```
============================ Program1.cpp ==================================

// compared to phase.cpp: x, r1, MAX, MAXWARM made dynamical; r2, calculation c

#include<iostream>
#include<cmath>
#include<fstream>

#include<stdio.h>
#include<string.h>
//#include<vector>

#define NMAX 1000
//#define  MAX 100
//#define MAXWARM 50
#define MAXITR  100000
// #define p_max 5
#define delta_t 0.01
//#define x 0
using namespace std;

int main()
{
  ofstream fp1("warm.dat"),fp2("final.dat"),fp3("rho1.dat"), fp4("rho2.dat");
  int MAX, MAXWARM;
  cout << "MAX, MAXWARM=?"<< endl;
  cin >> MAX >> MAXWARM;
long double t[MAX+1][MAX+1],p[MAX+1][MAX+1],newt[MAX+1][MAX+1],newp[MAX+1][MAX+
float x, r1, r2;
// initialize x (\xi= coupling)
     cout << "x, r1, r2=?" << endl;
     cin >> x >> r1>> r2;
     cout << x << " "<<  r1 <<" "<< r2<< endl;
     // long double kinetic=0;
for (int i=0;i<=MAX;i++)
    for (int j=0;j<=MAX;j++)
     {
       t[i][j]=(2*i-MAX)*(M_PI/MAX);// M_PI=3.14159265....
```

```
p[i][j]= (2*j-MAX)*(0.5/MAX);
//kinetic+= 0.5*(p[i][j])**2;
 }

// Warm up
// float r1=0.01;
for (int itr=0;itr<=MAXWARM;itr++)
{
for (int i=0;i<=MAX;i++)
     for (int j=0;j<=MAX;j++)
      {
  newt[i][j]=t[i][j]+delta_t*p[i][j] ;
if (newt[i][j] < -M_PI) newt[i][j]+=2*M_PI;
if (newt[i][j] >  M_PI)  newt[i][j]-=2*M_PI;
newp[i][j]=p[i][j]-2*x*r1*sin(t[i][j]) -
  2*x*r2*sin(2*t[i][j]);
  }

memcpy(t,newt,sizeof(newt));
memcpy(p,newp,sizeof(newp));
}

for (int i=0;i<=MAX;i++)
    for (int j=0;j<=MAX;j++)
{
fp1 << newt[i][j] << " " << newp[i][j] << endl;
}


// Main Routine
long double rho1=0; long double  rho2=0;
for (int itr=0;itr<=MAXITR;itr++)
{
for (int i=0;i<=MAX;i++)
for (int j=0;j<=MAX;j++)
{
  rho1+=cos(t[i][j]); rho2+= cos(2*t[i][j]);
}
```

```
rho1=rho1/((MAX+1)*(MAX+1));rho2=rho2/((MAX+1)*(MAX+1));

for (int i=0;i<=MAX;i++)
for (int j=0;j<=MAX;j++)
{
newp[i][j]=p[i][j]-x*rho1*sin(t[i][j])*delta_t; // First frog leap
newt[i][j]=t[i][j]+delta_t*newp[i][j] ; // Second frog leap
if (newt[i][j] < -M_PI) newt[i][j]+=2*M_PI;
if (newt[i][j] >  M_PI)  newt[i][j]-=2*M_PI;

}
// Third frog leap
for (int i=0;i<=MAX;i++)
for (int j=0;j<=MAX;j++)
{
rho1+=cos(newt[i][j]); rho2+= cos(2*newt[i][j]);
}
rho1=rho1/((MAX+1)*(MAX+1));rho2=rho2/((MAX+1)*(MAX+1));

for (int i=0;i<=MAX;i++)
for (int j=0;j<=MAX;j++)
{
newp[i][j]=newp[i][j]-x*rho1*sin(newt[i][j])*delta_t;
}
// Step ends

memcpy(t,newt,sizeof(newt));
memcpy(p,newp,sizeof(newp));
fp3 << itr << " "<< rho1 << endl;fp4 << itr << " "<< rho2 << endl;

}

for (int i=0;i<=MAX;i++)
    for (int j=0;j<=MAX;j++)
    {
fp2 << newt[i][j] << " " << newp[i][j] << endl;
}
```

```
}

=================== End-of-Program1.cpp ==========================

=================== Program2.cpp =================================

#include<iostream>
#include<cmath>
#include<sstream>
#include<string>
#include<fstream>
#include<vector>

#define debug false
#include<stdio.h>
#include<string.h>
//#include<vector>

#define MAX 100
#define MAXWARM 10
#define MAXITR  100000
#define SAVEITR 10000
#define delta_t 0.005
//#define xi .3
//# define a  0.170956
//# define b  0.566926
//#define xi 0.2
//#define a 0.125
//#define b 0
//# define xi .23
//#define  a 0.165959
//# define b 0.402027
# define xi 2
# define a -1.66631
# define b 0.865555


float r1=0.05;
```

```
using namespace std;

float f(float x) // This function defines the bh in phase space. a(xi),b(xi) ha
{
if ((a+2*b*xi*cos(x)>0))

return (sqrt(2*(a+2*b*xi*cos(x))));

else return(0.0);
}

class Point
{
public:
float t,p;
};

class PhaseObject
{
private:
vector<Point> newfinalpoints;

public:
int goodParticles;
float rho1;
vector<Point> finalpoints;

PhaseObject()
{ //Constructor
goodParticles=0;
Init_Blob();
}

void Init_Rectangle()
{
Point point[MAX+1][MAX+1];
// Initialization for the uniform configuration
for (int i=0;i<=MAX;i++)
```

```
    for (int j=0;j<=MAX;j++)
     {
            point[i][j].t=(2*i-MAX)*(M_PI/MAX);
            point[i][j].p=(2*j-MAX)*(0.5/MAX);
goodParticles++;
finalpoints.push_back(point[i][j]);
newfinalpoints.push_back(point[i][j]);
}
    }
void Init_Blob()
{ // Initialization for the bh configuration
Point point[MAX+1][MAX+1];
for (int i=0;i<=MAX;i++)
    for (int j=0;j<=MAX;j++)
     {
 float hmax=f(0)+0.15;
            point[i][j].t=(2*i-MAX)*(M_PI/MAX);
            point[i][j].p=(2*j-MAX)*(hmax/MAX);
if(fabs(point[i][j].p)<fabs(f(point[i][j].t)))
{
goodParticles++;
finalpoints.push_back(point[i][j]);
newfinalpoints.push_back(point[i][j]);
}
 }
 cout << ((float)goodParticles/((MAX+1)*(MAX+1))) << endl;
}

void Warm()
{ //Warm up routine.
for (int itr=0;itr<=MAXWARM;itr++)
{
for (unsigned int i=0;i < finalpoints.size();i++)
      {
  newfinalpoints[i].t=finalpoints[i].t+delta_t*finalpoints[i].p ;
if (newfinalpoints[i].t < -M_PI) newfinalpoints[i].t+=2*M_PI;
if (newfinalpoints[i].t >  M_PI) newfinalpoints[i].t-=2*M_PI;
newfinalpoints[i].p=finalpoints[i].p-2*xi*r1*sin(finalpoints[i].t);
}
finalpoints=newfinalpoints;
}
}

void McLoop()
{
//This function contains main montecarlo procedure, change it for any sign prob
    rho1=0;

for (unsigned int i=0;i<finalpoints.size();i++)
{
rho1+=cos(finalpoints[i].t);
}
rho1=rho1/goodParticles;

//rho1=b; // Not really

for (unsigned int i=0;i<finalpoints.size();i++)
{

newfinalpoints[i].p=finalpoints[i].p-xi*rho1*sin(finalpoints[i].t)*delta_t; //
newfinalpoints[i].t=finalpoints[i].t+delta_t*newfinalpoints[i].p ; // Second fr
if (newfinalpoints[i].t < -M_PI) newfinalpoints[i].t+=2*M_PI;
if (newfinalpoints[i].t >  M_PI) newfinalpoints[i].t-=2*M_PI;

}
// Third frog leap
rho1=0;
for (unsigned int i=0;i<finalpoints.size();i++)
{
rho1+=cos(newfinalpoints[i].t);
}
rho1=rho1/goodParticles;
//rho1=b;//Not really
for (unsigned int i=0;i<finalpoints.size();i++)
{
 newfinalpoints[i].p=newfinalpoints[i].p-xi*rho1*sin(newfinalpoints[i].t)*delta
```

```
} // Step ends
finalpoints=newfinalpoints;
}

float Calcrho_n(int n) //rho_n
{
float rhon=0;
for (unsigned int i=0;i<finalpoints.size();i++)
{
 rhon+=cos(n*newfinalpoints[i].t);
}
rhon/=goodParticles;
return(rhon);
}

float CalcE()
{
float E=0;
for (unsigned int i=0;i<finalpoints.size();i++)
E+=(finalpoints[i].p*finalpoints[i].p);
  E/=goodParticles;
  E-= 2*xi*rho1*rho1;
  return(E);
}


};

ostream& operator << (ostream& output, const PhaseObject& phase)
{
for (unsigned int i=0;i<(phase.finalpoints).size();i++) output << (phase.finalp
return(output);
}

ofstream& operator << (ofstream& output, const PhaseObject& phase)
{
for (unsigned int i=0;i<phase.finalpoints.size();i++)
output << phase.finalpoints[i].t << " " << phase.finalpoints[i].p << endl;
```

```
return(output);
}


int main()
{
ofstream fp0("init.dat"),fp1("warm.dat"),fp2("final.dat"),fp3("rho.dat");
PhaseObject phase;

fp0 << phase;

phase.Warm();

fp1 << phase;

for (int itr=0;itr<=MAXITR;itr++)
{
phase.McLoop();
fp3 << itr << " "<< phase.rho1 << " " << phase.Calcrho_n(2) << endl;
    }
fp2 << phase;
}

============== End-of-Program2.cpp =====================================
```