**Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования**

**"Уфимский университет науки и технологий"**

**Кафедра** Высокопроизводительных вычислительных технологий и систем

**Дисциплина:** Математическое моделирование.

**Отчет по лабораторной работе № 3**

**Тема:** «Исследование динамики одномерной цепочки частиц с различными потенциалами межчастичного взаимодействия Ферми-Паста-Улама»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Группа ПМ-457 | Фамилия И.О. | Подпись | Дата | Оценка |
| Студент | Исаева А.А. |  |  |  |
| Принял | Лукащук С.Ю. |  |  |  |

**Уфа 2024**

**Цель работы:** получить навык моделирования динамики системы многих частиц методами молекулярной динамики на примере задачи распространения возмущений в одномерной цепочке частиц одинаковой массы, связанных нелинейным потенциалом взаимодействия Ферми-Паста-Улама.

**Задание**

1. Определить вид правой части уравнения (1) для потенциала (2).
2. На языке программирования Си (Си++) выполнить программную реализацию приведенных численных алгоритмов Верле и симплектического типа Верле.
3. Провести сравнение точности алгоритмов посредством сравнения величины полной энергии системы (гамильтона)

после N временных шагов. Количество частиц n, параметры потенциала взаимодействия, а также значение N задаются преподавателем.

1. Для FPU-α потенциала взаимодействия определить диапазон значений в котором в пространстве скоростей существуют односолитонные решения при условии, что возмущение задаются в центре цепочки в виде приложениях заданных импульсов к двум соседним частицам, равных по величине и противоположных по направлению (так что суммарный импульс системы равен нулю). Выполнить анимационную визуализацию динамики цепочки.
2. Для FPU-β потенциала взаимодействия определить диапазоны значений в которых в пространстве скоростей существуют одно- , двух- и трехсолитонные решения. Возмущения задаются аналогично п. 4). Выполнить анимационную визуализацию цепочки.

**Теоретическая часть**

Рассматривается цепочка частиц одинаковой массы, находящаяся в начальный момент времени в состоянии равновесия (расстояния между частицами одинаковы). В качестве граничных условий используется условие периодичности (или условной замкнутости) цепочки. В качестве потенциала меж частичного взаимодействия рассматривается нелинейный степенной потенциал типа Ферми–Паста–Улама (FPU-потенциал)

|  |  |
| --- | --- |
|  | (1) |

При данный потенциал принято называть FPU-α, при – FPU-β.

Уравнение движения *i*-ой частицы будут иметь вид

|  |  |
| --- | --- |
|  | (2) |

Правая часть этого уравнения представляет собой сумму двух сил, действующих на *i*-ю частицу со стороны соседних (*i*–1)-ой и (*i*+1)-ой частиц. Величина – смещение *i*-ой частицы от положения равновесия.

Для интегрирования уравнений движения вида (1) используются два численных алгоритма

*Алгоритм Верле в скоростной форме.* Основные шаги алгоритма на каждом временном шаге:

Здесь и – скорость и ускорение *i*-ой частицы, соответственно. В начальный момент времени значения и известны и по ним рассчитываются значения .

*Симплектический алгоритм типа Верле в скоростной форме.* Основные действия алгоритма на каждом временном шаге:

Здесь параметр определяется из условия минимальности значения коэффициента при в оценке погрешности метода и равен

**Практическая часть**

**Уравнение движения**

для потенциала будет иметь вид:

|  |
| --- |
|  |

Начальные условия: при t = 0,

Начальное смещение: а = 1, Масса каждой частицы равна 1. , так как 5 вариант.

**Сравнение точности алгоритма посредством сравнения величины полной энергии системы (гамильтониана).**

Для того, чтобы проверить выполнение закона сохранения энергии для обоих алгоритмов возьмем .

|  |  |  |
| --- | --- | --- |
|  | Алгоритм Верле в скоростной форме | Симплектический алгоритм типа Верле в скоростной форме |
|  | 1.493e-06 | 9.241e-8 |
|  | 1.647e-06 | 9.406e-08 |
|  | 1.701e-06 | 1.212e-07 |

Симплектический алгоритм типа Верле в скоростной форме является более точным, чем алгоритм Верле в скоростной форме, однако у него уходит больше времени на вычисления

**Поиск солитонов с использованием симплектического алгоритма типа Верле в скоростной форме.**

Для FPU- потенциала подберём параметры , в котором в пространстве скоростей существуют односолитонные решения.

В результате для данной системы подходят параметры , .

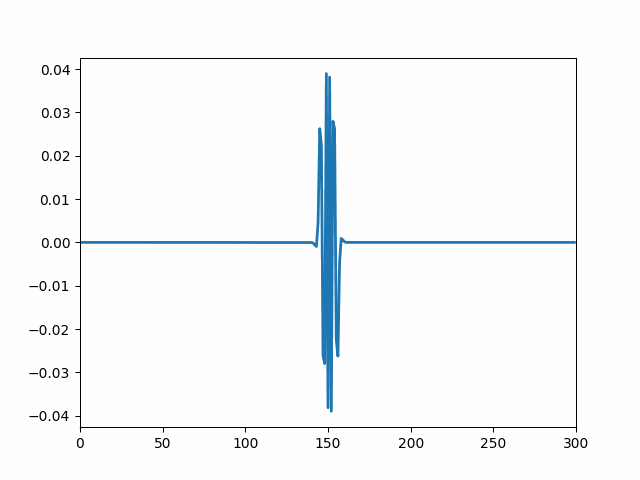


Рисунок 1 – Односолитонное решение при

Для FPU-β потенциала взаимодействия найдём диапазоны значений в которых в пространстве скоростей существуют одно- , двух- и трехсолитонные решения.

Односолитонное решение FPU-β потенциала существует при

. На рисунке 2 приведён пример анимации односолитонного решения с параметрами

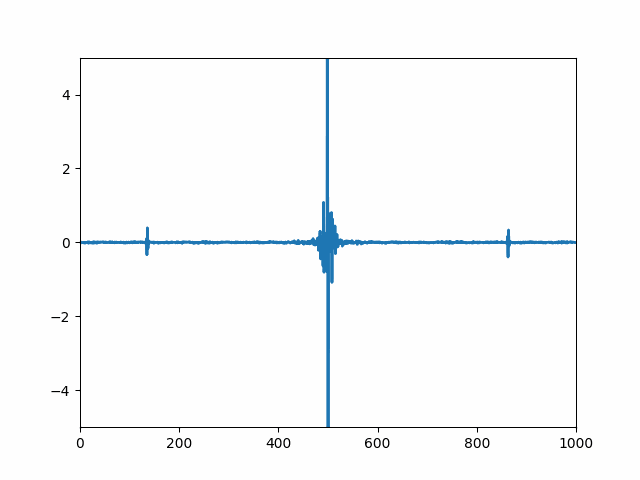


Рисунок 2 – Односолитонное решение при

Двухсолитонное решение FPU-β потенциала существует при

. На рисунке 3 приведён пример анимации односолитонного решения с параметрами .

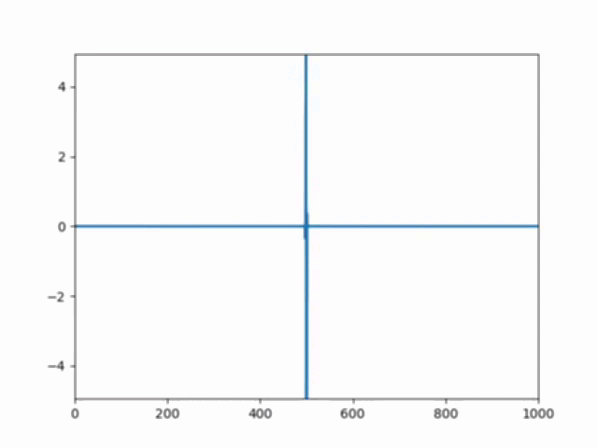


Рисунок 3 – Двухсолитонное решение при

Трёхсолитонное решение FPU-β потенциала существует при

. На рисунке 43 приведён пример анимации односолитонного решения с параметрами .

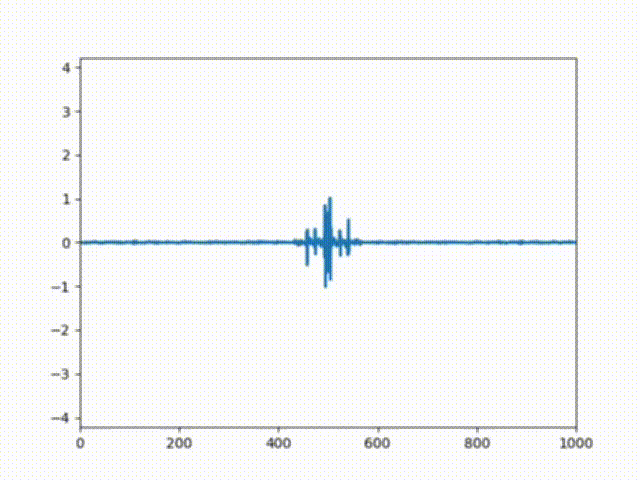


Рисунок 4 – Трехсолитонное решение при

**Вывод**

В ходе лабораторной работы были получены навыки моделирования динамики системы многих частиц методами молекулярной динамики на примере задачи распространения возмущений в одномерной цепочке частиц одинаковой массы, связанных нелинейным потенциалом взаимодействия типа Ферми-Паста-Улама. Проведено сравнение алгоритмов интегрирования движения частиц и сделан вывод о точности алгоритмов. Симплектический алгоритм типа Верле в скоростной форме точнее алгоритма Верле в скоростной форме на два порядка, но дольше работает. Визуализирована динамика частиц и найдены такие параметры, при которых возникают одно, двух- и трехсолитонные решения.

**Приложение**

#include <iostream>

#include <fstream>

#include <vector>

#include <cmath>

using namespace std;

// Проверка выполнения закона сохранения энергии

const int n = 500; // Число частиц

const int N\_time = 1e6; // Число слоев по времени

const double alpha = 1, beta = 1.;

const double tau = 0.01, m = 1.0;

const double xi = 0.1931833275037836;

double q0 = 0.06;

//1 солитон alpha

//const int n = 1000; // число частиц

//const int N\_time = 1e5; // число слоев по времени

//const double alpha = 0.7, beta = 0.;

//const double tau = 0.01, m = 1.0;

//const double xi = 0.1931833275037836;

//double q0 = 0.5;

// 1 солитон beta

//const int n = 1000; // Число частиц

//const int N\_time = 1e5; // Число слоев по времени

//const double alpha = 0., beta = 50;

//const double tau = 0.01, m = 1.0;

//const double xi = 0.1931833275037836;

//double q0 = 0.9;

const double v0 = 1.;

// 2 солитона beta

//const int n = 1000; // Число частиц

//const int N\_time = 3e6; // Число слоев по времени

//const double alpha = 0., beta = 106.;

//const double tau = 0.01, m = 1.0;

//const double xi = 0.1931833275037836;

//const double q0 = 0.9;

// 3 солитона beta

//const int n = 1000; // Число частиц

//const int N\_time = 1e5; // Число слоев по времени

//const double alpha = 0., beta = 200;

//const double tau = 0.01, m = 1.0;

//const double xi = 0.1931833275037836;

//double q0 = 0.9;

std::vector<double> GradV(std::vector<double>& q)

{

std::vector<double> GradV\_vector(q.size());

for (int i = 1; i < n - 1; i++)

{

GradV\_vector[i] = (q[i + 1] - 2 \* q[i] + q[i - 1]) +

alpha \* (pow(q[i + 1] - q[i], 2) - pow(q[i] - q[i - 1], 2)) +

beta \* (pow(q[i + 1] - q[i], 3) - pow(q[i] - q[i - 1], 3));

}

return GradV\_vector;

}

void Verle(std::vector<double>& q, std::vector<double>& v, std::vector<double>& a)

{

std::ofstream f\_out;

f\_out.open("data//Verle.dat");

clock\_t t\_solv\_end = clock();

for (int i = 0; i < N\_time; i++)

{

clock\_t t = clock();

if (i % 100 == 0)

{

for (int j = 0; j < n; j++)

{

f\_out << v[j] << " ";

}

f\_out << "\n";

}

for (int j = 1; j < n - 1; j++)

{

q[j] = q[j] + v[j] \* tau + 0.5 \* a[j] \* tau \* tau;

v[j] = v[j] + 0.5 \* a[j] \* tau;

}

q[0] = q[n - 2];

v[0] = v[n - 2];

q[n - 1] = q[1];

v[n - 1] = v[1];

std::vector<double> GradV\_vector = GradV(q);

for (int j = 1; j < n - 1; j++)

{

a[j] = 1. / m \* GradV\_vector[j];

v[j] = v[j] + 0.5 \* a[j] \* tau;

}

t = clock() - t;

#ifndef NDEBUG

std::cout << "Time for one iteration" << t << " clicks (" << ((double)t) / CLOCKS\_PER\_SEC << " seconds).\n";

std::cout << "End calculation for t = " << (i \* 100) / (N\_time) << "%\n";

#endif // NDEBUG

if (i % 100 == 0) std::cout << "End calculation Verle for t = " << (i \* 100) / (N\_time) << "%\n";

}

t\_solv\_end = clock() - t\_solv\_end;

std::cout << "Time Solve Verle: " << t\_solv\_end << " clicks (" << ((double)t\_solv\_end) / CLOCKS\_PER\_SEC << " seconds).\n";

for (int j = 0; j < n; j++)

{

f\_out << v[j] << " ";

}

f\_out << "\n";

f\_out.close();

}

void simplexVerle(std::vector<double>& q, std::vector<double>& v, std::vector<double>& a)

{

std::ofstream f\_out;

f\_out.open("data//simplexVerle.dat");

clock\_t t\_solv\_end = clock();

for (int i = 0; i < N\_time; i++)

{

for (int j = 0; j < n; j++)

{

f\_out << v[j] << " ";

}

f\_out << "\n";

clock\_t t = clock();

for (int j = 0; j < n; j++)

{

q[j] = q[j] + v[j] \* xi \* tau;

}

std::vector<double> GradV\_vector = GradV(q);

for (int j = 1; j < n - 1; j++)

{

a[j] = 1. / m \* GradV\_vector[j];

v[j] = v[j] + 0.5 \* a[j] \* tau;

}

for (int j = 0; j < n; j++)

{

q[j] = q[j] + v[j] \* (1. - 2. \* xi) \* tau;

}

GradV\_vector = GradV(q);

for (int j = 1; j < n - 1; j++)

{

a[j] = 1. / m \* GradV\_vector[j];

v[j] = v[j] + 0.5 \* a[j] \* tau;

q[j] = q[j] + v[j] \* xi \* tau;

}

int s = q.size() / 2;

for (int i = s + 1; i < q.size(); i++) {

v[i] = -v[q.size() - 1 - i];

a[i] = -a[q.size() - 1 - i];

q[i] = -q[q.size() - 1 - i];

}

q[0] = q[n - 2];

v[0] = v[n - 2];

q[n - 1] = q[1];

v[n - 1] = v[1];

t = clock() - t;

#ifndef NDEBUG

//std::cout << "Time for one iteration" << t << " clicks (" << ((double)t) / CLOCKS\_PER\_SEC << " seconds).\n";

//std::cout << "End calculation for t = " << (i \* 100) / (N\_time) << "%\n";

#endif // NDEBUG

//if (i % 100 == 0) std::cout << "End calculation simplexVerle for t = " << (i \* 100) / (N\_time) << "%\n";

}

t\_solv\_end = clock() - t\_solv\_end;

std::cout << "Time Solve simplexVerle: " << t\_solv\_end << " clicks (" << ((double)t\_solv\_end) / CLOCKS\_PER\_SEC << " seconds).\n";

for (int j = 0; j < n; j++)

{

f\_out << v[j] << " ";

}

f\_out << "\n";

f\_out.close();

}

double V(double r)

{

return (r \* r / 2. + alpha \* r \* r \* r / 3. + beta \* r \* r \* r \* r / 4.);

}

double Gamiltonian(std::vector<double>& v, std::vector<double>& q)

{

double P = 0.0;

// Левая точка выколота

for (int i = 1; i < n - 1; i++)

{

P += m \* v[i] \* v[i] / 2. + V(q[i + 1] - q[i]);

}

return P;

}

double F(double q\_last, double q, double q\_next, double m, double alpha, double beta) {

return 1.0 / (m) \* ((q\_next - 2 \* q + q\_last) + alpha \* (q\_next - q) \* (q\_next - q) + beta \* (q\_next - q) \* (q\_next - q) \* (q\_next - q)

- alpha \* (q - q\_last) \* (q - q\_last) - beta \* (q - q\_last) \* (q - q\_last) \* (q - q\_last));

}

vector<vector<double>> SimplexVerle2(vector<double> q, vector<double>v, double alpha, double beta, double tau, int N, double m) {

vector<double> a(q.size(), 0);

int s = q.size() / 2 - 1;

ofstream f("data//speed.txt");

for (int t = 0; t < N; t++) {

for (int i = 0; i < s + 1; i++) {

q[i] = q[i] + v[i] \* tau \* xi;

}

a[0] = F(-q[0], q[0], q[1], m, alpha, beta);

for (int i = 1; i < s; i++) {

a[i] = F(q[i - 1], q[i], q[i + 1], m, alpha, beta);

}

a[s] = F(q[s - 1], q[s], -q[s], m, alpha, beta);

for (int i = 0; i < s + 1; i++) {

v[i] = v[i] + 0.5 \* a[i] \* tau;

q[i] = q[i] + v[i] \* tau \* (1 - 2 \* xi);

}

a[0] = F(-q[0], q[0], q[1], m, alpha, beta);

for (int i = 1; i < s; i++) {

a[i] = F(q[i - 1], q[i], q[i + 1], m, alpha, beta);

}

a[s] = F(q[s - 1], q[s], -q[s], m, alpha, beta);

for (int i = 0; i < s + 1; i++) {

v[i] = v[i] + 0.5 \* a[i] \* tau;

q[i] = q[i] + v[i] \* tau \* xi;

}

for (int i = s + 1; i < q.size(); i++) {

v[i] = -v[q.size() - 1 - i];

a[i] = -a[q.size() - 1 - i];

q[i] = -q[q.size() - 1 - i];

}

if (t % 300 == 0 && t > 000000) {

// cout << t + 1 << endl;

// cout << v[s / 2] << " " << v[s / 2 + 1] << endl;

for (int i = 0; i < v.size(); i++) f << v[i] << " ";

f << endl;

}

}

for (int i = 0; i < v.size(); i++) f << v[i] << " ";

// f << endl;

f.close();

return { q,v };

}

int main()

{

std::vector<double> q(n, 0), v(n, 0), a(n, 0);

std::vector<double> GradV\_vector;

for (int i = 0; i < n; i++)

{

q[i] = v[i] = a[i] = 0.0;

}

v[n / 2] = 1.;

v[n / 2 + 1] = -v0;

//q[n / 2 - 1] = q0;

//q[n / 2] = -q0;

GradV\_vector = GradV(q);

for (int j = 1; j < n - 1; j++)

{

a[j] = 1. / m \* GradV\_vector[j];

}

double start = Gamiltonian(v, q);

std::cout << "first H = " << start << "\n";

//simplexVerle(q, v, a);

simplexVerle(q, v, a);

double end = Gamiltonian(v, q);

std::cout << "after simplexVerle H = " << end << "\n";

std::cout << "simplexVerle уrror: " << abs(start - end) << "\n";

return 0;

}

// Начальные условия

//q[n / 2 + 1] = q0;

//q[n / 2] = -q0;

//GradV\_vector = GradV(q);

//for (int j = 1; j < n - 1; j++)

//{

// a[j] = 1. / m \* GradV\_vector[j];

//}

//double start = Gamiltonian(v, q);

//std::cout << "first H = " << start << "\n";

//Verle(q, v, a);

//double end = Gamiltonian(v, q);

//std::cout << "after Verle H = " << end << "\n";

//std::cout << "Verle уrror: " << abs(start - end) << "\n";