**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ 6**

В кинетических алгоритмах МК определяется время совершения каждого события, что позволяет вести наблюдения за динамикой реакционной системы.

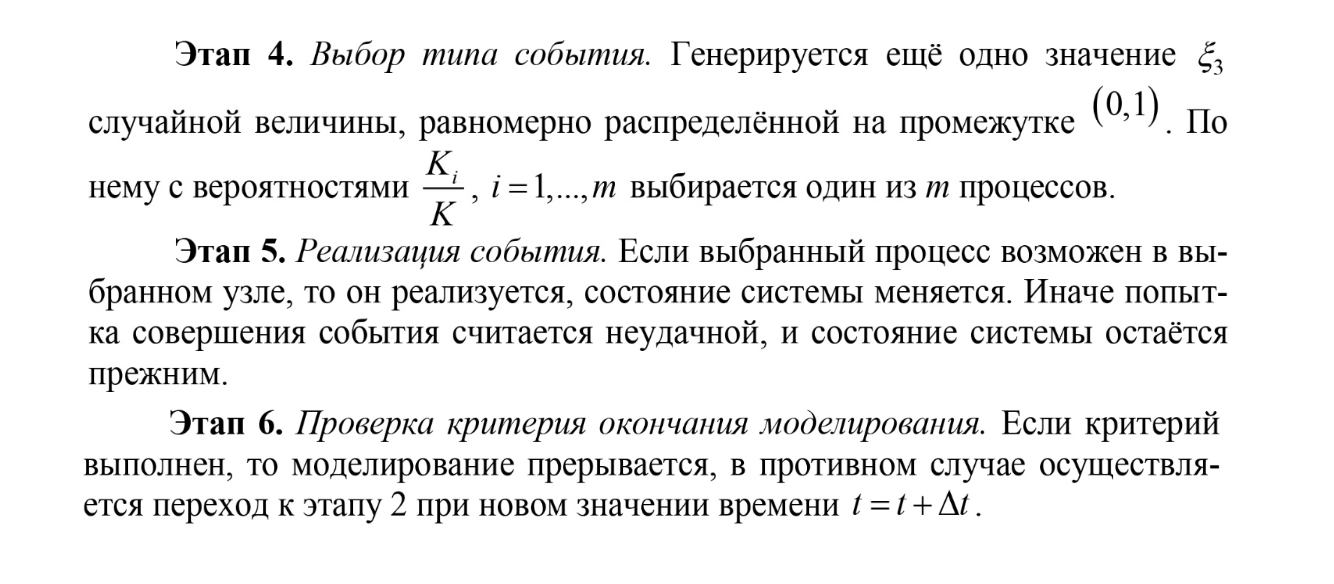
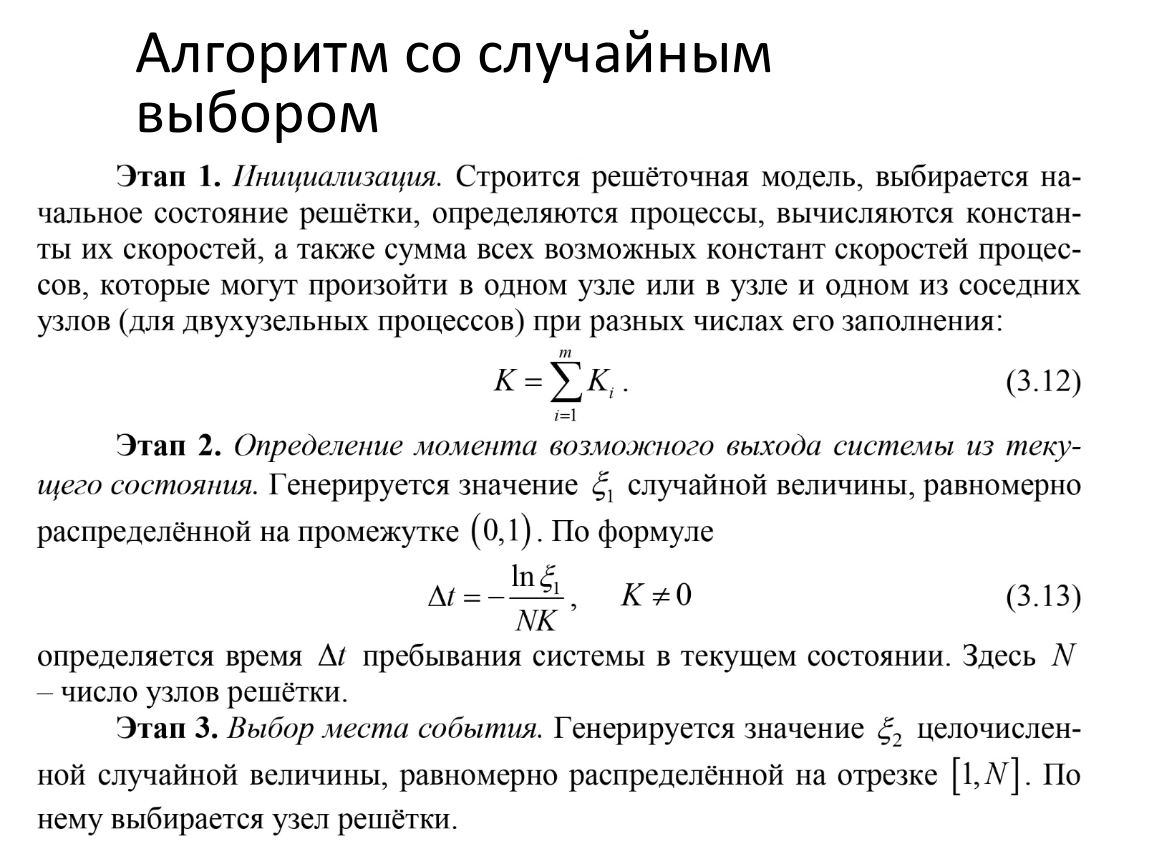
Вероятности событий на решётке в КМК алгоритмах пропорциональны скоростям соответствующих элементарных процессов независимо от того, приводят события к уменьшению или увеличению энергии системы

В алгоритмах с отказами наряду с успешными попытками совершения событий в решёточной системе рассматриваются и неудачные.

Часть вычислительных ресурсов тратится на анализ ситуаций, которые не ведут к изменению системы.

Может стать неэффективным для жестких систем, т.е. систем, где скорости элементарных процессов решетки различаются на несколько порядков.

Очень трудоемки в расчете из-за пересчета вероятностей перехода на каждой итерации, даже там, где решетка не будет меняться



**Код**

#include <iostream>

#include <vector>

#include <cmath>

#include <random>

#include <chrono>

#include <thread>

#include <fstream>

// Константы для управления параметрами симуляции

const int GRID\_SIZE = 20; // Размер сетки

const int SIMULATION\_TIME = 10 \* (GRID\_SIZE \* GRID\_SIZE); // Время симуляции

const int SLEEP\_TIME = 95; // Задержка между шагами (мс)

const double MOVE\_PROB = 1.0; // Вероятность движения

const double BIRTH\_PROB = 3.0; // Вероятность размножения

const double DEATH\_PROB = 2.0; // Вероятность смерти

const double SUM\_EVENTS = MOVE\_PROB + BIRTH\_PROB + DEATH\_PROB; // Сумма вероятностей событий

// Названия файла для вывода и обозначения клеток

const std::string GRID\_FILENAME = "output.txt";

const std::string LIVING\_CELL = "O "; // Живая клетка в файле

const std::string DEAD\_CELL = ". "; // Мертвая клетка в файле

const std::string LIVING\_CELL\_CMD = "\u2591"; // Живая клетка в терминале

const std::string DEAD\_CELL\_CMD = "\u2588"; // Мертвая клетка в терминале

const std::string RESET\_COLOUR = "\033[0m"; // Сброс цвета

const std::string LIGHT\_GREEN = "\033[1;32m"; // Светло-зеленый цвет

// Перечисления для типа событий и типа вывода

enum class Event {

MOVE = 1, // Движение

DEATH, // Смерть

BIRTH // Размножение

};

enum class PrintType {

TERMINAL, // Вывод в терминал

TO\_FILE // Вывод в файл

};

// Структура клетки

struct Cell {

int x, y; // Координаты клетки

bool alive; // Статус клетки (живая/мертвая)

};

// Объявление функций

std::vector<std::vector<Cell>> initialize\_grid(std::vector<std::pair<int, int>>& first\_generation);

int count\_neighbors(const std::vector<std::vector<Cell>>& grid, int x, int y);

void perform\_event(std::vector<std::vector<Cell>>& grid, int x, int y, std::default\_random\_engine& rng);

void run\_simulation(PrintType type);

void print\_grid(const std::vector<std::vector<Cell>>& grid);

void print\_grid\_to\_file(const std::vector<std::vector<Cell>>& grid, std::ofstream& output);

Event linear\_search(double number);

bool has\_alive\_cells(const std::vector<std::vector<Cell>>& grid);

PrintType process\_user\_input();

unsigned short int actions\_count = 0; // Счетчик событий

int main() {

PrintType type = process\_user\_input(); // Определение типа вывода от пользователя

run\_simulation(type); // Запуск симуляции с заданным типом вывода

return 0;

}

// Функция для обработки ввода от пользователя и выбора типа вывода

PrintType process\_user\_input() {

std::cout << "Choose where to print data: enter terminal for terminal and file for file\n>";

std::string input;

while(true) {

std::cin >> input;

if (input == "terminal")

return PrintType::TERMINAL;

else if (input == "file")

return PrintType::TO\_FILE;

}

}

// Инициализация сетки с первыми живыми клетками

std::vector<std::vector<Cell>> initialize\_grid(std::vector<std::pair<int, int>>& first\_generation) {

std::vector<std::vector<Cell>> grid(GRID\_SIZE, std::vector<Cell>(GRID\_SIZE, {0, 0, false}));

for(const auto& gen : first\_generation) {

grid[gen.first][gen.second].alive = true;

}

return grid;

}

// Подсчет количества живых соседей для клетки

int count\_neighbors(const std::vector<std::vector<Cell>>& grid, int x, int y) {

int count = 0;

const std::vector<std::pair<int, int>> neighbors = {

{1, 0}, {-1, 0}, {0, 1}, {0, -1} // Соседи (вверх, вниз, вправо, влево)

};

for (const auto& [dx, dy] : neighbors) {

int nx = (x + dx + GRID\_SIZE) % GRID\_SIZE; // Обработка выхода за границы по x

int ny = (y + dy + GRID\_SIZE) % GRID\_SIZE; // Обработка выхода за границы по y

if (grid[nx][ny].alive) ++count; // Увеличиваем счетчик, если клетка живая

}

return count;

}

// Линейный поиск события на основе случайного числа

Event linear\_search(double number) {

double prob = MOVE\_PROB / SUM\_EVENTS;

if (number < prob)

return Event::MOVE;

prob += BIRTH\_PROB / SUM\_EVENTS;

if (number < prob)

return Event::BIRTH;

return Event::DEATH;

}

// Выполнение события (движение, смерть, размножение)

void perform\_event(std::vector<std::vector<Cell>>& grid, int x, int y, std::default\_random\_engine& rng) {

int neighbors = count\_neighbors(grid, x, y); // Подсчет соседей

std::uniform\_real\_distribution<double> dist(0.0, 1.0);

double event = dist(rng);

if (grid[x][y].alive) {

Event action = linear\_search(event); // Определение типа события

switch(action){

case Event::MOVE: { // Движение

std::vector<std::pair<int, int>> moves = {

{1, 0}, {-1, 0}, {0, 1}, {0, -1}

};

int index = rng() % moves.size(); // Случайный выбор направления движения

int nx = (x + moves[index].first + GRID\_SIZE) % GRID\_SIZE;

int ny = (y + moves[index].second + GRID\_SIZE) % GRID\_SIZE;

if (!grid[nx][ny].alive) { // Если целевая клетка мертвая, перемещаем клетку

grid[nx][ny].alive = true;

grid[x][y].alive = false;

++actions\_count;

}

break;

}

case Event::DEATH: { // Смерть

if (neighbors > 3) { // Умирает, если более 3 соседей

grid[x][y].alive = false;

++actions\_count;

}

break;

}

case Event::BIRTH: { // Размножение

if (neighbors > 0 && neighbors < 4) { // Если 1-3 соседа, размножение

std::vector<std::pair<int, int>> moves = {

{1, 0}, {-1, 0}, {0, 1}, {0, -1}

};

for (const auto& move : moves) {

int nx = (x + move.first + GRID\_SIZE) % GRID\_SIZE;

int ny = (y + move.second + GRID\_SIZE) % GRID\_SIZE;

if (!grid[nx][ny].alive){ // Размножение в свободной клетке

grid[nx][ny].alive = true;

++actions\_count;

break;

}

}

}

break;

}

}

}

}

// Проверка наличия живых клеток на поле

bool has\_alive\_cells(const std::vector<std::vector<Cell>>& grid) {

for (int i = 0; i < GRID\_SIZE; ++i) {

for (int j = 0; j < GRID\_SIZE; ++j) {

if (grid[i][j].alive)

return true;

}

}

return false;

}

// Печать сетки в терминал

void print\_grid(const std::vector<std::vector<Cell>>& grid) {

for (int i = 0; i < GRID\_SIZE; ++i) {

for (int j = 0; j < GRID\_SIZE; ++j) {

if (grid[i][j].alive)

std::cout << LIVING\_CELL\_CMD;

else

std::cout << LIGHT\_GREEN << DEAD\_CELL\_CMD << RESET\_COLOUR;

}

std::cout << '\n';

}

std::cout << '\n';

}

// Печать сетки в файл

void print\_grid\_to\_file(const std::vector<std::vector<Cell>>& grid, std::ofstream& output) {

for (int i = 0; i < GRID\_SIZE; ++i) {

for (int j = 0; j < GRID\_SIZE; ++j) {

output << (grid[i][j].alive ? LIVING\_CELL : DEAD\_CELL);

}

output << '\n';

}

output << "\n\n";

}

// Запуск симуляции с выбранным типом вывода

void run\_simulation(PrintType type) {

std::vector<std::pair<int, int>> first\_generation = { {5, 5}, {15, 5}, {5, 15}, {15, 15} };

auto grid = initialize\_grid(first\_generation);

std::default\_random\_engine rng(std::chrono::system\_clock::now().time\_since\_epoch().count());

std::uniform\_int\_distribution<int> cell\_selector(0, GRID\_SIZE - 1);

int chosen\_x = 0, chosen\_y = 0;

switch(type){

case PrintType::TERMINAL: {

for (int t = 0; t < SIMULATION\_TIME; ++t) {

std::cout << "\x1b[2J\nStep " << t << ":\n";

std::cout << "Chosen cell: " << '(' << chosen\_x << ';' << chosen\_y << ')' << '\n';

print\_grid(grid);

int x = cell\_selector(rng);

int y = cell\_selector(rng);

chosen\_x = x;

chosen\_y = y;

perform\_event(grid, x, y, rng);

if(!has\_alive\_cells(grid)) {

std::cout << "All the cells died out after " << t << " steps.\n";

break;

}

std::this\_thread::sleep\_for(std::chrono::milliseconds(SLEEP\_TIME));

}

break;

}

case PrintType::TO\_FILE: {

std::ofstream output(GRID\_FILENAME);

if (!output.is\_open()){

std::cerr << "Error in opening file " << GRID\_FILENAME << '\n';

exit(0);

}

for (int t = 0; t < SIMULATION\_TIME; ++t) {

output << "Step " << t << ":\n";

print\_grid\_to\_file(grid, output);

int x = cell\_selector(rng);

int y = cell\_selector(rng);

perform\_event(grid, x, y, rng);

if(!has\_alive\_cells(grid)) {

std::cout << "All the cells died out after " << t << " steps.\n";

break;

}

}

output.close();

std::cout << "Data has been saved to " << GRID\_FILENAME << '\n';

break;

}

}

std::cout << "\n\nSome statistics:\n" << actions\_count << " actions were taken in " << SIMULATION\_TIME << " steps\n";

}

import matplotlib.pyplot as plt

# Функция для парсинга файла

def parse\_file(filename):

steps = []

live\_cells = []

with open(filename, 'r') as file:

step = 0

alive\_count = 0

in\_step = False

for line in file:

if line.startswith("Step"): # Определяем начало нового шага

if in\_step: # Если это не первый шаг, сохраняем данные предыдущего шага

steps.append(step)

live\_cells.append(alive\_count)

step = int(line.split()[1].strip(':')) # Получаем номер шага

alive\_count = 0 # Обнуляем счётчик живых клеток для нового шага

in\_step = True

elif '.' in line or 'O' in line: # Считываем состояние клеток в строке

alive\_count += line.count('O') # Считаем живые клетки ('O')

# Добавляем последний шаг в данные

if in\_step:

steps.append(step)

live\_cells.append(alive\_count)

return steps, live\_cells

# Функция для построения графика

def plot\_data(steps, live\_cells):

plt.figure(figsize=(10, 6))

plt.plot(steps, live\_cells, marker='o', color='b', linestyle='-', label='Live Cells')

plt.title('Number of Live Cells over Steps (Time)')

plt.xlabel('Step (Time)')

plt.ylabel('Number of Live Cells')

plt.grid(True)

plt.legend()

plt.show()

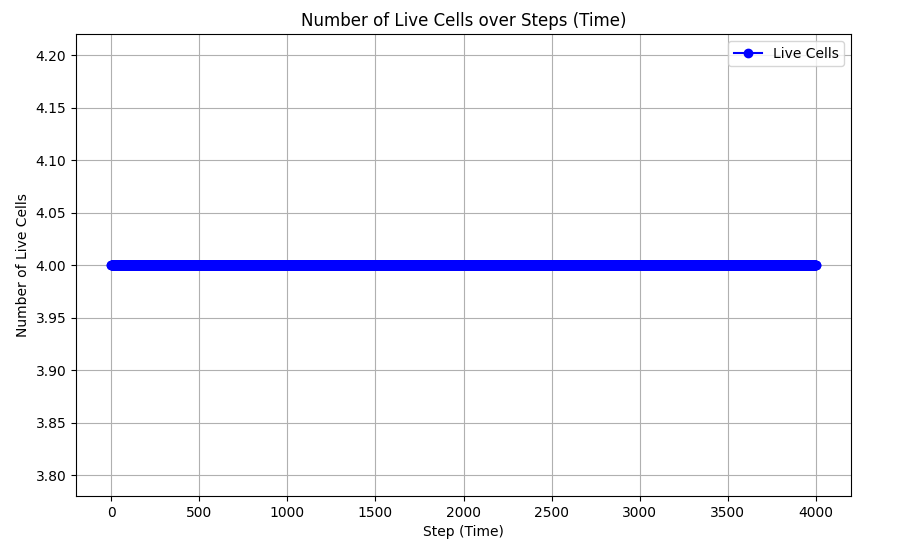
# Основная функция

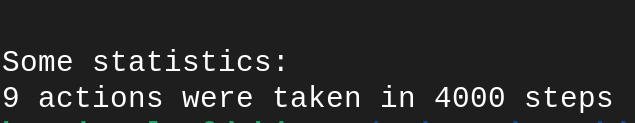
if \_\_name\_\_ == '\_\_main\_\_':

filename = '/home/buzzismaloy/University\_shit/7th-term/monte-karlo/sixth-lab/output.txt' # Укажите путь к вашему файлу

steps, live\_cells = parse\_file(filename)

plot\_data(steps, live\_cells)

  
Рис.1. Зависимость количества живых клеток от времени(шаг)



**Вывод**

Тк в данном методе с успешными попытками совершения событий в решёточной системе рассматриваются и неудачные, то система изменяется крайне медленно.