

ISING PROJECT

БОНДАРЬ РОМАН
ПЯТКИН СТАНИСЛАВ
СЕМЕНЕНКО АЛЕКСАНДР

ЕФПМИ

Долгопрудный 2020

Содержание

1 Введение	2
1.1 Цель проекта	2
1.2 Краткое описание	2
2 Теория	3
2.1 Описание модели	3
2.2 Алгоритмы модели	4
2.2.1 Heat bath algorithm	4
2.2.2 Cluster algorithm	4
2.3 Работа алгоритмов	5
3 Инструкция по использованию	7
3.1 Технические требования	7
3.2 Сборка проекта	7
3.3 Пример работы	7
3.4 Параметры командной строки	8
4 Реализация	9
4.1 Устройство классов	9
4.2 Взаимодействие модулей	9
4.3 Распределение обязанностей	11
5 Средства разработки	11
Список источников	11

1 Введение

1.1 Цель проекта

Смоделировать фазовый переход с помощью двумерной модели Изинга (математическая модель, предназначенная для описания намагничивания материала). Исследовать её поведение при различных температурах, а также создать удобный графический интерфейс для визуализации модели в работе.

1.2 Краткое описание

Каждому узлу кристаллической решетки сопоставляется число, равное ± 1 («направление вверх»/«направление вниз»). С помощью программы в реальном времени можно наблюдать, как эволюционирует модель. От внешних условий будет зависеть ориентация спинов, а именно будет ли она у всех одинакова или хаотически распределена.

2 Теория

2.1 Описание модели

Состояние модели полностью описывается значениями спинов в узлах решетки, и несколькими дополнительными параметрами. Значения спинов и их количество хранится в объекте базового класса **lattice**:

lattice::N - число спинов.

lattice::L - массив из N значений спинов σ_i .

lattice::nbrs - степень узла решетки (число соседей каждого спина).

Параметры модели и симуляции хранятся в объекте класса **parameters**:

parameters::beta - величина β , обратная температуре:

$$\beta = \frac{1}{kT} \quad (1)$$

parameters::J - энергия взаимодействия соседних в решетке спинов.

parameters::H - внешнее магнитное поле H .

parameters::mu - магнитный момент спина μ .

Каждому состоянию S из 2^N возможных приписывается энергия, равная:

$$E(S) = -J \sum_{i,j-neighbors} \sigma_i \sigma_j - H \sum \mu \sigma_i \quad (2)$$

Алгоритм симуляции **heat_bath_simulate** предельно прост:

- Выбрать произвольный спин σ .
- Присвоить $\sigma = +1$ вероятностью π_h^+ , и -1 с вероятностью $(1 - \pi_h^+)$.
- Повторить $N \cdot \mathbf{steps}$ раз.

π_h^+ можно вычислить, используя распределение Гиббса по энергиям:

$$p(S) \sim e^{-\beta E(S)} \quad (3)$$

$$\pi_h^+ = \frac{e^{-\beta E^+}}{e^{-\beta E^+} + e^{-\beta E^-}}$$

где E^+ и E^- - энергии состояний, в которых $\sigma = +1$ и -1 соответственно.

Количество шагов алгоритма **steps** берется достаточное, чтобы система пришла в термодинамическое равновесие.

Однако вблизи температуры фазового перехода β_c , сказывается так называемый “эффект критического замедления”, когда **steps** $\rightarrow \infty$, и **heat_bath_simulate** неэффективен.

Для преодоления этого эффекта реализован алгоритм **clusters_simulate**.

В отличие от **heat_bath_simulate**, спины переворачиваются не поодиночке, а кластерами. Вот краткое описание его работы:

- Начать собирать кластер с произвольного спина σ .

- Добавлять в кластер соседние спины того же знака, что и σ , с вероятностью $\pi(\beta)$, и по завершении перевернуть кластер.
- Повторить **steps** раз.

Подробнее оба алгоритма и их реализация описаны в следующем разделе.

2.2 Алгоритмы модели

2.2.1 Heat bath algorithm

Алгоритм достаточно прост, и вся его смысловая часть заключена в трех строчках внутри двойного цикла.

Был значительно оптимизирован подсчет π_h^+ :

$$\pi_h^+ = \frac{e^{-\beta E^+}}{e^{-\beta E^+} + e^{-\beta E^-}} = \frac{1}{1 + e^{-2J\beta h - \mu H}}$$

где h - сумма спинов соседей (высчитывается методом **sum_nbr**).

Также, поскольку $h \in [-\text{nbrs}, +\text{nbrs}]$, то число возможных значений π_h^+ ограничено, и равно $1 + 2 \cdot \text{nbrs}$. Все возможные вероятности записываются в массив **prob_arr** перед началом алгоритма.

```
void Monte_Carlo::heat_bath_simulate(lattice *l, int steps) const
{
    int rand_spin, prob, nbrs = l->getnbrs(), *L = l->getL(), N = l->getN();
    int prob_arr[1 + 2 * nbrs]; // Массив всех возможных вероятностей

    for (int i = -nbrs; i <= nbrs; ++i) // Заполнение массива
        prob_arr[i + nbrs] = RAND_MAX / (1 + exp(-2 * beta * i - mu * H));

    for (int i = 0; i < steps; ++i)
    {
        for (int j = 0; j < N; ++j)
        {
            rand_spin = big_rand() % N; // Выбрать произвольный спин
            prob = prob_arr[l->sum_nbr(rand_spin) + nbrs]; // Взять высчитанную ранее вероятность
            L[rand_spin] = def_spin(prob); // Присвоить +1 или -1
        }
    }
}
```

Рис. 1: Алгоритм **heat_bath_simulate**

2.2.2 Cluster algorithm

В алгоритме **cluster_simulate** спины переворачиваются кластерами, что весьма ускоряет процесс симуляции.

Создание кластера **Cluster** начинается с выбора произвольного спина. Он кладется в **Cluster** и в дополнительный список **Pocket**. В **Pocket** хранятся спины, чьи соседи еще не обработаны алгоритмом.

Для каждого спина из **Pocket**, рассматриваются его соседи того же знака, и с вероятностью **prob** добавляются в **Cluster**.

Когда **Pocket** пуст, создание кластера прекращается, и все спины в нем меняют знак.

Вероятность **prob** берется равной:

$$\pi(\beta) = 1 - e^{-2\beta} \quad (4)$$

```
void Monte_Carlo::clusters_simulate(lattice *l, int steps) const {
    int spin, nbrs = l->getnbrs(), nbr_arr[nbrs], *L = l->getL(), N = l->getN();
    int prob = RAND_MAX * (1 - exp(-2 * beta)); // Магическое число

    for (int j = 0; j < steps; ++j)
    {
        spin = big_rand() % N; // Произвольный выбор спина
        vector<int> Cluster {spin}, Pocket {spin}; // Кладем его в кластер и в карман

        while (!Pocket.empty())
        {
            spin = Pocket[big_rand() % Pocket.size()]; // Произвольный выбор из кармана
            l->getnbrs(spin, nbr_arr); // Получить соседей спина

            for (int i = 0; i < nbrs; ++i) // Проверить всех соседей
            {
                if (L[spin] == L[nbr_arr[i]] && // Если спин соседа совпадает
                    !vcontains(Cluster, nbr_arr[i]) && // и его еще нет в кластере,
                    rand() < prob) // то с вероятностью prob
                {
                    Pocket.push_back(nbr_arr[i]); // Добавление в карман
                    Cluster.push_back(nbr_arr[i]); // Добавление в кластер
                }
            }
            vdel(Pocket, spin); // Удалить из кармана
        }
        for (auto i = Cluster.begin(); i != Cluster.end(); ++i)
            L[*i] = - L[*i]; // Переворот кластера
    }
}
```

Рис. 2: Алгоритм `clusters_simulate`

2.3 Работа алгоритмов

Для анализа работы алгоритма `heat_bath_simulate` и поведения системы можно построить график средней намагниченности **avg_magn** от обратной к температуре величины **beta**.

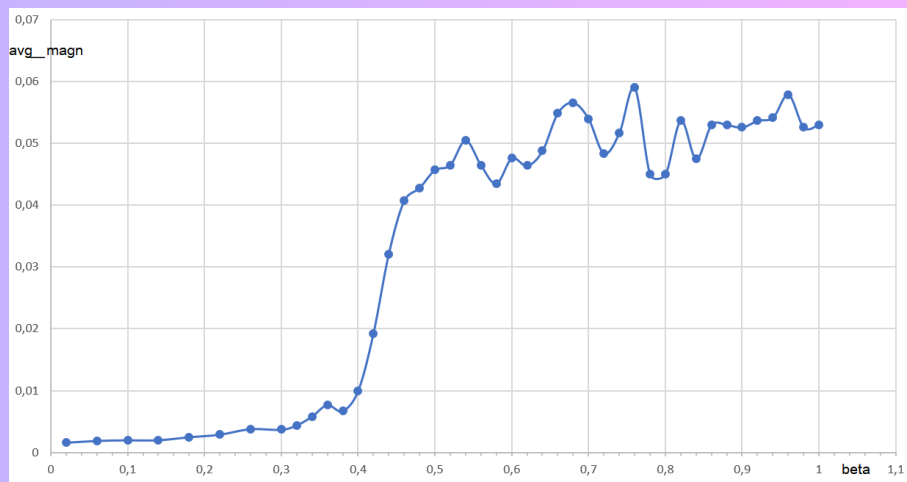


Рис. 3: График средней намагниченности **avg_magn** от **beta**

На промежутке $[0.4; 0.5]$ - фазовый переход: наблюдается резкое повышение средней намагниченности. Это говорит о том, что спины группируются в более-менее постоянные области одного знака.

С использованием алгоритма **clusters_simulate** можно подробнее изучить область, где происходит фазовый переход: На более подробном графике видно,

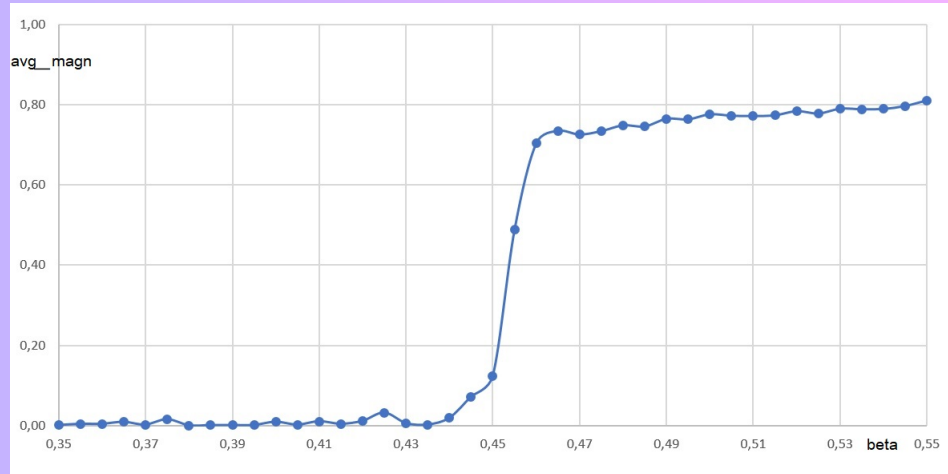


Рис. 4: График средней намагниченности avg_magn от beta

что скачок происходит при $\mathbf{beta} \in [0.44; 0.46]$, что согласуется с полученным аналитически Л. Осагнером значением

$$\beta_c = \frac{\ln(1 + \sqrt{2})}{2J} \approx 0.44$$

3 Инструкция по использованию

3.1 Технические требования

Версия Qt 5.14.1, компилятор C++ 11.

3.2 Сборка проекта

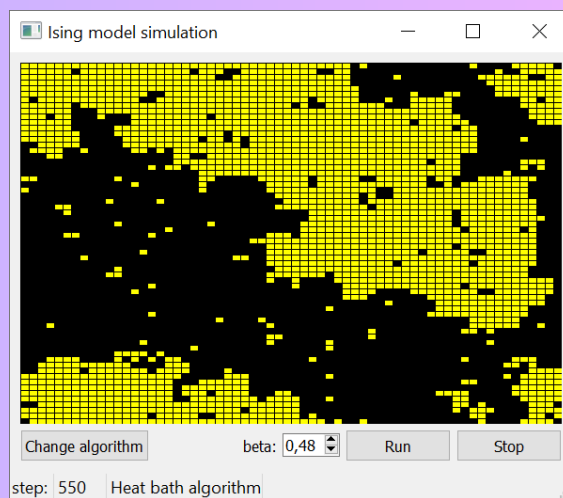
Windows 10:

1. Клонировать репозиторий с *github.com/gitrbond/ising_project*
2. Убедиться, что qmake, make и windeployqt в PATH, перейти в директорию проекта (файл .pro)
3. qmake
4. make
5. windeployqt . (при необходимости)
6. ising_model.exe

Linux:

1. qmake
2. make
3. linuxdeployqt . (при необходимости)
4. ./ising_model

3.3 Пример работы



ising_model.exe

3.4 Параметры командной строки

Скомпилированный файл `ising_model.exe` (или просто `ising_model` в linux) позволяет делать две вещи:

- Запустить симуляцию модели Изинга в графическом режиме.
Синтаксис: **`ising_model.exe [option]`**
При запуске без параметров размер решетки по умолчанию будет 64×64 (пример на картинке выше). Использование опции **`-lsize=<number>`** позволяет задать размер до запуска.
- Построить график средней намагниченности решетки от температуры β .
В каждом измерении спины вначале ориентированы произвольно, и средняя намагниченность берется через `steps` шагов алгоритма.
Синтаксис: **`ising_model.exe -plot input output [plot options]`**
Точки для построения графика (значения β) берутся из файла с именем `input`, а результаты (средняя намагниченность `avg_magn`) записываются в файл `output`. Формат входа: сначала количество точек n , затем n значений. Размер решетки можно выбрать опцией **`-lsize=<number>`**.
Используемый алгоритм выбирается опцией **`-algo=<1|2>`**, где 1 - Heat bath algorithm, 2 - Clusters algorithm.
Число шагов `steps` алгоритма задается с помощью **`-steps=<number>`**.
Рекомендуется также использовать усреднение результатов. Каждая точка усредняется `averaging` число раз, его можно задать опцией **`-avg=<number>`**. При построении обоих графиков выше усреднение происходило по 50.

Пример команды:

`ising_model.exe -plot input.txt output.txt -steps=200 -algo=1 -avg=50`

Пример входных данных:

```
5 0.3 0.4 0.5 0.6 0.7
```

Краткая справка по использованию:

`ising_model.exe -help`

4 Реализация

4.1 Устройство классов

parameters - класс, в котором хранятся основные параметры модели.

Monte_Carlo - класс с алгоритмами для симуляции, наследуется от **parameters**.

lattice - абстрактный класс, содержащий в себе решетку спинов, и методы для работы с ней. Имеет два чисто виртуальных метода, которые необходимо отдельно реализовать для каждого из типов решетки: линейной одномерной, двухмерной прямоугольной и квадратной, трехмерной произвольного размера и кубической. У всех решеток циклические граничные условия (это значит, например, что прямоугольная решетка расположена на торе).

Обозначения:

— - private

- protected

+ - public

Курсивом выделены абстрактный класс **lattice** и его виртуальные методы.

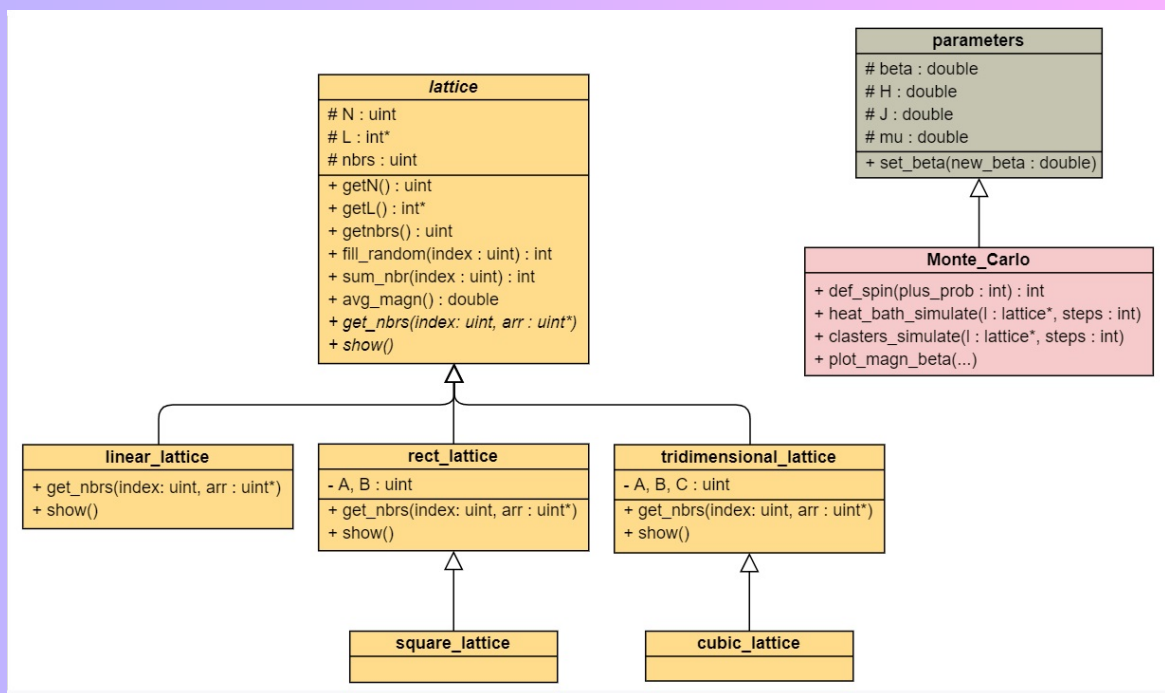


Рис. 5: Схема наследования классов

4.2 Взаимодействие модулей

В GUI-приложении есть пять QObject-ов, общающихся между собой сигналами:

MainWindow - Главный поток исполнения программы. Окно приложения.

paintWidget - Холст для рисования решетки на экране.

ui - Пользовательский интерфейс (кнопки, надписи, счетчики)

thread - Вторичный поток с симуляцией модели.

worker - Класс в потоке **thread**, в методе которого происходит симуляция.

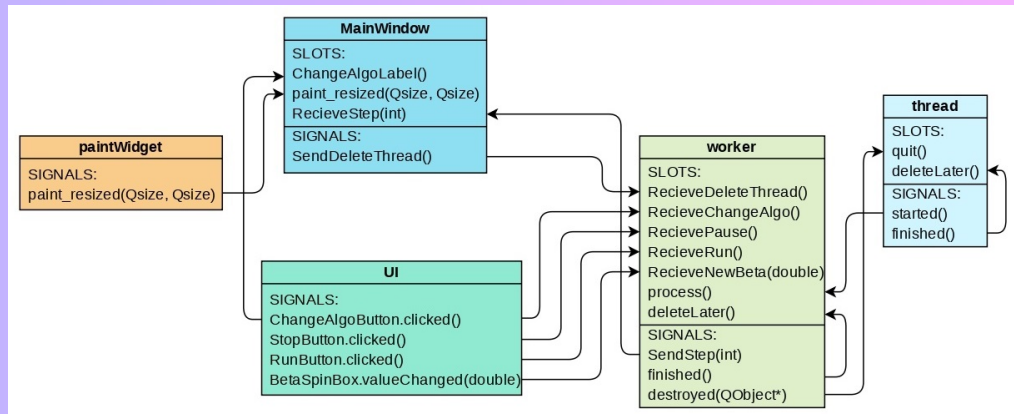


Рис. 6: Схема связей между объектами QObject

На картинке ниже обрисованы линии жизни четырех QObject-ов во время работы приложения, а также цепочки событий при различных действиях пользователя. Таймлайн идет сверху вниз.

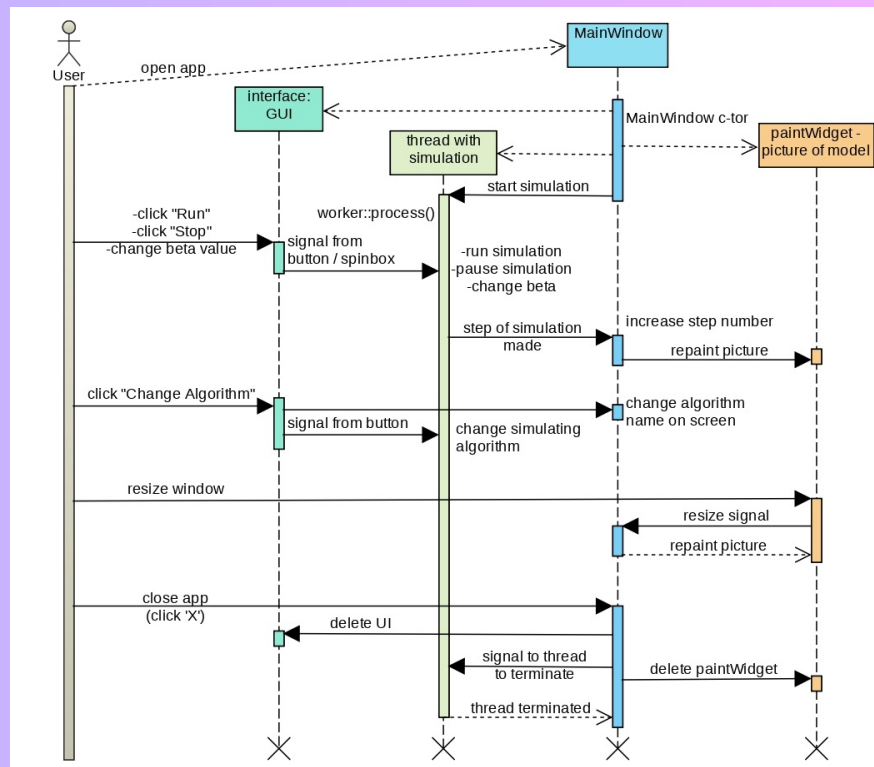


Рис. 7: Диаграмма активностей GUI-приложения

4.3 Распределение обязанностей

- Бондарь Роман → Автор идеи и структуры проекта; Реализация алгоритмов; Создание графического интерфейса; Построение графиков; Написание подробной документации и help-а.
- Пяткин Станислав → Автоматизация вычисления заданных точек для графика; Исследование полученных графиков и подбор параметров вычисления для получения наиболее близкого к теории графика; Обсчет модели на большой решётке; Создание 3D решётки; Оптимизация кода алгоритма Heat bath.
- Семененко Александр → Создание документации; Имплементация разных типов решеток; Поддержка безопасности работы программы; Реализация типовой решётки; Подбор теоретической справки.

5 Средства разработки

Язык программирования C++. Среды: CodeBlocks, CLion, Visual Studio.
Графический интерфейс реализован с помощью Qt в среде QtCreator.

Список источников

- [1] Werner Krauth : Statistical Mechanics : Algorithms and Computations
- [2] Jesper Jacobsen, Stephane Ouvry : Exact Methods in Low-dimensional Statistical Physics and Quantum Computing - 2008
- [3] Wikipedia : Ising model
- [4] А.В.Третьяков : drawing 6.4E1 (A simple GUI popular-scientific drawing framework)