale wtedy w wysokich temperaturach układ będzie po prostu regularnie odwracał wszystkie spiny niezależnie od konfiguracji wyjściowej, będzie „migotał", co nie ma nic wspólnego z rzeczywistością ani z intuicją. Tak jest w wielu modelach fizyki komputerowej. Uproszczenia i optymalizacja nie zawsze idą w parze z użytecznością potrzebną do demonstracji.

Omówiony model jest popularny, ale dla odmiany nasza symulacja wykorzysta nieco inny algorytm, zwany *łaźnią cieplną*: ponieważ dla każdego spinu znamy jego aktualną energię oddziaływania, a ewentualne odwrócenie spinu zmienia tylko jej znak, możemy od razu losować nową konfigurację (a nie jej zmianę) z prawdopodobieństwem danym przez rozkład Gibbsa. Zauważmy, że nowe wartości spinów nie zależą w ogóle od ich wartości poprzednich, jedynie od sąsiadów. Nie jest to zupełnie fizyczne, każdy rzeczywisty obiekt ma pewną bezwładność, co powoduje, że zmiana jego atrybutów też kosztuje, bez względu na otoczenie, ale do naszych celów przyjęte założenie wystarczy.

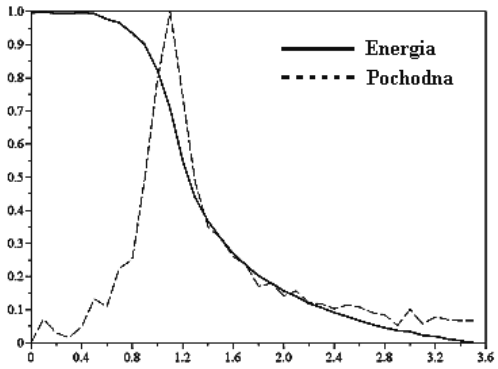
Poniższe rysunki pokazują z grubsza ustalone konfiguracje dla $T = 10$, $T = 1.2$ i $T = 0.5$. W niskich temperaturach wynik po bardzo długim okresie symulacji jest banalny, układ staje się jedną domeną, więc przerwaliśmy symulację wcześniej.



Pomiary

Reprezentacją układu jest tablica 300 × 300 punktów, z periodycznymi warunkami brzegowymi: na prawo od ostatniej kolumny znajduje się znowu pierwsza itp. Całkowity czas symulacji wyniósł zaledwie kilka minut na 2 GHz PC (mimo iż program napisany w Scilabie nie był najszybszy), więc możemy być nieco ambitniejsi. Symulacja rzadko sprowadza się do oglądania, popatrzmy więc na kilka globalnych (termodynamicznych) własności układu. Co interesującego możemy tutaj zmierzyć?

- Temperaturę, w jakiej zachowanie modelu ulega gwałtownej zmianie, przechodzi od fazy nieuporządkowanej do spontanicznie „namagnesowanej”.
- Energię całkowitą układu w funkcji temperatury, a także jej pochodną: ciepło właściwe.
- Średnią magnetyzację w funkcji temperatury, a także jej fluktuacje. Te ostatnie są ściśle związane z ciepłem właściwym układu.



Najpierw trzeba zdecydować się na zakres temperatur i pożądaną dokładność. Rysunek przedstawia energię i jej pochodną (obliczoną numerycznie) w funkcji temperatury; pochodna przedstawia oczywiście ciepło właściwe układu. Zauważamy, że w pobliżu $T = 1$ ciepło właściwe dość gwałtownie rośnie, co sugeruje przejście fazowe, „topnienie” lub „krzepnięcie”, przejście między stanami mniej lub bardziej uporządkowanymi.

Skale rysunku zostały dobrane dowolnie (a pochodna zmieniła znak), ale dobrym ćwiczeniem jest zastanowienie się nad wymiarami wielkości fizycznych otrzymywanymi z symulacji; kiedy operujemy tylko na liczbach, komputerowi jest wszystko jedno, czy są to milimetry, czy kilogramy. Jeśli osobie programującej te symulacje też jest wszystko jedno, to staje się to zabawą mającą saby związek z fizyką...

Interesującą wielkością byłaby także średnia magnetyzacja w funkcji temperatury. W wysokich temperaturach nie ma domen i średnia magnetyzacja winna być równa zeru. Poniżej temperatury przejścia magnetyzacja winna gwałtownie wzrosnąć. Jak jednak dokonać tego pomiaru? Policzyć średnią magnetyzację, tj. znormalizowaną sumę wartości spinów całego układu? Tak byłoby najprościej i w wysokich temperaturach dostalibyśmy zero. Jednak poniżej przejścia fazowego równowaga ustala się wolno i zupełnie możliwa jest sytuacja, w której połowa układu będzie „czarna", a druga połowa „biała" przez bardzo długi czas. Średni spin będzie nadal równy zeru. To nie jest dobra technika.

W sukurs przychodzi nam ***ergodyczność symulowanego układu***. W ogólności jest to pojęcie skomplikowane, wymagające dużej precyzji matematycznej. Dla nas wystarczy sformułowanie następujące: układ jest ergodyczny, jeśli popatrzenie na ewolucję jakiegoś jego małego fragmentu *w czasie* dostarcza nam próbek statystycznie równoważnych próbkom wziętym z tego samego układu w tym samym momencie, ale z różnych miejsc przestrzeni konfiguracyjnej. Proponujemy więc następujący algorytm: po wstępnym okresie ustalania równowagi popatrzymy na małą próbkę, powiedzmy 5× 5 spinów, ale przez dłuższy czas, i policzymy sumaryczny średni spin tego podukładu. Trzeba zdać sobie sprawę, że powtórzenie doświadczenia może dać wynik o odwrotnym znaku, to, czy w symulacji natrafiliśmy na domenę bardziej „białą", czy „czarną", jest kwestią przypadku (w należytym rozumieniu słowa *przypadek*...).

Statystyczne układy fizyczne zwykle są ergodyczne. Ale zaraz: przecież powiedzieliśmy, że czas symulacji nie ma nic wspólnego z czasem realnym. Nie szkodzi. Ba, można uzasadnić, że ten czy inny układ symulowany jest ergodyczny w oderwaniu od dynamiki modelowania. W naszym przypadku ergodyczność jest więc jeszcze jednym chwytem technicznym, ale zgodnym z naszą intuicją, ostatecznie *możemy* sobie wyobrazić, że z grubsza układ dąży do równowagi w symulowanym czasie tak, jakby to robił naprawdę. Realizacje tego lub innego algorytmu pozwalającego zmierzyć magnetyzację w modelu Isinga pozostawimy Czytelnikowi, a do tematu metod Monte-Carlo jeszcze wrócimy.

Kilka propozycji samodzielnych ćwiczeń

- Model Isinga dotyczy spinów „kwantowych", przyjmujących dwie wartości, np. w górę lub w dół osi *z*, jeśli przestrzenią układu jest płaszczyzna *xy*. Możemy popatrzyć na układ znacznie ciekawszy wizualnie, o spinach przyjmujących

wartości wektorowe $\vec{s}$ ~*s* w płaszczyźnie *xy*, z energią oddziaływania wyrażoną przez iloczyn skalarny $-J\vec{s}_i \cdot \vec{s}_j$. (Dla przyspieszenia symulacji można się ograniczyć np. do 256 możliwości i stablicować cosinusy kątów). Przedstawić tworzące się domeny, używając strzałek albo kolorów.

- Wprowadzić zewnętrzne pole magnetyczne, dodające do energii spinu dodatkowy człon, proporcjonalny do wartości samego spinu. Można wtedy popatrzyć, jak wygląda podatność magnetyczna, zależność średniej magnetyzacji od pola magnetycznego w różnych temperaturach.
- Przetestować inne algorytmy symulacji, np. algorytm z „demonem", który biega po układzie i losowo zabiera albo przydziela energię spinom, zachowując energię całkowitą. Pojęcie temperatury tutaj nie istnieje. Program co jakiś czas „oziębia" demona, odbierając mu część „łupu". W tej symulacji pomiar średniej energii jest oczywiście bez sensu, gdyż to *my* ją kontrolujemy. Zastanowić się, co mierzyć, aby wykryć przejście fazowe. Popatrzyć np. na fluktuacje magnetyzacji.

Szczegóły realizacji

Ponieważ naszym celem nie jest pokazywanie obrazków, tylko zachęcenie do własnej pracy, a wybór Scilabu jako języka programowania ma to ułatwić, przedstawimy w sposób niekompletny, ale zrozumiały, konstrukcję programu. O całość można się zwrócić do autora, a niebawem wszystko będzie zamieszczone na stronach internetowych *Fotonu*.

Dla osób znających typowe techniki programowania nietypowym elementem będzie unikanie pętli (instrukcji **for** i zbliżonych). Zaczniemy od utworzenia *maski* – tablicy pozwalającej oddzielić spiny „parzyste", s_{ij} z $i + j$ parzystym, od „nieparzystych". Ponieważ energia spinu zależy *wyłącznie* od sąsiadów, program zmieni „jednym strzałem" wszystkie spiny parzyste, a potem wszystkie nieparzyste. Wartość `N` poniżej wynosi 300, a `N1= N - 1.`

```
x=1:N;                // wektor [1,2,3,4,...]
m1=x(ones(x),:);      // tabl. złoż. z identycznych rzędów jak wyżej
m1=modulo(m1+m1',2); // tabl [0,1,0,1, ...]; Drugi rząd jest odwrotny
m2=1-m1;              // maska przeciwna
```

Tutaj `m1'` jest tablicą `m1` przetransponowaną (wiersze zastępują kolumny i odwrotnie). Teraz zadajemy temperaturę, np. `T = 1.2` (przejście fazowe jest koło jedynki) i definiujemy `beta=1/T`. Pamiętajmy, że w Scilabie forma `a(i,:)`, gdzie `a` jest tablicą dwuwymiarową, a `i` – pewną wartością wskaźnika, jest cała tablica 1-wymiarowa, stanowiąca *i*-tą kolumnę tablicy `a`.

```
a=2*round(rand(N,N))-1;    // początek symulacji, tablica losowa N×N
                           // zawiera elementy +1 oraz -1
```

```
e1=exp(beta); e2=e1*e1;
q1=e1/(e1+1/e1);
q2=e2/(e2+1/e2);          // To są prawdopodobieństwa nowych stanów!
pmat=[q2;q1;0.5;1-q1;1-q2]; // ... umieszczone w tablicy.
for i=1:200,              // Pętla stabilizująca stan
mm=m1; m1=m2; m2=mm;      // "Przekręcenie" masek
// Energia potencjalna aktualnej konfiguracji.
u=([a(:,N),a(:,1:N1)] + [a(:,2:N),a(:,1)] + ...
   [a(N,:);a(1:N1,:)] + [a(2:N,:);a(1,:)])/2;
   // u zawiera wartości -2, -1, 0, 1, oraz 2. Żeby wybrać odp.
   // prawdopodobieństwo, dodajemy 3 -> 1, 2, 3, 4, oraz 5.
b=matrix(pmat(u+3),[N,N]); // I to jest globalna tablica prawdo-
                           // podobieństw nowych stanów

r=rand(N,N);  // Podstawowa wartość losowa generująca nowy stan
c=((r<=b)*(-1)+(r>b)*1);   // *Ewentualne* nowe wartości
a=a.*m1 + c.*m2;
// Jedna maska wybiera spiny parzyste, druga maska nieparzyste. Przy na-
stępnej
// iteracji to się powtórzy, ale z drugą połową spinów. Po to właśnie
// przekręcamy maski przy każdej iteracji.
end;          // Koniec pętli iteracyjnej
```

I to jest wszystko, reszta jest utworzeniem wizualizacji i sumowaniem energii. Najbardziej charakterystyczną instrukcją jest obliczenie tablicy `u`: aby dodać wartości sąsiadów danego spinu, tworzymy tablice przesunięte: w lewo, w prawo, w górę i w dół i po prostu je dodajemy, co daje odpowiednie sumy dla wszystkich spinów na raz. Ten sposób programowania wymaga przyzwyczajenia, ale gdy to już nastąpi, staje się drugą naturą...

Podsumowanie

Liczby losowe, syntetyczne szumy i metody Monte-Carlo stanowią chleb powszedni naukowców i inżynierów. Warto się z nimi zapoznać. Pozwalają obliczać wyniki symulowanych skomplikowanych eksperymentów, a także dostarczają sporo technik użytecznych dla teoretyków. Ciekawe jest, że komputery, bezduszne narzędzia, ślepo wykonujące dokładnie określone matematyczne operacje, pozwoliły na rozwój technik operujących losem, nieokreślonością, fluktuacjami... Nie jest to jednak zasługą informatyki, a raczej dobrych teorii matematycznych oraz intuicji tych, którzy zaczęli je stosować.