Рух системи взаємодіючих частинок (до розділу 2)

```
const N=100;    // Кількість частинок
const Lx=20;    // Лінійні розміри прямокутного резервуара
const Ly=20;
const Scale=10; // Масштаб виведення точок на екран
double vx[N], vy[N], x[N], y[N], xbak[N], ybak[N], vmax=20;
double t=0, dt=1e-3, epsilon=0.5, sigma=1;
void Initial()
 {
  int j=1, i=1, k=0;
  double rx, ry, r;
  randomize();
         // Випадковим чином задаємо координати та проекції
         // швидкостей кожної частинки
  for (i=0; i<N; i++)
   {
```

```
  x[i]=random(Lx);
  y[i]=random(Ly);
  vx[i]=vmax*(2.0*random(1001)/1000.0-1);
  vy[i]=vmax*(2.0*random(1001)/1000.0-1);
 };
    // Коригуємо  координати  таким  чином, щоб
    // частинки були розподілені у  резервуарі
    // більш-менш рівномірно
for(;;)
 {
  int flag=0;
  for (i=0; i<N-1; i++)
   for (int j=i+1; j<N; j++)
    {
       rx=x[j]-x[i];
       ry=y[j]-y[i];
       r=sqrt(rx*rx+ry*ry);
       if (r<1)
        {
         x[j]=random(Lx);
         flag=1;
        };
    };
 if (!flag) break;
};
    // Розраховуємо повний імпульс системи в x-
    // та  y-напрямках і коригуємо  швидкості
    // таким чином, щоб повний імпульс системи
    // дорівнював нулю
double SumVx=0, SumVy=0;
for (i=0; i<N; i++)
  {
   SumVx+=vx[i];
   SumVy+=vy[i];
```

```
    }
  SumVx/=N;
  SumVy/=N;
  for (i=0; i<N; i++)
    {
     vx[i]-=SumVx;
     vy[i]-=SumVy;
    }
 };
    // Якщо  відстань між частинками перевищує половину
    // геометричного розміру резервуара - зменшуємо  її
    // відповідно  до  припущення про періодичність меж
    // резервуара (див. опис до програми)
void Test1(double *dx, double *dy)
 {
   int sign=*dx>0?1:-1;
   if (fabs(*dx)>Lx/2.0)
       *dx=*dx-sign*Lx;
   sign=*dy>0?1:-1;
   if (fabs(*dy)>Ly/2.0)
       *dy=*dy-sign*Ly;
 };
      // Якщо  частинка  досягає  стінки  резервуара, то
      // вона  з'являється з іншого боку
void Test2(double *x, double *y) {
  if (*x<0) *x=*x+Lx;
  if (*x>Lx) *x=*x-Lx;
  if (*y<0) *y=*y+Ly;
  if (*y>Ly) *y=*y-Ly;
};
         // Рисуємо частинки на екрані
void Out(double x[N], double y[N], double x1[N], \
                                          double y1[N])
{
```

```
  for (int i=0; i<N; i++)
   {
   ...
     // Видаляємо  частинки  з координатами  x[N], y[N].
     // Для  цього як поточний колір вибираємо колір фону
     // і малюємо даним кольором  частинки на екрані.
     // Координати перераховуємо згідно заданого масштабу
     // x[N/2]*Scale, y[N/2]*Scale
   ...
     // Вибираємо інший колір для виведення частинок.
     // Виводимо  частинки  у  точки  з  координатами
     // x1[i]*Scale, y1[i]*Scale
   }
};
     // Виводимо на екран залежність температури від часу
void GrT(double t, double T)
 {
  ...
     // Виводимо точку з координатами ((400+t*50),(300-T))
     // на графік
  ...
};
     // Розраховуємо траєкторії руху системи частинок
double x_n[N], y_n[N], vx_n[N], vy_n[N], a[N],b[N];
void Simula()
 {
  int i;
  double rx, rx6, ry, ry6, Fx, Fy, r, r6, F;
  double t=0, T, SumX, SumY;
    // На наступних кроках за часом координати та швидкості
    // частинок розраховуємо за методом з переступом
  for (i=0; i<N-1; i++)
   {
    SumX=SumY=0;
```

```
   for (int j=i+1; j<N; j++)
    {
     rx=x[i]-x[j];
     ry=y[i]-y[j];
     Test1(&rx,&ry);
     r=sqrt(rx*rx+ry*ry);
     r6=pow(sigma/r,6);
     F=24*epsilon*sigma/r*r6*(2*r6-1);
     SumX=SumX+F*rx;
     SumY=SumY+F*ry;
           // Згідно з третім законом Ньютона, збільшуючи
           // імпульс (швидкість) однієї із  взаємодіючих
           // частинок, зменшуємо імпульс іншої
    a[j]-=F*rx;
    b[j]-=F*ry;
   };
 vx[i]=vx[i]+(SumX+a[i])*dt/2;
 vy[i]=vy[i]+(SumY+b[i])*dt/2;
 };
vx[i]=vx[i]+a[i]*dt/2;
vy[i]=vy[i]+b[i]*dt/2;
while (!kbhit())
 {
   T=0;
   for (i=0; i<N; i++)
    {
     x_n[i]=x[i]+vx[i]*dt;
     y_n[i]=y[i]+vy[i]*dt;
     Test2(&x_n[i],&y_n[i]);
    };
   ...
      // Обнуляємо масиви a та b
   ...
   for (i=0; i<N-1; i++)
```

```
 {
  SumX=0;
  SumY=0;
  for (int j=i+1; j<N; j++)
   {
    rx=x_n[i]-x_n[j];
    ry=y_n[i]-y_n[j];
    Test1(&rx,&ry);
    r=sqrt(rx*rx+ry*ry);
    r6=pow(sigma/r,6);
           // Похідна від потенціалу Леннарда-Джонса
    F=24*epsilon*sigma/r*r6*(2*r6-1);
    SumX=SumX+F*rx;
    SumY=SumY+F*ry;
    a[j]-=F*rx;
    b[j]-=F*ry;
   };
  vx_n[i]=vx[i]+(SumX+a[i])*dt;
  vy_n[i]=vy[i]+(SumY+b[i])*dt;
 };
vx_n[i]=vx[i]+a[i]*dt;
vy_n[i]=vy[i]+b[i]*dt;
           // Розраховуємо температуру системи
for (i=0; i<N; i++)
  T+=vx_n[i]*vx_n[i]+vy_n[i]*vy_n[i];
T=T/N;
GrT(t,T);
Out(x,y,x_n,y_n);
...
     // копіюємо елементи масиву $x_n$  у  масив $x$,
     // відповідно  $y_n$ у масив $y$, $vx_n$ у $vx$,
     // $vy_n$ у масив $vy$
...
t+=dt;
```

```
   };
};
void main() {
  ...
  Initial();
  Simula();
  ...
}
```

Більярд Синая (до розділу 2)

```
%%%%%%%%%%%%%%%%%
% Програма Sinaj %
%%%%%%%%%%%%%%%%%
global R Lx Ly; R=5; Lx=100; Ly=100;
r=[10 60; 40 80];  % x- та y-координати куль
v=[10 3; 6 7];     % x- та y-компоненти швидкостей куль
plothandle = plot(r(1,:),r(2,:),'o', ...
      'Color','black', ...
      'MarkerSize',25);
 % Створюємо графічний  об'єкт - plot, за допомогою якого
 % рисуємо на екрані кулі у точках з x-координатами r(1,:)
 % (усі стовпці першого рядка матриці r) та y-координатами
 % r(2,:). Кулі розміром 25 чорного кольору виводяться на
 % екран у формі кола  ('o'). У подальшому робота з даним
 % об'єктом виконується через  дескриптор  plothandle  за
 % допомогою команди set.
axis([0 Lx 0 Ly]);
           % Встановлюємо  межі  зміни x- та y-координат.
dt=0.05;   % крок за часом
t=0;
pause;
 % пауза забезпечує негайне виведення зображення на екран;
 % для продовження необхідно натиснути будь-яку клавішу
```

```
while t<100,   % час розрахунків
  [r,v]=Raschet(r,v,dt);      % Викликаємо функцію Raschet,
      % передаємо початкові координати (масив r), швидкості
      % (масив v) та часовий інтервал. Як вихідні дані
      % [r,v] функція повертає нові координати та швидкостя
      % через час dt.
  pause(0.001);           % пауза для високошвидкісних ЕОМ
  set(plothandle,'xdata',r(1,:),'ydata',r(2,:));
                          % Перерисовуємо кулі на екрані.
  t=t+dt;
end;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Підпрограма розрахунку траєкторій руху куль на проміжку %
% часу dt                                                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [r,v]=Raschet(rin,vin,dt);
global R Lx Ly;
RR=4*R*R;
eps=1E-8;
r=rin; v=vin;
Left_Down_Corner=[R R; R R];
       % Координати лівої нижньої межі області для обох
       % куль, яка є доступною центрам куль
Right_Upper_Corner=[Lx-R Ly-R ;Lx-R Ly-R];
       % Координати правої верхньої межі
t=dt;
while (t>0)
 time_Left_Down_col=(Left_Down_Corner-r)./v;
       % Розраховуємо усі можливі моменти зіткнення з лівою
       % (у x-напрямку) та нижньою (у y-напрямку) стінками
       % для обох куль.
 time_Left_Down_col=time_Left_Down_col+...
                         1E8*((time_Left_Down_col <=0));
 % Якщо зіткнення відбулося у минулому
```

```
% time_Left_Down_col<=0,
% даний момент часу зіткнення нескінченно   збільшуємо
% (у подальшому він використовуватися не буде), інакше
% залишаємо без змін
time_Right_Upper_col=(Right_Upper_Corner-r)./v;
time_Right_Upper_col=time_Right_Upper_col+...
                    1E8*((time_Right_Upper_col <=0));
[t1,j1]=min(time_Left_Down_col(:));
[t2,j2]=min(time_Right_Upper_col(:));
% Вибираємо найбільш  раннє  зіткнення із стінками
% (t1 та t2), і кулю, яка  зіткнулася  із  стінкою
% (j1 та j2).
if(t1<t2)
 t0=t1;j0=j1;
else
 t0=t2;j0=j2;
end;
  % згідно з наведеними формулами знаходимо момент
  % зіткнення куль
r0=r(:,1)-r(:,2);
v0=v(:,1)-v(:,2);
rr=r0'*r0;
vv=v0'*v0;
rv=r0'*v0;
d=rv*rv-(rr-RR)*vv;      % дискримінант
if (d>0)&(rv<0)&(vv>eps)
 time_col=-(sqrt(d)+rv)/vv;
else
 time_col=inf;           % нескінченність
end;
if(t<t0)&(t<time_col)
          % Якщо поточний час менший за час
          % зіткнення куль зі стінкою або одна з одною
 r=r+v.*t;         % розраховуємо нові координати куль
```

```
 elseif (t0<time_col)  % якщо досягли часу зіткнення із
                       % стінкою
  r=r+v.*t0;
  v(j0)=-v(j0);   % змінюємо напрямок руху частинки при
                  % зіткненні із стінкою
 else             % зіткнення куль одна з одною
  t0=time_col;
  r=r+v.*t0;
  r0=r(:,1)-r(:,2);
  v0=v(:,1)-v(:,2);
  dv=r0'*v0/RR*r0;
  ddv=[-dv,dv ];
      % змінюємо швидкості відповідно до наведених
      % у розділі формул
  v=v+ddv;
 end;
 t=t-t0;
end;
```