Санкт-Петербургский государственный университет
Физический факультет
Кафедра вычислительной физики

Разработка Java-конструктора моделей электрических цепей постоянного и переменного электрического тока

Выпускная квалификационная работа бакалавра
_____ **Тюркиной Александры Германовны**

Научный руководитель:
_____ к. ф.-м. н., доц. **Монахов В.В.**

Рецензент:

_____ к. ф.-м. н., доц. **Смирнов А.В.**

Санкт-Петербург
2018

Оглавление

Введение

Программный комплекс BARSIC

Программный комплекс BARSIC включает в себя язык программирования под тем же названием BARSIC, который является предметно-ориентированным языком программирования (DSL - Domain-Specific Language), то есть языком, специализированным для конкретных задач. Основным применением BARSIC является программная поддержка моделей интернет-олимпиады по физике. Данные модели имитируют реальные физические эксперименты, с помощью которых проверяется практическое применение полученных знаний [1].

Постановка задачи

Исполняющая среда BARSIC, под управлением которой работают модели виртуальных лабораторий, предназначена для работы под ОС Windows. Для работы под ОС Linux необходимо использовать эмулятор Windows API (например, VirtualBox или Wine). Конечно, большинство участников заходят на сайт интернет-олимпиады с устройств под операционной системой Windows (~67%), но, учитывая массовость интернет-олимпиады школьников (около 40 тыс. участников в год), а также нарастающую популярность планшетных компьютеров, крайне необходимо создание модели для платформ Android и iOS [1].

На момент написания работы уже имеется работающий прототип проигрывателя BARSIC для платформы Android, а также набор конструкторов для моделей устройств, таких как мультиметр. источник постоянного тока, резистор и т.д.

Целью данной работы является дополнения существующего набора конструкторов, а именно:

- Создание моделей источника переменного тока, катушки индуктивности и конденсатора:
- Графическая реализация вышеперечисленных устройств

1 Краткий обзор ранее полученных результатов

При разработке конструктора цепей моделей электрических цепей требуется решить следующие задачи:

- Построение математической модели расчета электрических цепей
- Построение иерархии классов, отвечающих за бизнес-логику приложения и иерархию классов, отвечающих за визуализацию (согласно принципу разделения бизнес логики и визуализации)

Фрагменты вышеописанных задач были решены студентами прошлых годов, а именно:

- Была разработана математическая модель для расчета цепей постоянного тока
- Была построена иерархия классов устройств
- Были построены классы, реализующие логику расчетов электрических цепей постоянного тока

Ниже представлен краткий обзор решений данных задач.

Для более детального рассмотрения реализации решений можно обратиться к работам Фриша В.С. [2] и Файзулина Е.Э. [3].

1.1 Математическая модель

Для расчета электрических цепей был реализован метод узловых потенциалов [2].

Основная идея заключается в построении систем уравнений в матричном виде, составленных на основании законов Ома и Кирхгофа:

$$AYA^tU_0 = -A(J + YE)$$

где:

A - матрица соединений, элементы которой характеризуют топологию электрической цепи

Y – диагональная матрица проводимостей

U_0 – матрица-столбец узловых потенциалов

J – матрица-столбец источников тока

E – матрица столбец источников ЭДС

Выбор метода обусловлен его достоинствами, таких как простота в реализации решения системы уравнений и простота в построении компьютерной модели.

1.2 Реализация компьютерной модели

1.2.1 Модели устройств

Как уже было сказано выше, была построена иерархия классов устройств. На Рис. 1 представлена UML-диаграмма классов.

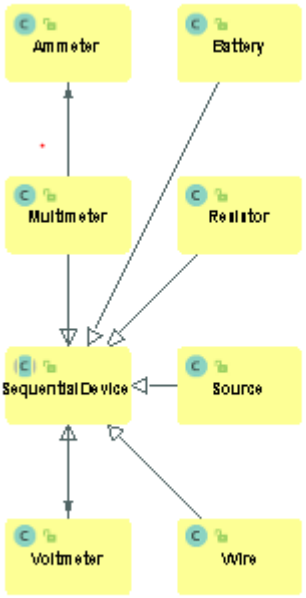

Рис. 1. UML-диаграмма основных классов устройств

Эти классы являются логическими, т.е. содержат только необходимые поля - параметры устройств и необходимые геттеры/сеттеры – методы для

получения/установления значений полей. Все поля и методы наследуются от абстрактного класса-родителя `SequentialDevice`, который, в свою очередь, является наследником абстрактного класса `Device`, который и содержит все параметры, характерные для устройств (например, внутреннее сопротивление устройства) и соответствующие геттеры/сеттеры.

Также была построена иерархия классов, отвечающих за графическое отображение соответствующих устройств. Не вдаваясь в подробности, эта иерархия классов выглядит аналогично представленной выше. Говоря кратко, данные классы содержат всю логику отрисовки устройств, а также методы, которые будут вызываться при определенных действиях (например, касание сенсорного экрана). Каждый из этих классов наследует класс `PDevice`, в котором содержится поле `SequentialDevice` для осуществления связи между “логическими” и “графическими классами” [3].

1.2.2 Построение матриц и решение системы уравнений

Логика расчета электрических цепей постоянного тока была реализована в следующих классах [2]:

- `Solver` – класс с набором статических методов (использование данных методов не требует инстанцирования) для решения системы уравнений в матричном виде.
- `Scheme` – содержит поля connects (коллекция `ArrayList`, содержащая объекты типа Connect, представляющие собой узлы электрической цепи) и `devices`(коллекция `ArrayList` добавленных устройств), а также методы, с помощью которых происходит итерирование по коллекциям и заполнение матриц. Далее вызывается метод из класса `Solver`, в качестве аргументов передаются полученные матрицы (представляющие собой массивы), а возвращается массив, элементами которого являются потенциалы в соответствующих узлах.

Стоит отметить, что такие операции, как перемножение матриц считались вручную, т.е. для этого были реализованы соответствующие методы.

Позднее, студентом 3 курса Розвезевым Владиславом, в рамках курсовой работы, была переписана логика решения системы уравнений с использованием Apache Commons Math[7]. Данный API содержит большое количество методов и классов для решения широкого спектра задач, в том числе задач линейной алгебры. Данная модификация значительно упростила расчеты, делая их более прозрачными и компактными, а главное – были исправлены некоторые ошибки.

Выводы по главе 1

Был приведен использующийся метод для расчета электрических цепей постоянного тока, а также краткий обзор набора классов, которые использовались для решения задач в данной работе.

2 Конструктор модели электрической цепи переменного тока

На основе результатов, описанных в главе 1 требовалось модифицировать расчеты таким образом, чтобы можно было производить расчеты цепей переменного тока, а также добавить классы устройств, характерных для цепи переменного тока.

2.1 Разработка класса модели генератора переменного тока

2.1.1 Класс `Alternator`, отвечающий за логику

В соответствии с существующей структурой, были разработаны классы Alternator, наследник абстрактного класса `SequentialDevice` – “логический” класс и “графический” класс `PAlternator`, наследник класса `PDevice`.

Абстрактный класс `Device` является суперклассом для всех устройств и содержит все необходимые поля и методы. Соответственно, в класс `Alternator` были лишь добавлены конструктор без параметров, внутри которого полю `resistance` (внутреннее сопротивление) устанавливается сопротивление по умолчанию (0.001 Ом), а также конструктор с параметром, внутри которого полю `resistance` присваивается значение, переданное в качестве параметра. Также, в дальнейшем будет использоваться унаследованное поле `voltage` (напряжение).

Кроме того, в класс `Device` были добавлены поля `inductance` (индуктивность) и `capacity` (емкость), для дальнейшей реализации моделей катушки индуктивности и конденсатора.

2.1.2 Графическая реализация генератора переменного тока

Требовалось разработать класс, отвечающий за отрисовку устройства генератора переменного тока.

Графическое представление данного устройства в системе виртуальных лабораторий BARSIC представлено на Рис. 2.

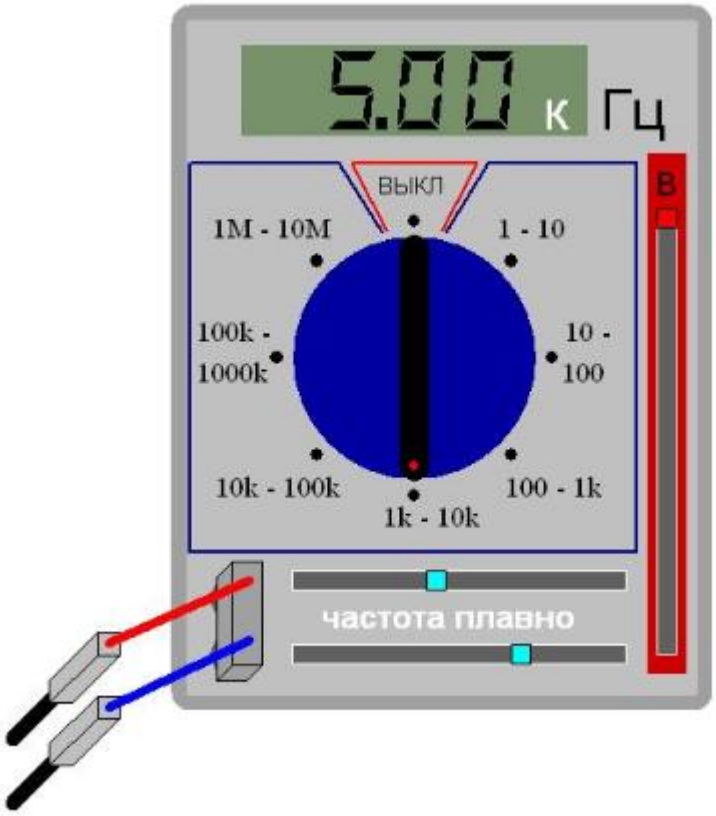

Рис. 2. Внешний вид генератора переменного тока

Был разработан класс `PAlternator`, расширяющий класс `PDevice` и имплементирующий интерфейс `IIndicative` (классы, имплементирующие этот интерфейс, характеризует устройства, отображающие состояние цепи).

Для рисования графических объектов на платформе Android используются классы из пакета android.graphics. Класс `Canvas` из данного пакета позволяет рисовать различные объекты (линии, круги, прямоугольники и др.)[4]. При вызове соответствующего метода, кроме необходимых параметров фигуры, которую требуется нарисовать, также передается ссылка на объект типа `Paint` (тоже из пакета android.graphics), поля которого содержат информацию о цвете, толщине линии и др. (т.е. таким образом задаются параметры “кисти”)[5].

Все “графические” классы переопределяют метод paint, в который передается ссылка на объект типа Canvas, нужный для отрисовки. Учитывая тот факт, что код, отвечающий за рисование графического объекта, является достаточно громоздким в силу того, что внешний вид устройства относительно сложен для рисования, было принято решение вынести логику отрисовки частей устройства в отдельные методы, для увеличения прозрачности кода. Таким образом, в методе `paint()` последовательно вызываются методы отрисовки и при желании, можно легко найти, в какой части кода что рисуется. На листинге 1 представлен фрагмент кода, демонстрирующий интуитивность метода `paint()`, осуществляющего отрисовку всех частей устройства.

Листинг 1. Метод отрисовки устройства

```
@Override
public void paint(Canvas canvas) {
    drawOuterContainer(canvas);
    drawInnerContainer(canvas);
    drawVoltageScrollContainer(canvas);

        //…..
}
```

Стоит уделить особое внимание таким частям как переключатель частоты, который регулирует интервал, в котором с помощью “ползунков” будет задаваться значение частоты переменного тока.

Было решено использовать структуру данных `Map` как для упрощения отрисовки, так и для упрощения получения интервала частот (а также ввиду того, что временная сложность выборки элемента из `HashMap` – O(1) (кроме случая вырождения `HashMap` в связный список, но в данном контексте это невозможно)).

Ниже представлено декларация и инициализация поля, с помощью которого можно за константное время получить интервал значений частот, соответствующий положению переключателя.

```
private Map<Integer, ValueInterval> switcherValues = new HashMap<>();
```

Здесь `ValueInterval` – класс, который содержит 2 поля – значения левого и правого концов интервала.

Аналогичные поля были введены для получения расположения меток, соответствующих положению переключателя, благодаря чему отрисовка деталей переключателя частот сводится к выполнению простого цикла, представленного на Листинге 2.

Листинг 2. Отрисовка переключателя диапазона частот

```
for (int index = 0; index < switcherModesCount; index++) {
    canvas.drawText(switcherLabels.get(index),
            switcherLabelsPositions.get(index).width,
            switcherLabelsPositions.get(index).height,
            paint
    );
    canvas.drawCircle(switcherMarks.get(index).width,
            switcherMarks.get(index).height,
            5,
            paint
    );
}
```

Таким образом, было реализована отрисовка генератора переменного тока (см. Приложение А).

Ниже представлен полученный результат:

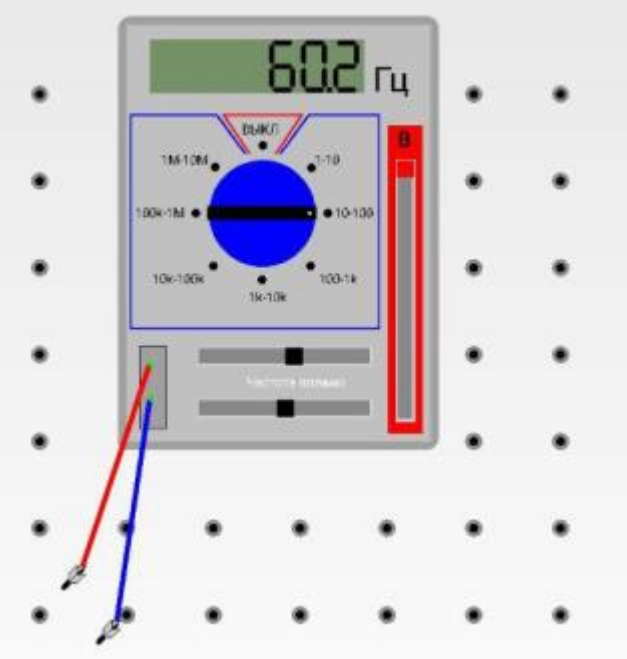

Рис. 3. Внешний вид реализованного генератора переменного тока

2.2 Получение необходимых значений

Выше было упомянуто, что для реализации связи между “логическим” и “графическим” классами, в родительском “графическом” классе `PDevice` имеется поле, содержащее ссылку на соответствующий объект “логического” класса `Device`.

В конструкторе класса `PAlternator` происходит инициализация унаследанного поля `device`. Таким образом, чтобы получить заданное напряжение (которое задается с помощью “ползунка” в правой части устройства), требуется повесить на “ползунок” так называемый listener. Это можно сделать с помощью вызова следующего метода

```
voltageScroll.addOnTouchListener(listener);
```

где `listener` – ссылка на объект класса, имплементирующего интерфейс `OnTouchListener ()` и у которого переопределен метод `onTouch()`. Код, который находится внутри этого метода будет выполнен, когда наступит соответствующее событие, в данном случае это событие-касание пальцем.

Листинг 3. OnTouchListener для регулятора напряжения

```
@Override
public boolean onTouch(View view, MotionEvent event) {
    double currentVoltage = calculateVoltage();
    device.setVoltage(currentVoltage);
    ListsOfDevices.updateScheme();
    return true;
}
```

На Листинге 3 представлен расчет напряжения на основании положения “ползунка”, а также в поле `voltage` объекта `device` записывается рассчитанное значение.

Для получения же значения частоты генерируемого переменного тока требуется сеять показания с переключателя, а также с обоих “ползунков” (один из которых позволяет регулировать частоту в “грубом” режиме, а второй в “плавном”). Кроме того, полученное значение требуется отобразить на циферблате.

Аналогично “ползунку”, регулирующему напряжение, следует добавить listener-ы к вышеуказанным объектам. На Листинге 4 представлен метод `onTouch`(), который выполняется при переключении диапазона частот.

Листинг 4. OnTouchListener для переключателя диапазона частот

```
@Override
public boolean onTouch(View view, MotionEvent event) {
    if (switcher.getState() == 0) {
        display.switchOff();
        Scheme.setFrequency(0);
        ListsOfDevices.updateScheme();
    } else {
        ValueInterval interval =
```

```
switcherValues.get(switcher.getState());
            setMinFrequency(interval.getMinValue());
            setMaxFrequency(interval.getMaxValue());
            setUnit();
            setAccuracy();
            Scheme.setFrequency(calculateFrequency());
            setDisplayFrequency();
            ListsOfDevices.updateScheme();
        }
        return true;
    }
```

На Листинге 5 представлен метод `onTouch`(), который выполняется при регуляции частоты.

Листинг 5. OnTouchListener для регулятора частоты

```
@Override
public boolean onTouch(View view, MotionEvent event) {
    Scheme.setFrequency(calculateFrequency());
    setDisplayFrequency();
    ListsOfDevices.updateScheme();
    return true;
}
```

При переключении диапазона частот происходит проверка на “нулевое” положение (т.е. положение “выкл”). Если переключатель находится в таком положении, циферблат очищается (выключается), частоте присваивается значение 0 и цепь пересчитывается. В противном случае, меняется диапазон частот, на основании положения “ползунков” для регулирования частоты пересчитывается итоговая частота, меняется значение на циферблате и цепь также пересчитывается. Процесс расчета цепей будет пояснен позднее.

При регулировании частоты с помощью “ползунков” просто пересчитывается частота, меняется показание циферблата и цепь пересчитывается.

2.3 Добавление классов `Inductor` и `Capacitor`

Процесс разработки классов для катушки индуктивности и конденсатора ничем не отличается от процесса, описанного выше. Стоит напомнить, что в класс Device были добавлены поля inductance и capacity, поэтому в “графических” классов `PInductor` и `PCapacitor`, после инициализации переменной device, можно полям inductance и capacity задать значение. Отрисовка данных графических объектов ничем не отличается от отрисовки уже имеющихся на момент начала работы резисторов. В качестве примера, класс `PCapacitor` представлен в Приложении Б.

2.4 Модификация алгоритма расчета электрической цепи постоянного тока

Вернемся к классу Scheme, назначение которого описывалось в главе 1.

Из вышеприведенных фрагментов расчета частоты переменного тока, обратим внимание на метод `Scheme.setFrequency(double frequency)`. Данный метод задает значения статического поля `frequency` класса `Scheme`. Поэтому при вызове метода расчета имеются все необходимые параметры для построения матриц, необходимых для решения системы уравнений.

Идея адаптирования существующих расчетов цепи постоянного тока к цепям переменного тока заключается в переходе к комплексным значениям токов/потенциалов.

Как известно, в случае цепей переменного тока, реактивные элементы цепи (катушка индуктивности и конденсатор) проявляют реактивные свойства. Реактивные элементы на переменном токе ведут себя как элементы с конечным сопротивлением (такое сопротивление называется импедансом).

Для катушки индуктивности величина импеданса рассчитывается по формуле

$$X_L = i\omega L \qquad (1)$$

для конденсатора

$$X_C = \frac{1}{i\omega C} \qquad (2)$$

Для резистора же величина импеданса не зависит от частоты.

На Листинге 5 представлен метод, в котором происходит заполнение матриц для решения системы уравнений, написанный студентом 3 курса Розвезевым Владиславом, в рамках курсовой работы [7].

Листинг 6. Заполнение матриц

```
public void createMatrix1() {
    int q = connects.size(); // количество узлов в цепи
    int p = devices.size(); // количество рёбер в цепи
    A = new double[q - 1][p];
    Y = new double[p][p];
    J = new double[p];
    E = new double[p];
    U = new double[q - 1];
    for (int i = 0; i < p; i++){
        Y[i][i] = 1.0 / devices.get(i).getResistance();
        J[i] = devices.get(i).getAmperage();
        E[i] = devices.get(i).getVoltage();
        for (int j = 0; j < (q - 1); j ++){
            if (devices.get(i).getCon1() == connects.get(j)) {
                A[j][i] = 1.0;
            } else if (devices.get(i).getCon2() ==
connects.get(j)) {
                A[j][i] = -1.0;
            } else {
                A[j][i] = 0.0;
```

```
                }
            }
        }
}
```

Как видно, в цикле происходит итерирование по всем узлам и ребрам (добавленным устройствам) цепи. Вспомним, что в класс `Device` были добавлены поля `inductance` и `capacity`, что делает возможным при итерировании сделать проверку на значение данных полей и если они окажутся ненулевыми, то рассчитать емкостное, либо индуктивное сопротивление по формулам (1)-(2) и записать значение проводимости в соответствующий элемент матрицы проводимостей.

Возникает проблема: данное значение является комплексным, а в Java, как известно, нет реализации для комплексных чисел. Решением данной проблемы стало использование уже упомянутой библиотеки Apache Commons Math, в которой содержатся необходимые классы, реализующие комплексные числа.

Рассмотрим классы, которые были использованы.

org.apache.commons.math3.complex.`Complex` – класс комплексных чисел, который также содержит все необходимые методы для работы с ними, такие как

- Complex add(Complex c) – возвращает сумму комплексных чисел
- Complex sub(Complex c) – возвращает разность комплексных чисел
- Complex abs() – возвращает модуль комплексного числа

На Листинге 7 представлен фрагмент решения системы уравнений, реализованное студентом 3 курса.

Листинг 7. Решение системы уравнений

```
    public static double[] solveLinearSystem(double[][] a,
double[][] y, double[] j, double[] e){
```

```
        //…
        RealMatrix A = new Array2DRowRealMatrix(a);
        RealMatrix Y = new Array2DRowRealMatrix(y);
        RealVector J = new ArrayRealVector(j);
        RealVector E = new ArrayRealVector(e);
        RealMatrix coefficients =
A.multiply(Y).multiply(A.transpose());
        RealVector constants = A.operate(J.add(Y.operate(E)));
        DecompositionSolver solver = new
LUDecomposition(coefficients).getSolver();
        RealVector solution = solver.solve(constants);
        return solution.toArray();
        //…
    }
```

Как видно, используются матрицы действительных чисел, требуется перейти к матрицам комплексных чисел. Для этого были использованы классы из пакета org.apache.commons.math3.linear, позволяющие дженерифицировать классы матриц.

На Листинге 8 представлен фрагмент модифицированного метода для заполнения матрицы проводимостей.

Листинг 8. Заполнение матриц в случае цепи переменного тока

```
currentDevice = devices.get(i);
double angularFrequency = 2 * Math.PI * Scheme.getFrequency();
if (Double.compare(currentDevice.getInductance(), 0.0) != 0) {
    // inductive resistance = i*w*L, where w-angular frequency,
L-inductivity
    Complex inductiveResistance = new Complex(0.0,
```

```
            angularFrequency * currentDevice.getInductance());
    Y[i][i] = inductiveResistance.pow(-1);
} else if (Double.compare(currentDevice.getCapacity(), 0.0) !=
0) {
    // capacitive resistance = 1/(i*w*C), where w-angular
frequency, C-capacity
    if (Double.compare(angularFrequency, 0.0) != 0) {
        Y[i][i] = new Complex(0.0, angularFrequency *
currentDevice.getCapacity());
    } else {
        Y[i][i] = new Complex(0.0, 0.0);
    }
} else {
    Y[i][i] = new Complex(currentDevice.getResistance(),
0.0).pow(-1);
}
for (int k = 0; k < p; k++) {
    if (k != i) {
        Y[i][k] = new Complex(0.0, 0.0);
    }
}
```

Можно заметить, что при расчетах также учитывается тот факт, что при постоянном токе сопротивление конденсатора стремится к бесконечности.

На Листинге 8 представлен модифицированное решение системы уравнений.

Листинг 9. Решение системы уравнений в случае цепи переменного тока

```
public static Complex[] solveLinearSystem(Complex[][] a,
Complex[][] y, Complex[] j, Complex[] e) {
```

```
        //……
        FieldMatrix<Complex> A = new
Array2DRowFieldMatrix<>(a);
        FieldMatrix<Complex> Y = new
Array2DRowFieldMatrix<>(y);
        FieldVector<Complex> J = new ArrayFieldVector<>(j);
        FieldVector<Complex> E = new ArrayFieldVector<>(e);
        FieldMatrix<Complex> coefficients =
A.multiply(Y).multiply(A.transpose());
        FieldVector<Complex> constants =
A.operate(J.add(Y.operate(E)));
        FieldDecompositionSolver<Complex> solver = new
FieldLUDecomposition<>(coefficients).getSolver();
        FieldVector<Complex> solution =
solver.solve(constants);

        return solution.toArray();

        //……
    }
```

Можно увидеть, что до изменения метода, в котором происходит решение системы уравнений, возвращался массив действительных чисел и поле `potential` в классе `Connect` являлось значением типа `double`. Теперь же возвращается массив комплексных чисел и тип поля potential был изменен на тип `Complex`. Таким образом, при расчете разности действительных потенциалов теперь считается разность комплексных потенциалов с последующим расчетом модуля полученного значения.

На Листинге 10 представлен метод получения значения амперметра (в классе `Ammeter`).

Листинг 10. Расчет силы тока

```
public double getValue() {
    if (getConnection1() != null && getConnection2() != null) {
        return
(getConnection1().getPotential().subtract(getConnection2().getP
otential())).abs()
                / getResistance();
    }else{
        return 0;
    }
}
```

С полной версией модифицированного метода, к котором происходит заполнение матриц можно ознакомиться в Приложении Б.

Выводы по главе 2

С помощью Apache Commons API удалось адаптировать расчеты для электрических цепей постоянного тока к цепям переменного тока.

3 Проверка полученных результатов

Проверка была осуществлена на устройстве Xiaomi Redmi Note 4 под управлением OS Android версии 6.0 с разрешением экрана 1920x1080.

3.1 Проверка расчетов цепи постоянного тока

Для начала, стоит убедиться, что расчеты для цепи постоянного тока остались верными. Для проверки используется закон Ома

$$I = \frac{U}{R}$$

Была собрана следующая простая цепь (Рис. 4).

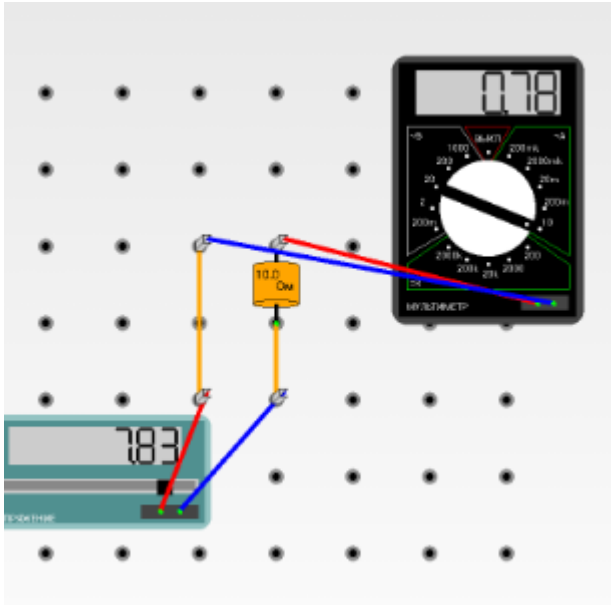

Рис. 4. Простая цепь постоянного тока для проверки универсальности расчетов цепей

Полученное показание амперметра совпадает с теоретическим.

$$I = \frac{7.83\ \text{В}}{10\ \text{Ом}} = 0.783\ A \simeq 0.78\ A$$

Далее требуется проверить случай электрической цепи с подключенным конденсатором (Рис. 5). Как уже упоминалось, при постоянном токе, сопротивление конденсатора стремится к бесконечности и значение на амперметре должно быть равно 0.

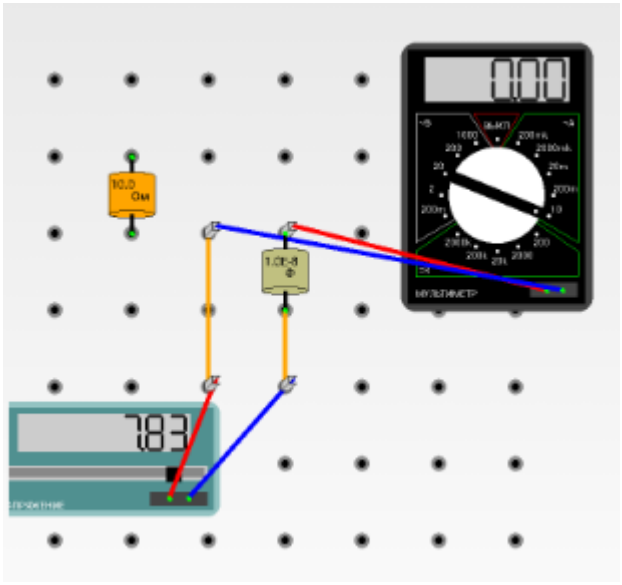

Рис. 5. Проверка отсутствия тока в цепи постоянного тока с подключенным конденсатором

Как видно, показание амперметра согласуется с теорией.

3.2 Проверка расчетов цепи переменного тока

Для расчетов силы тока в цепи переменного тока использовались следующие формулы:

$$I = U\omega C$$

$$I = \frac{U}{\omega L}$$

где:

ω – частота переменного тока

L - индуктивность катушки

C – емкость конденсатора

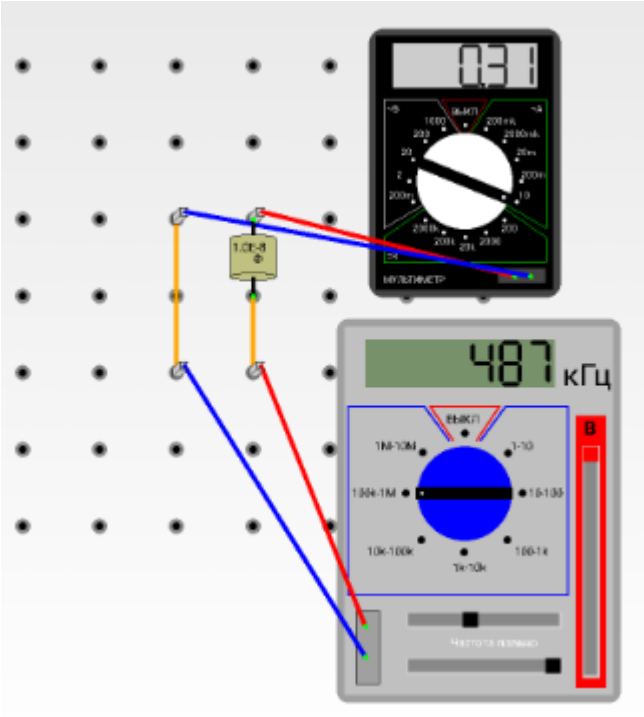

Рис. 6. Проверка расчетов тока в цепи переменного тока с подключенным конденсатором

$$I = U\omega C = U2\pi fC = 10\text{В} \cdot 2 \cdot \pi \cdot 487 \cdot 10^3\text{Гц} \cdot 10^{-8}\Phi = 0.305836\ A \approx 0.31A$$

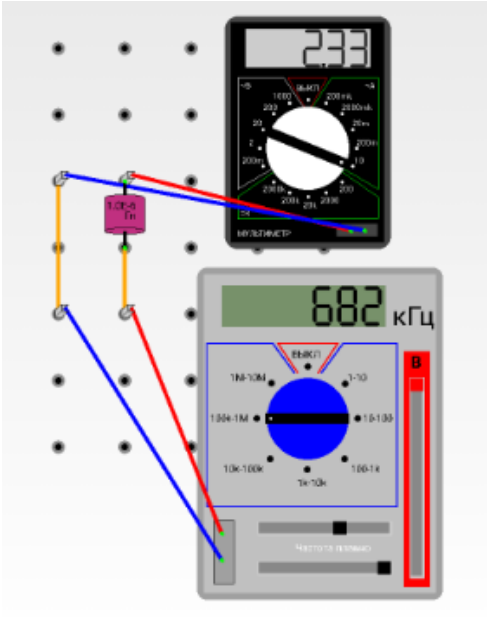

Рис. 7. Проверка расчетов тока в цепи переменного тока с подключенной катушкой индуктивности

$$I = \frac{U}{\omega L} = \frac{U}{2\pi fL} = \frac{10\text{ В}}{2 \cdot \pi \cdot 682 \cdot 10^3\text{ Гц} \cdot 10^{-6}\text{ Гн}} \approx 2.33365A \approx 2.33A$$

Также была исследована зависимость силы тока от частоты в цепи с конденсатором, представленной на Рис. 6. Результаты представлены в Таблице 1.

Таблица 1. Исследование зависимости силы тока от частоты в цепи с конденсатором ёмкостью 10^{-8} *Ф*

Частота переменного тока, Гц	**Показания мультиметра, А**
1	$0.6 \cdot 10^{-6}$
10	$6.3 \cdot 10^{-6}$
10^2	$62.8 \cdot 10^{-6}$

10^3	$0.63 \cdot 10^{-3}$
10^4	$6.28 \cdot 10^{-3}$
10^5	$62.8 \cdot 10^{-3}$
10^6	0.63
10^7	6.28

В Таблице 2 представлены результаты исследования зависимости силы тока от частоты для цепи с катушкой индуктивности, представленной на Рис. 7.

Таблица 2. Исследование зависимости силы тока от частоты в цепи с катушкой с индуктивностью $10^{-6} Гн$

Частота переменного тока, Гц	**Показания мультиметра, А**
1	Зашкаливание
10	Зашкаливание
10^2	Зашкаливание
10^3	Зашкаливание
10^4	159.07
10^5	15.92
10^6	1.59
10^7	0.16

На основании Таблиц 1 и 2 можно сделать вывод о том, что значения получаются с ограниченной точностью и хотя переключатель для задания точности позволяет получить значение с меньшей погрешностью, данные виртуальные устройства не позволяют получить (отобразить) слишком маленькие или слишком большие значение, происходит зашкаливание (как показано в Таблице 2), что накладывает некоторые ограничения на диапазон возможных параметров устройств.

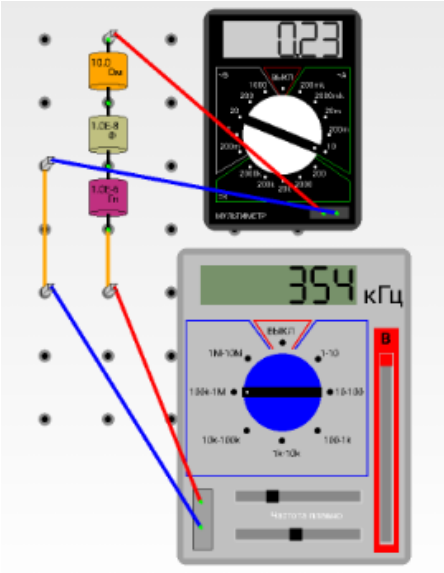

Рис. 7. Проверка расчетов тока в более сложной цепи с последовательно подключенными резистором, катушкой индуктивности и конденсатором

Закон Ома для RLC-контура, представленном на Рис. 7 [6]:

$$I = \frac{U}{Z}\text{, где } Z = \sqrt{R^2 + (\omega L - \frac{1}{\omega C})^2} - \text{полное сопротивление цепи}$$

$$I = \frac{U}{\sqrt{R^2+\left(\omega L-\frac{1}{\omega C}\right)^2}} = \frac{10\ \text{В}}{\sqrt{10^2\,\text{Ом}+\left(2\cdot\pi\cdot 354\cdot 10^3\,\text{Гц}\cdot 10^{-6}\ \text{Гн}-\frac{1}{2\cdot\pi\cdot 354\cdot 10^3\,\text{Гц}\cdot 10^{-8}\,\Phi}\right)^2}}$$

$$= 0.227847\ \text{A} \approx 0.23\ \text{A}$$

Проверим правильность расчетов напряжений. Для этого сначала измерим ток (Рис. 8), а затем, зная ток, параметры элементов цепи и частоту переменного тока, сравним показания мультиметра в режиме измерения напряжения на каждом из элементов в цепи с теоретическими расчетами (Рис. 9-11).

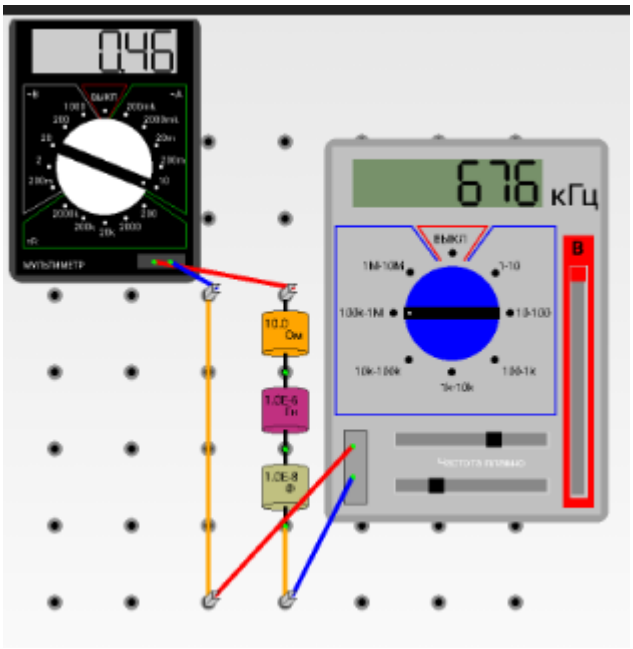

Рис. 8. Измерение тока для дальнейшей проверки значений напряжения

$$I = 0.46\ A, \omega = 2 \cdot \pi \cdot f = 4247433.27\ \frac{\text{рад}}{\text{с}}, C = 10^{-8}\text{Ф}, L = 10^{-6}\text{Гн},$$

$$Z_L = \omega L \approx 4.25\ \text{Ом}, Z_c = \frac{1}{\omega C} \approx 23.543631\ \text{Ом}$$

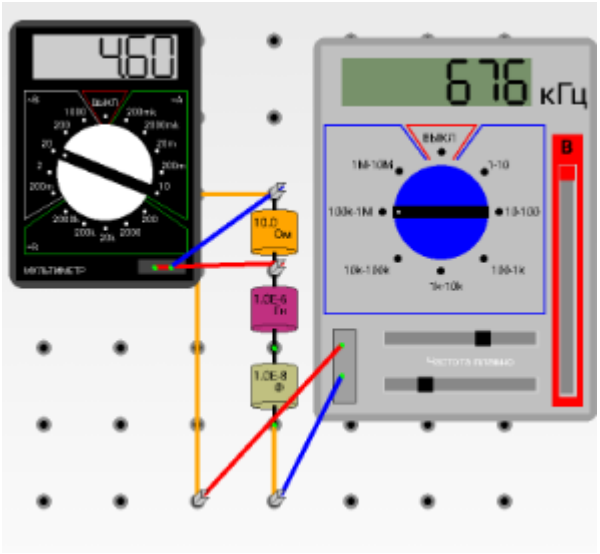

Рис. 9. Проверка значения напряжения на резисторе

$$U_R = IR = 0.46A \cdot 10\ \text{Ом} = 4.6\ \text{В}$$

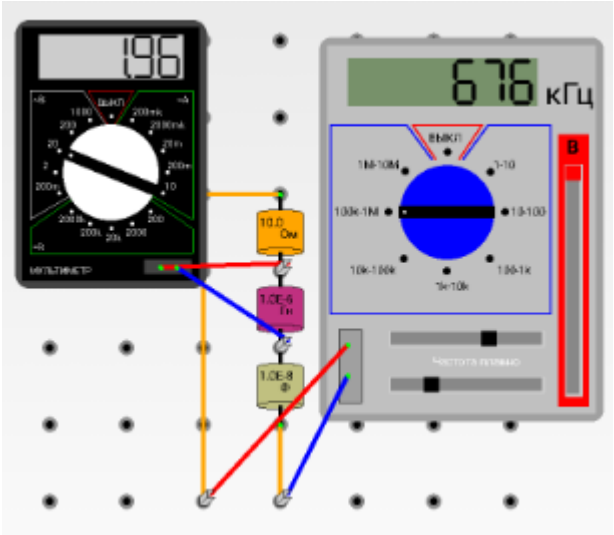

Рис. 10. Проверка значения напряжения на катушке

$$U_L = IZ_L = 0.46A \cdot 4.25\ \text{Ом} = 1.955\ \text{В}$$

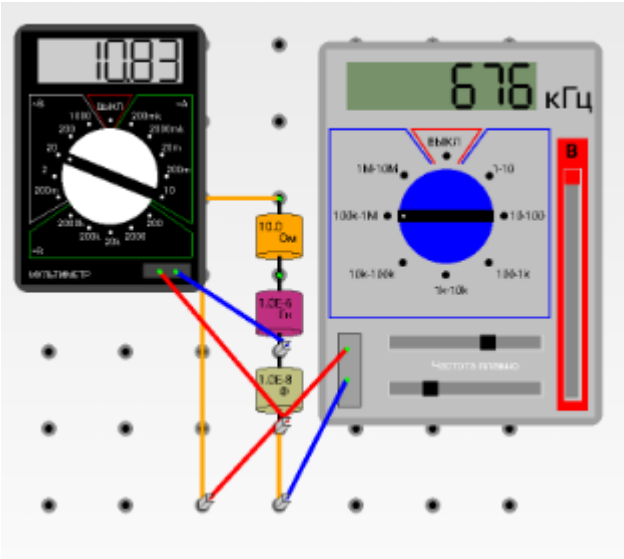

Рис. 11. Проверка значения напряжения на конденсаторе

$$U_R = IZ_C = 0.46A \cdot 23.543631\ \text{Ом} = 10.8301\ \text{В}$$

Все напряжения совпадают с теоретическими значениями.

Проверим явление последовательного резонанса (резонанса напряжений), возникающее при частоте источника напряжения равной частоте колебательного контура в RLC-контуре (Рис. 12)[8]($\omega = \frac{1}{\sqrt{LC}} = \frac{1}{\sqrt{10^{-6}\text{Гн} \cdot 10^{-8}\text{Ф}}} = 10^7 \frac{\text{рад}}{\text{с}}$ или 1591549 Гц $\approx 1.59 \cdot 10^6$Гц)

При такой частоте сила тока рассчитывается по формуле $I = \frac{U}{R}$, где R-активное сопротивление, т.е. должна равняться $\frac{10\ \text{В}}{10\ \text{Ом}} = 1\ \text{А}$. Выставив нужную частоту, действительно можно наблюдать резонанс напряжений:

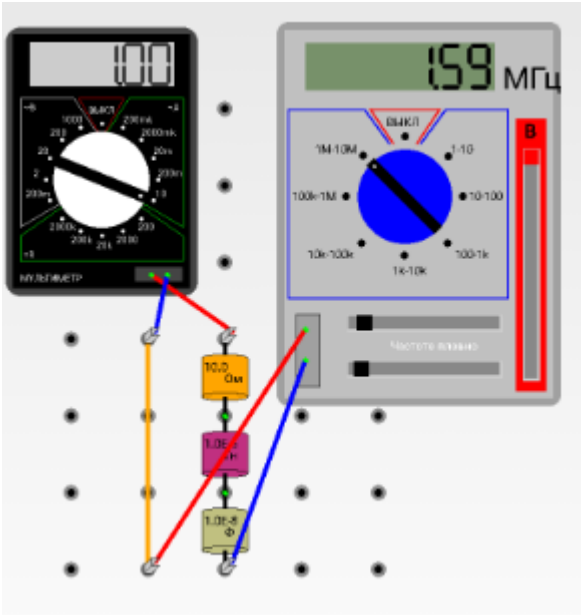

Рис. 12. Исследование явления резонанса напряжений

Также наблюдается резонанс токов в параллельном контуре, представленном на Рис. 13.

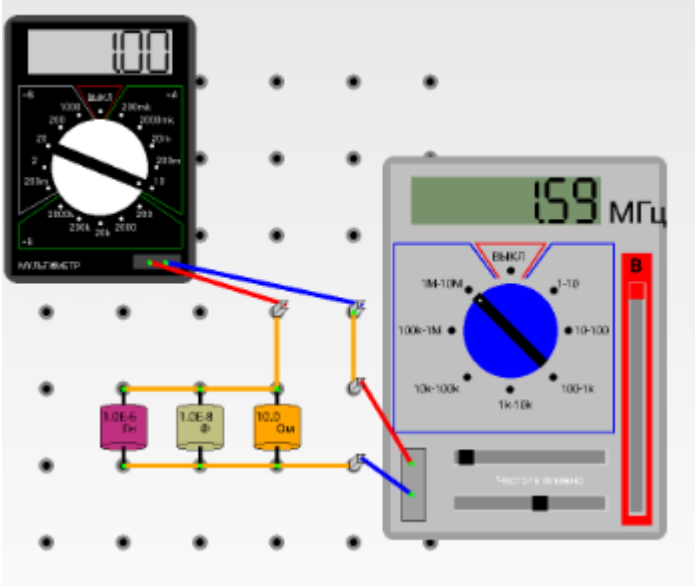

Рис. 13 Исследование явления резонанса токов

Выводы по главе 3

Полученные показания амперметра при расчете цепей переменного и постоянного тока полностью совпадают с теоретическими расчетами. Были исследованы явления последовательного и параллельного резонансов. Также была проверена правильность расчета напряжения на элементах цепи.

Выводы

В рамках данной работы были получены следующие результаты:

- Была доработана существующая иерархия классов, а именно:
  - разработаны классы для устройств, таких как генератор переменного тока, конденсатор и катушка индуктивности
  - разработаны классы для корректного отображения устройств
- Сделано обобщение для расчетов электрических цепей таким образом, что расчеты являются универсальными для электрических цепей переменного и постоянного тока с учетом особенностей свойств элементов
- Была сделана проверка получившихся результатов и получена согласованность результатов с теоретическими расчетами
- Был проведен частичный рефакторинг существующего кода

Литература

1. М.А. Максимов, В.В. Монахов, С.А. Мортынюк, Е.В. Монахова, Н.В. Кузьмин – Разработка программных средств мультиплатформенной поддержки интернет-олимпиады школьников по физике / III Международная научно-практическая конференция "Инновации в информационных технологиях и образовании", 2014. – С. 317-324
2. Фриш В.С. Разработка на языке Java конструктора моделей простых электрических цепей: Бакалаврская работа. СПб: СПбГУ, 2013.
3. Файзулин Е.Э. Разработка конструктора моделей электрических цепей для платформы Android: Магистерская работа. СПб: СПбГУ, 2017.
4. Canvas [Электронный ресурс] // Android Developers / Google. URL: https://developer.android.com/reference/android/graphics/Canvas (дата доступа: 24.05.2018)
5. Paint [Электронный ресурс] // Android Developers / Google. URL: https://developer.android.com/reference/android/graphics/Paint (дата доступа: 24.05.2018)
6. Расчет электрических цепей: учебно-практическое пособие / П.А. Галайдин, Ю.Н. Мустафаев; Балт. гос. техн. ун-т. — СПб., 2014. — 98 с.
7. Розвезев В. С. Расчет и измерение токов и напряжений в Android-модели электрического конструктора. Курсовая работа. СПб, СПбГУ, 2018. -24 с.
8. Теоретические основы электротехники. Электрические цепи : учебник для бакалавров / Л. А. Бессонов. — 12-е изд., исправ. и доп. — М.: Издательство Юрайт, 2014. — 701 с.

Приложение А. Класс PAlternator

```
public class PAlternator extends PDevice implements IIndicative {
    private static final int DEFAULT_WIDTH = 370 * ModelView.sc;
    private static final int DEFAULT_HEIGHT = 500 * ModelView.sc;
    private static final float labelSize = 15 * ModelView.sc;
    private int switcherModesCount = 8;
    private String KILO_HERZ = "кГц";
    private String HERZ = "Гц";
    private String MEGA_HERZ = "МГц";
    private String currentUnit = HERZ;
    private double currentAccuracy = 1;
    private PSwitcher switcher;
    private PScrollPane voltageScroll;
    private PScrollPane frequencyScrollRough;
    private PScrollPane frequencyScrollSmooth;
    private PDisplay display;
    private PWire pWire1;
    private PWire pWire2;
    private LinkedList<PClamp> pClampsList;
    private Paint paint;
    private double minFrequency;
    private double maxFrequency;
    private final double minVoltage = 0;
    private final double maxVoltage = 10;
    private double roughFrequency;
    private double smoothFrequency;

    @SuppressLint("UseSparseArrays")
    private Map<Integer, String> switcherLabels = new HashMap<>();
    @SuppressLint("UseSparseArrays")
    private Map<Integer, Dimension> switcherLabelsPositions = new HashMap<>();
    @SuppressLint("UseSparseArrays")
    private Map<Integer, Dimension> switcherMarks = new HashMap<>();
    @SuppressLint("UseSparseArrays")
    private Map<Integer, ValueInterval> switcherValues = new HashMap<>();

    public PAlternator() {
        paint = new Paint();
        device = new Alternator(0.001);
        device.setVoltage(10.0);
        size.width = DEFAULT_WIDTH;
        size.height = DEFAULT_HEIGHT;

        voltageScroll = new PScrollPane(new PVerticalScroll(), 150, 10);
        voltageScroll.getpScroll().setSize(voltageScroll.getSize().width,
voltageScroll.getSize().width);
        voltageScroll.setHolder(this);

        frequencyScrollRough = new PScrollPane(new PHorizontalScroll(), 10, 100);
        frequencyScrollRough.getpScroll().setSize(frequencyScrollRough.getSize().height,
frequencyScrollRough.getSize().height);
        frequencyScrollRough.setHolder(this);
        frequencyScrollRough.setValue(0.0);

        frequencyScrollSmooth = new PScrollPane(new PHorizontalScroll(), 10, 100);
        frequencyScrollSmooth.getpScroll().setSize(frequencyScrollSmooth.getSize().height,
frequencyScrollSmooth.getSize().height);
        frequencyScrollSmooth.setHolder(this);
        frequencyScrollSmooth.setValue(0.0);

        pWire1 = new PWire();
        pWire2 = new PWire();
        pWire1.setPosition(10, 10);
```

```
pWire1.setMovable(false);
pWire2.setMovable(false);

display = new PDisplay(Color.rgb(119, 145, 106));
display.setHolder(this);
display.switchOff();

switcher = new PSwitcher(switcherModesCount, Color.BLUE);
initializeSwitcherValues();
switcher.setHolder(this);
switcher.setState(0);

pClampsList = new LinkedList<>();

pWire1.getPClampsList().getFirst().setType(PClamp.TYPE_0);
pWire2.getPClampsList().getFirst().setType(PClamp.TYPE_0);
pWire1.getPClampsList().getFirst().setMovable(false);
pWire2.getPClampsList().getFirst().setMovable(false);
pWire1.getPClampsList().getLast().moveBy(-100, size.height);
pWire2.getPClampsList().getLast().moveBy(-40, size.height);
pWire1.getPClampsList().getLast().setType(PClamp.TYPE_2);
pWire2.getPClampsList().getLast().setType(PClamp.TYPE_2);
pWire1.setColor(Color.RED);
pWire2.setColor(Color.BLUE);

pWire1.getPClampsList().getLast().setHolder(this);
pClampsList.add(pWire1.getPClampsList().getLast());
pWire1.setHolder(this);
pWire2.getPClampsList().getLast().setHolder(this);
pClampsList.add(pWire2.getPClampsList().getLast());
pWire2.setHolder(this);
type = "alternator";
deviceType = 6;

addOnSwitcherTouchListener(new OnTouchListener() {
    @Override
    public boolean onTouch(View view, MotionEvent event) {
        if (switcher.getState() == 0) {
            display.switchOff();
            Scheme.setFrequency(0);
            ListsOfDevices.updateScheme();
        } else {
            ValueInterval interval = switcherValues.get(switcher.getState());
            setMinFrequency(interval.getMinValue());
            setMaxFrequency(interval.getMaxValue());
            setUnit();
            setAccuracy();
            Scheme.setFrequency(calculateFrequency());
            setDisplayFrequency();
            ListsOfDevices.updateScheme();
        }
        return true;
    }
});

addOnFrequencyScrollTouchListener(new OnTouchListener() {
    @Override
    public boolean onTouch(View view, MotionEvent event) {
        Scheme.setFrequency(calculateFrequency());
        setDisplayFrequency();
        ListsOfDevices.updateScheme();
        return true;
    }
});
```

```
        addOnVoltageTouchListener(new OnTouchListener() {
            @Override
            public boolean onTouch(View view, MotionEvent event) {
                double currentVoltage = calculateVoltage();
                device.setVoltage(currentVoltage);
                ListsOfDevices.updateScheme();
                return true;
            }
        });

        setZIndex(ListsOfDevices.getMaxZIndex());
        ListsOfDevices.sortByZIndex();
        ListsOfDevices.getIndicativeList().add(this);
    }

    private void setDisplayFrequency(){
        double currentFrequency = Scheme.getFrequency();
        double displayFrequency = currentUnit.equals(HERZ) ?
                currentFrequency : currentUnit.equals(KILO_HERZ) ?
                currentFrequency / 1000.0 : currentFrequency / 1_000_000.0;
        display.setValue(displayFrequency, currentAccuracy);
    }

    private void initializeSwitcherValues() {
        switcherValues = new HashMap<>();
        switcherValues.put(0, new ValueInterval(0, 0));
        switcherValues.put(1, new ValueInterval(1, 10));
        switcherValues.put(2, new ValueInterval(10, 100));
        switcherValues.put(3, new ValueInterval(100, 1000));
        switcherValues.put(4, new ValueInterval(1000, 10000));
        switcherValues.put(5, new ValueInterval(10000, 100000));
        switcherValues.put(6, new ValueInterval(100000, 1000000));
        switcherValues.put(7, new ValueInterval(1000000, 10000000));
    }

    private void initializeSwitcherLabels() {
        switcherLabelsPositions.put(0, new Dimension(switcher.getPosition().x +
switcher.getSize().width / 2 - 23, switcher.getPosition().y - 30));
        switcherLabelsPositions.put(1, new Dimension(switcher.getPosition().x +
(switcher.getSize().width * 3) / 4 + 30, switcher.getPosition().y +
switcher.getSize().height / 4 - 25));
        switcherLabelsPositions.put(2, new Dimension(switcher.getPosition().x +
switcher.getSize().width + 22, switcher.getPosition().y + switcher.getSize().height / 2 +
5));
        switcherLabelsPositions.put(3, new Dimension(switcher.getPosition().x +
(switcher.getSize().width * 3) / 4 + 35, switcher.getPosition().y +
(switcher.getSize().height * 3) / 4 + 50));
        switcherLabelsPositions.put(4, new Dimension(switcher.getPosition().x +
switcher.getSize().width / 2 - 15, switcher.getPosition().y + switcher.getSize().height +
40));
        switcherLabelsPositions.put(5, new Dimension(switcher.getPosition().x - 65,
switcher.getPosition().y + (switcher.getSize().height * 3) / 4 + 50));
        switcherLabelsPositions.put(6, new Dimension(switcher.getPosition().x - 85,
switcher.getPosition().y + switcher.getSize().height / 2 + 5));
        switcherLabelsPositions.put(7, new Dimension(switcher.getPosition().x - 55,
switcher.getPosition().y + switcher.getSize().height / 4 - 25));

        switcherLabels.put(0, "ВЫКЛ");
        switcherLabels.put(1, "1-10");
        switcherLabels.put(2, "10-100");
        switcherLabels.put(3, "100-1k");
        switcherLabels.put(4, "1k-10k");
        switcherLabels.put(5, "10k-100k");
        switcherLabels.put(6, "100k-1M");
```

```
        switcherLabels.put(7, "1M-10M");

        switcherMarks.put(0, new Dimension(switcherLabelsPositions.get(0).width + 24,
switcherLabelsPositions.get(0).height + 13));
        switcherMarks.put(1, new Dimension(switcherLabelsPositions.get(1).width - 4,
switcherLabelsPositions.get(1).height + 4));
        switcherMarks.put(2, new Dimension(switcherLabelsPositions.get(2).width - 8,
switcherLabelsPositions.get(2).height - 5));
        switcherMarks.put(3, new Dimension(switcherLabelsPositions.get(3).width - 10,
switcherLabelsPositions.get(3).height - 20));
        switcherMarks.put(4, new Dimension(switcherLabelsPositions.get(4).width + 15,
switcherLabelsPositions.get(4).height - 25));
        switcherMarks.put(5, new Dimension(switcherLabelsPositions.get(5).width + 70,
switcherLabelsPositions.get(5).height - 20));
        switcherMarks.put(6, new Dimension(switcherLabelsPositions.get(6).width + 70,
switcherLabelsPositions.get(6).height - 5));
        switcherMarks.put(7, new Dimension(switcherLabelsPositions.get(7).width + 62,
switcherLabelsPositions.get(7).height + 3));
    }

    private void addOnSwitcherTouchListener(OnTouchListener listener) {
        switcher.addOnTouchListener(listener);
    }

    private void addOnFrequencyScrollTouchListener(OnTouchListener listener) {
        frequencyScrollSmooth.addOnTouchListener(listener);
        frequencyScrollRough.addOnTouchListener(listener);
    }

    private void addOnVoltageTouchListener(OnTouchListener listener) {
        voltageScroll.addOnTouchListener(listener);
    }


    @Override
    public LinkedList<PClamp> getPClampsList() {
        return pClampsList;
    }

    @Override
    public void updateClamps() {
        if (pClampsList.getFirst().getConnect() != null) {
            device.setConnection1(pClampsList.getFirst().getConnect().getConnection());
        } else {
            device.setConnection1(null)
            ;
        }
        if (pClampsList.getLast().getConnect() != null) {
            device.setConnection2(pClampsList.getLast().getConnect().getConnection());
        } else {
            device.setConnection2(null);
        }

    }

    @Override
    public void updateValue() {

    }

    private double calculateVoltage() {
        return (1 - voltageScroll.getValue()) * (getMaxVoltage() - getMinVoltage());
    }

    private void setUnit() {
```

```
        int switcherState = switcher.getState();
        if (switcherState <= 3) {
            currentUnit = HERZ;
        } else if (switcherState <= 6) {
            currentUnit = KILO_HERZ;
        } else {
            currentUnit = MEGA_HERZ;
        }
    }

    private void setAccuracy() {
        int switcherState = switcher.getState();
        if (switcherState == 1 || switcherState == 4 || switcherState == 7) {
            currentAccuracy = 0.01;
        } else if (switcherState == 2 || switcherState == 5) {
            currentAccuracy = 0.1;
        } else {
            currentAccuracy = 1;
        }
    }

    private double calculateFrequency() {
        roughFrequency = frequencyScrollRough.getValue();
        smoothFrequency = frequencyScrollSmooth.getValue();
        return getMinFrequency()
                + frequencyScrollRough.getValue()
                * (getMaxFrequency() - getMinFrequency())
                * 0.95
                + frequencyScrollSmooth.getValue()
                * (getMaxFrequency() - getMinFrequency())
                * 0.05;
    }

    @Override
    public void paint(Canvas canvas) {
        drawOuterContainer(canvas);
        drawInnerContainer(canvas);
        drawVoltageScrollContainer(canvas);
        paint.setColor(Color.WHITE);
        paint.setStyle(Paint.Style.STROKE);
        paint.setTextSize(labelSize);
        canvas.drawText("Частота плавно",
                position.x + (int) (size.width / 2.5),
                150 + position.y + 10 * size.height / 18,
                paint
        );
        paint.setColor(Color.BLACK);
        initializeSwitcherLabels();
        paint.setStyle(Paint.Style.FILL);
        for (int index = 0; index < switcherModesCount; index++) {
            canvas.drawText(switcherLabels.get(index),
                    switcherLabelsPositions.get(index).width,
                    switcherLabelsPositions.get(index).height,
                    paint
            );
            canvas.drawCircle(switcherMarks.get(index).width,
                    switcherMarks.get(index).height,
                    5,
                    paint
            );
        }

        paint.setTextSize(40);
        canvas.drawText(currentUnit,
                display.getPosition().x + display.getSize().width + 10,
```

```
                display.getPosition().y + 60, paint
        );
        drawSwitcherOutlines(canvas);
        pWire1.getPClampsList().getFirst().setCenter(position.x + size.width / 10,
                position.y + size.height - 100
        );
        pWire2.getPClampsList().getFirst().setCenter(position.x + size.width / 10,
                position.y + size.height - 60
        );
        drawClampsContainer(canvas);
    }

    private void drawClampsContainer(Canvas canvas) {
        paint.setColor(Color.rgb(160, 160, 160));
        paint.setStyle(Paint.Style.FILL);
        RectF clampsOutline = new RectF(
                getPosition().x + 25,
                getPosition().y + size.height - 120,
                getPosition().x + 55,
                getPosition().y + size.height - 25
        );

        canvas.drawRoundRect(clampsOutline, 0, 0, paint);
        paint.setStyle(Paint.Style.STROKE);
        paint.setColor(Color.BLACK);
        canvas.drawRoundRect(clampsOutline, 0, 0, paint);
    }

    private void drawSwitcherOutlines(Canvas canvas) {
        Path path = new Path();
        paint.setColor(Color.RED);
        paint.setStyle(Paint.Style.STROKE);
        paint.setStrokeWidth(2.5f);
        //drawing relatively to "ВЫКЛ" switcher mark
        path.moveTo(switcherMarks.get(0).width - 15, switcherMarks.get(0).height + 10);
        path.lineTo(switcherMarks.get(0).width - 45, switcherMarks.get(0).height - 35);
        path.lineTo(switcherMarks.get(0).width + 45, switcherMarks.get(0).height - 35);
        path.lineTo(switcherMarks.get(0).width + 15, switcherMarks.get(0).height + 10);
        canvas.drawPath(path, paint);

        path.reset();

        paint.setColor(Color.BLUE);
        path.moveTo(switcherMarks.get(0).width - 20, switcherMarks.get(0).height + 10);
        path.lineTo(switcherMarks.get(0).width - 53, switcherMarks.get(0).height - 35);
        path.lineTo(position.x + 15, switcherMarks.get(0).height - 35);
        path.lineTo(position.x + 15, switcherMarks.get(0).height - 35 +
switcher.getSize().height * 2);
        path.lineTo(voltageScroll.getPosition().x - 20, switcherMarks.get(0).height - 35 +
switcher.getSize().height * 2);
        path.lineTo(voltageScroll.getPosition().x - 20, switcherMarks.get(0).height - 35);
        path.lineTo(switcherMarks.get(0).width + 53, switcherMarks.get(0).height - 35);
        path.lineTo(switcherMarks.get(0).width + 20, switcherMarks.get(0).height + 10);
        canvas.drawPath(path, paint);

        paint.setStrokeWidth(0.0f);
    }

    private void drawVoltageScrollContainer(Canvas canvas) {
        paint.setColor(Color.RED);
        paint.setStyle(Paint.Style.FILL);
        int dx = 10;
        int dyTop = 40;
        int dyBottom = 15;
        RectF voltageOutline = new RectF(
```

```
                voltageScroll.getPosition().x - dx,
                voltageScroll.getPosition().y - dyTop,
                voltageScroll.getPosition().x + voltageScroll.getSize().width + dx,
                voltageScroll.getPosition().y + voltageScroll.getSize().height + dyBottom
        );
        canvas.drawRoundRect(voltageOutline, 0, 0, paint);
        paint.setColor(Color.BLACK);
        paint.setTextSize(25);
        paint.setTypeface(Typeface.DEFAULT_BOLD);
        canvas.drawText("B", voltageScroll.getPosition().x + 1,
voltageScroll.getPosition().y - 15, paint);
        paint.setTypeface(Typeface.DEFAULT);

    }

    private void drawInnerContainer(Canvas canvas) {
        paint.setColor(Color.rgb(192, 192, 192));
        paint.setStyle(Paint.Style.FILL);
        int dx = 10, dy = 10; // distance between inner and outer container
        RectF innerContainer = new RectF(
                position.x + dx,
                position.y + dy,
                position.x + size.width - dx,
                position.y + size.height - dy
        );
        canvas.drawRoundRect(innerContainer, 0, 0, paint);
    }

    private void drawOuterContainer(Canvas canvas) {
        paint.setColor(Color.rgb(160, 160, 160));
        paint.setStyle(Paint.Style.FILL);
        RectF outerContainer = new RectF(position.x,
                position.y,
                position.x + size.width,
                position.y + size.height
        );
        canvas.drawRoundRect(outerContainer, 15, 15, paint);
    }

    @Override
    public String getInfo(int n) {
        return null;
    }

    @Override
    public void updateCoordinates() {
        size.width = (int) (DEFAULT_WIDTH * getScaleFactor());
        size.height = (int) (DEFAULT_HEIGHT * getScaleFactor());

        switcher.setPosition(position.x + (int) (size.width / 3.5), position.y + 6 *
size.height / 18);
        switcher.setSize(size.width / 3, size.width / 3);
        voltageScroll.setPosition(position.x + (int) (size.width / 1.15), position.y +
(size.height / 3));
        frequencyScrollRough.setPosition(position.x + size.width / 4, 105 + position.y + 10
* size.height / 18);
        frequencyScrollSmooth.setPosition(position.x + size.width / 4, 165 + position.y + 10
* size.height / 18);
        frequencyScrollRough.setValue(roughFrequency);
        frequencyScrollSmooth.setValue(smoothFrequency);
        display.setPosition(position.x + size.width / 10, position.y + size.height / 18);
        display.setSize(size.width - size.width / 3, size.height / 8);

        updateValue();
    }
```

```
    @Override
    public void changePositions(float dx, float dy) {

    }

    public double getMinFrequency() {
        return minFrequency;
    }

    public void setMinFrequency(double minFrequency) {
        this.minFrequency = minFrequency;
    }

    public double getMaxFrequency() {
        return maxFrequency;
    }

    public void setMaxFrequency(double maxFrequency) {
        this.maxFrequency = maxFrequency;
    }


    public double getMinVoltage() {
        return minVoltage;
    }


    public double getMaxVoltage() {
        return maxVoltage;
    }

}
```

Приложение Б. Класс PCapacitor

```
package ru.spbu.abarsic.abarsicgraph.graphica.devices;

import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.RectF;

import java.util.LinkedList;

import ru.spbu.abarsic.abarsicgraph.ModelView;
import ru.spbu.abarsic.abarsicgraph.graphica.PClamp;
import ru.spbu.abarsic.abarsicgraph.logica.sequential.Capacitor;

public class PCapacitor extends PDevice {
    private LinkedList<PClamp> pClampsList = new LinkedList<>();
    private PClamp pClamp1 = new PClamp(PClamp.TYPE_1);
    private PClamp pClamp2 = new PClamp(PClamp.TYPE_1);
    private boolean known;
    public static final int DEFAULT_WIDTH = 60 * ModelView.sc;
    public static final int DEFAULT_HEIGHT = 100 * ModelView.sc;
    private static final float textSize = 16 * ModelView.sc;
    private int midX;
    private int y1;
    private int y2;
    private int y3;
    private int x4;
    private int y4;
    private int x5;
    private int y5;
    private String caption;
    private int height;
    private int r;
    private int length;
    private Paint p = new Paint();


    public PCapacitor() {
        pClamp1.setHolder(this);
        pClamp2.setHolder(this);
        pClamp1.setMovable(false);
        pClamp2.setMovable(false);
        pClampsList.add(pClamp1);
        pClampsList.add(pClamp2);
        size.width = DEFAULT_WIDTH;
        size.height = DEFAULT_HEIGHT;
        setColor(Color.BLACK);
        known = true;
        type = "capacitor";
        deviceType = 7;
        updateCoordinates();
    }

    public PCapacitor(double capacity) {
        device = new Capacitor();
        pClamp1.setHolder(this);
        pClamp2.setHolder(this);
        pClamp1.setMovable(false);
        pClamp2.setMovable(false);
        pClampsList.add(pClamp1);
        pClampsList.add(pClamp2);
        size.width = DEFAULT_WIDTH;
        size.height = DEFAULT_HEIGHT;
```

```
        setColor(Color.BLACK);
        device.setCapacity(capacity);
        known = true;
        type = "capacitor";
        deviceType = 7;
        updateCoordinates();
    }

    public PCapacitor(double capacity, boolean known) {
        pClamp1.setHolder(this);
        pClamp2.setHolder(this);
        pClamp1.setMovable(false);
        pClamp2.setMovable(false);
        pClampsList.add(pClamp1);
        pClampsList.add(pClamp2);
        size.width = DEFAULT_WIDTH;
        size.height = DEFAULT_HEIGHT;
        setColor(Color.BLACK);
        this.known = known;
        device.setCapacity(capacity);
        type = "capacitor";
        deviceType = 7;
        updateCoordinates();
    }

    public String getInfo(int n) {
        String temp = "par_objects[" + n + ",";
        String result = temp + "0]="
                + deviceType + ";\n"
                + temp + "1]="
                + position.x
                + ";\n" + temp
                + "2]=" + position.y + ";\n"
                + temp + "3]=" + device.getCapacity()
                + ";\n";
        return result;
    }

    @Override
    public void updateCoordinates() {
        size.width = (int) (DEFAULT_WIDTH * getScaleFactor());
        size.height = (int) (DEFAULT_HEIGHT * getScaleFactor());
        midX = position.x + size.width / 2;
        r = size.height / 10;
        height = size.height - 4 * r;
        y1 = position.y + 2 * r;
        y2 = position.y + size.height - 3 * r;
        y3 = y2 + r / 2;
        length = 5 * r / 2;
        x4 = midX;
        y4 = position.y + 4 * size.height / 7;
        x5 = position.x + size.width / 20;
        y5 = position.y + 3 * size.height / 7;
        if (known) {
            caption = Double.toString(device.getCapacity());
        } else {
            caption = "???";
        }
        if (caption.length() > 6) {
            caption = caption.substring(0, 6);
        }
        pClamp1.setCenter(midX, position.y);
        pClamp2.setCenter(midX, position.y + size.height);
        info = type + "," + position.x + "," + position.y + "," +
                device.getVoltage() + "," + device.getCapacity() +
```

```
            "," + device.getConnection1() + "," + device.getConnection2() + ";";
}

@Override
public void changePositions(float dx, float dy) {
    midX += dx;
    y1 += dy;
    y2 += dy;
    y3 += dy;
    x4 += dx;
    y4 += dy;
    x5 += dx;
    y5 += dy;
}

@Override
public void paint(Canvas g) {
    p.setTextSize(textSize);
    p.setColor(Color.BLACK);
    p.setStyle(Paint.Style.FILL);
    g.drawRect(midX - 2, position.y, midX + 3, position.y + length, p);

    p.setColor(Color.rgb(192, 192, 128));
    p.setStyle(Paint.Style.FILL);
    RectF roundRect = new RectF(position.x, y1, position.x + size.width, y1 + height);
    g.drawRoundRect(roundRect, size.width, r, p);

    p.setColor(Color.BLACK);
    p.setStyle(Paint.Style.STROKE);
    g.drawRoundRect(roundRect, size.width, r, p);

    p.setColor(Color.BLACK);
    p.setStyle(Paint.Style.STROKE);
    RectF oval = new RectF(position.x, y2, position.x + size.width, y2 + r);
    g.drawOval(oval, p);

    p.setColor(Color.BLACK);
    p.setStyle(Paint.Style.FILL);
    g.drawRect(midX - 2, y3, midX + 3, y3 + length, p);

    p.setStyle(Paint.Style.STROKE);
    g.drawText(caption, x5, y5, p);
    g.drawText("Φ", x4, y4, p);
}

@Override
public boolean isChosen(float x, float y) {
    if (super.isChosen(x, y)) {
        pClamp2.setFixed(false);
        pClamp1.setFixed(false);
        return true;
    }
    return false;
}

@Override
public LinkedList<PClamp> getPClampsList() {
    return pClampsList;
}

@Override
public void updateClamps() {
    if (pClamp1.getConnect() != null) {
        device.setConnection1(pClamp1.getConnect().getConnection());
    } else {
```

```
            device.setConnection1(null);
        }
        if (pClamp2.getConnect() != null) {
            device.setConnection2(pClamp2.getConnect().getConnection());
        } else {
            device.setConnection2(null);
        }
    }

    public boolean isKnown() {
        return known;
    }

    public void setKnown(boolean known) {
        this.known = known;
    }

    @Override
    public void setScaleFactor(double scaleFactor) {
        if (!pClamp1.isFixed() && !pClamp1.isFixed()) {
            super.setScaleFactor(scaleFactor);
        }
    }
}
```

Приложение В. Метод для заполнения необходимых для решения системы уравнений матриц с учетом комплексности элементов

```
public void createMatrix1() {
    int q = connects.size(); // количество узлов в цепи
    int p = devices.size(); // количество рёбер в цепи
    A = new Complex[q - 1][p];
    Y = new Complex[p][p];
    J = new Complex[p];
    E = new Complex[p];
    U = new Complex[q - 1];
    SequentialDevice currentDevice;
    for (int i = 0; i < p; i++) {
        currentDevice = devices.get(i);
        double angularFrequency = 2 * Math.PI * Scheme.getFrequency();
        if (Double.compare(currentDevice.getInductance(), 0.0) != 0) {
            // inductive resistance = i*w*L, where w-angular frequency, L-inductivity
            Complex inductiveResistance = new Complex(0.0,
                    angularFrequency * currentDevice.getInductance());
            Y[i][i] = inductiveResistance.pow(-1);
        } else if (Double.compare(currentDevice.getCapacity(), 0.0) != 0) {
            // capacitive resistance = 1/(i*w*C), where w-angular frequency, C-capacity
            if (Double.compare(angularFrequency, 0.0) != 0) {
                Y[i][i] = new Complex(0.0, angularFrequency * currentDevice.getCapacity());
            } else {
                Y[i][i] = new Complex(0.0, 0.0);
            }
        } else {
            Y[i][i] = new Complex(currentDevice.getResistance(), 0.0).pow(-1);
        }
        for (int k = 0; k < p; k++) {
            if (k != i) {
                Y[i][k] = new Complex(0.0, 0.0);
            }
        }
        J[i] = new Complex(currentDevice.getAmperage(), 0.0);
        E[i] = new Complex(currentDevice.getVoltage(), 0.0);
        for (int j = 0; j < (q - 1); j++) {
            if (currentDevice.getConnection1() == connects.get(j)) {
                A[j][i] = new Complex(1.0, 0.0);
            } else if (currentDevice.getConnection2() == connects.get(j)) {
                A[j][i] = new Complex(-1.0, 0.0);
            } else {
                A[j][i] = new Complex(0.0, 0.0);
            }
        }
    }
}
```