

1. О приложении

Разработанная программа представляет из себя самостоятельное кроссплатформенное приложение, спроектированное на языке JavaSE 7u75;

Отличительными особенностями приложения являются:

- Математический пакет собственной разработки, а так же модуль наглядного отображения и ввода математических формул. Модуль полностью построен на векторной графике, что позволяет редактору адаптироваться под любые разрешения экрана, описанная идея ранее нигде не использовалась и полностью принадлежит автору.
- В данном продукте реализована идея живого приложения, по возможности автор освобождает пользователя от нажатия на лишние кнопки. Как только пользователь вносит малейшее изменение в параметры, графики, изменяет формулы, незамедлительно происходят все необходимые перерасчеты и перепостроения. Например, в основном разделе моделирования не используется не единой кнопки, а благодаря многопоточной оптимизации внутренних расчетов производительность интерфейса практически не снижается.
- Векторный движок собственной разработки, благодаря нему программный продукт выглядит одинаково хорошо как на большом мониторе, так и на экране мобильного телефона. Кроме того реализована возможность программного сглаживания готового изображения, что позволяет рисовать дробную часть пикселя и в конечном итоге позволяет получить наилучшее отображение графиков и текста.

Все изображения, используемые в приложении защищены авторским правом, алгоритмы и методы, используемые в приложении являются интеллектуальной собственностью автора, респатент 00000 от 00.00.0000.

Приложение предназначено для моделирования управляемого движения трехколесного робота, анализа качества управления, скорости выхода робота на желаемую траекторию движения, результаты математического моделирования которого используются в параграфе 3.4 главы 3 настоящей диссертации.

2. Разделы программного продукта

Моделирование начинается с запуска приложения trowth.exe для Windows и trowth.jar - для Linux, в основном меню программы пользователю предоставляется возможность ознакомиться с теоретической частью, лежащей в основе математического моделирования управляемого движения колесного робота, ознакомиться с информацией об авторах проекта или перейти к демонстрации работы заложенного закона управления.

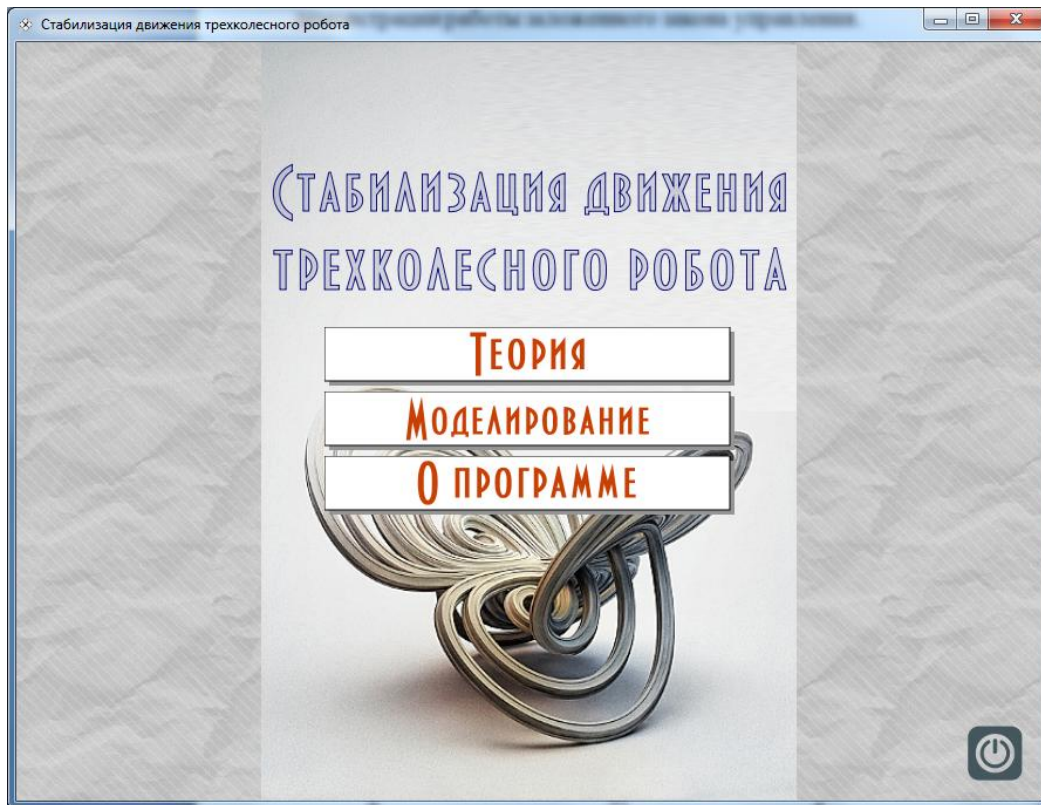


Рис. Внешний вид главного меню.

2.1. Теоретический раздел

В теоретическую часть входит схематическое изображение управляемого объекта, система дифференциальных уравнений, описывающая движение робота и описание характеристик, используемых в построении управления. Подробные теоретические выкладки содержатся в главе 3 настоящей диссертации.

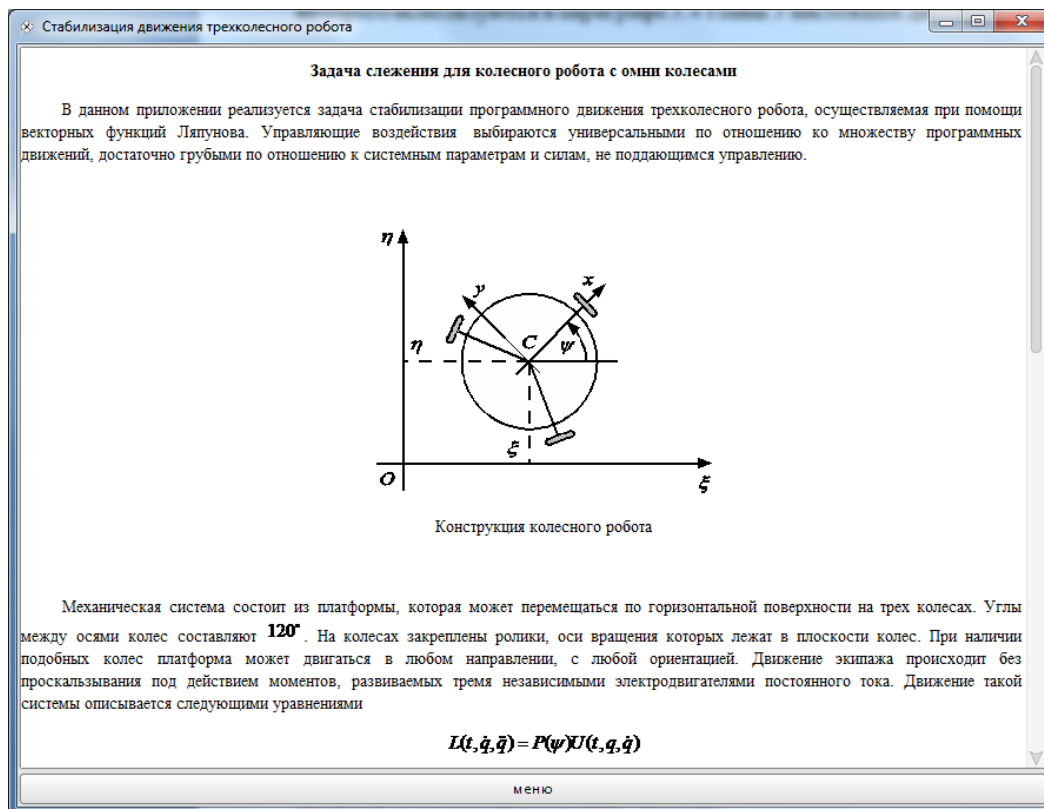


Рис. Внешний вид теоретического раздела.

2.2.Раздел моделирование

При переходе непосредственно к математическому моделированию пользователю предлагается задать параметры системы, параметры управления, а так же начальные точки положения робота, либо согласиться с предложенными. О значении каждого из параметров пользователь может прочесть в разделе приложения «Теория».

Параметры системы:	
m =	1.0
m _d =	3.0
l =	0.1
a =	0.2
h =	1.6
Параметры управления:	
k ₁ =	1.0
k ₂ =	0.3
k ₃ =	0.5
k ₄ =	0.8
Начальные точки:	
ξ ₍₀₎ ' =	0.5
ψ ₍₀₎ ' =	0.1
η ₍₀₎ ' =	1.0
ξ ₍₀₎ '' =	0.1
ψ ₍₀₎ '' =	0.1
η ₍₀₎ '' =	0.1

Рис. Параметры для расчетов

Следующим этапом работы с программой является задание движения робота одним из следующих способов:

1. Аналитический ввод функций, описывающих движение робота;
2. Задание траектории движения графически;
3. Загрузить файлы с массивами данных.

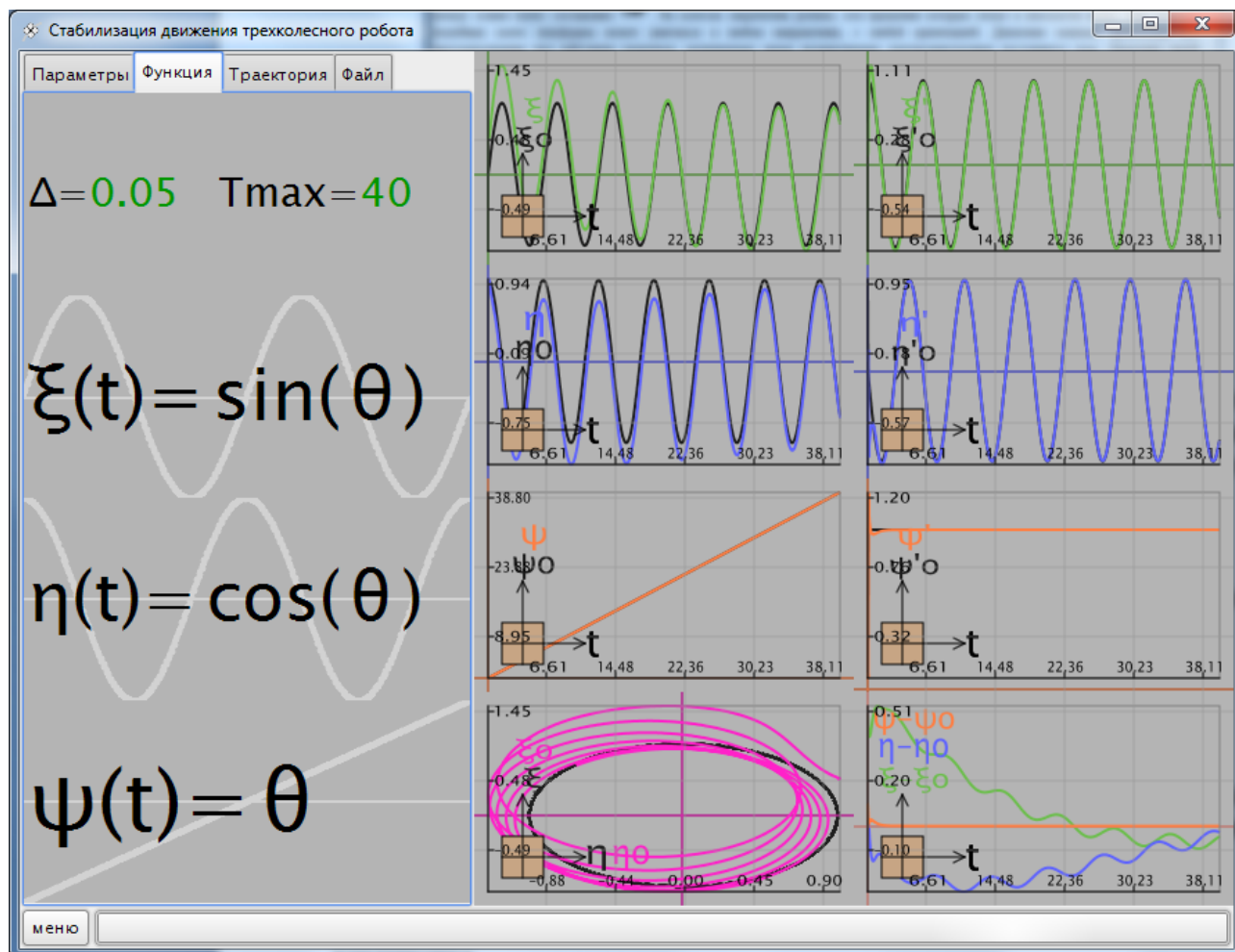


Рис. Внешний вид раздела моделирование.

В случае возникновения ошибки на одном из этапов ввода данных, пользователь незамедлительно увидит предупреждение о некорректном вводе и возможных методах исправления ошибки.

2.2.1. Задание закона движения робота аналитически

Ввод формул данным методом задействует достаточно мощный математический пакет самостоятельно разработанный автором.

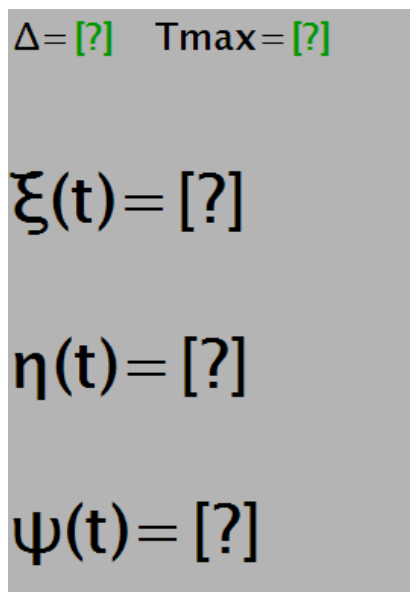
Использование векторной графики позволило сделать ввод формулы интерактивным, формула во время ввода автоматически занимает все доступное для нее место, в правой части стоит приглашение к вводу в виде знака "[?]". При клике мыши на этот значок выпадает список с вариантами возможных операций. Процесс создания формулы ведется путем собирания формулы из

отдельных функциональных блоков. После ввода формулы происходит её компиляция в оперативную память и при каждом следующем обращении процессор практически не участвует в вычислении, за счет этого достигается высокая скорость производительности пакета.

Стоит отметить, что тестирование скорости обработки формул математическим пакетом, выявило его преимущество перед такими известными инструментами, как MathCad 14 и Maple 12. Несомненно, преимущество в скорости обусловлено относительно скромным набором функций, входящих в арсенал разработанного продукта, однако, библиотека формул является более чем достаточной для проведения моделирования и может быть расширена при необходимости.

Для более подробного ознакомления с реализацией математического пакета см. Листинги 2-2.4.

На первом этапе пользователю предлагается ввести функции $\xi_0(t)$, $\eta_0(t)$, $\psi_0(t)$, описывающие движение робота, а так же шаг дискретизации Δ и время моделирования T_{\max} . По мере введения каждой функции под ней рисуется миниатюра графика, что позволяет ориентироваться и подбирать коэффициенты в сложных формулах. По окончанию ввода последней формулы незамедлительно происходит моделирование управляемого движения робота.



The image shows a software interface with a light gray background. At the top, there are two input fields: $\Delta = [?]$ and $T_{\max} = [?]$, where the question marks are green. Below these, there are three input fields for functions: $\xi(t) = [?]$, $\eta(t) = [?]$, and $\psi(t) = [?]$, where the question marks are black.

Рис. Система ожидает ввода управления с помощью формул.

При наборе формулы поддерживаются следующие действия:

- все математические операции (умножение, деление, сложение, вычитание, возведение в степень);
- ввод без ограничения по длине формулы;
- ввод без ограничений по количеству вложенных скобок;
- добавление делегированных тригонометрических и алгебраических функций ($\sin(t)$, $\cos(t)$, $tg(t)$, $ctg(t)$, $\exp(t)$, $\ln(t)$);
- ввод делегированных констант π ;
- ввод переменной t ;

Приоритеты операций в редакторе являются общепринятыми в математике, так, например в выражении $x+y*z$ сначала будет выполнено умножение, а потом сложение.

Кроме того, редактор дает возможность менять приоритеты по умолчанию, указывая их в явном виде с помощью символов парных скобок. При этом глубина вложенности прямо пропорциональна величине приоритета, то есть более внутренние скобки указывают на больший приоритет, чем внешние, обрамляющие их. В предыдущем примере с суммой и произведением порядок вычисления можно поменять, используя скобки, записав всё выражение так:

$$(x+y)*z$$

Очередность ввода компонент уравнения осуществляется путем движения извне вовнутрь.

Пример $\sin(t)+2*t$, порядок действий ввода:

- 1) Добавить действие сложение;
- 2) Левый аргумент блока сложения заменить на функцию $\sin([?])$;
- 3) Добавить переменную времени аргументом функции;
- 4) Правый аргумент блока сложения заменить на блок умножения;
- 5) Левый аргумент блока умножения заменить на число(выделен зеленым);
- 6) Ввести число;
- 7) Правый аргумент блока умножения заменить на переменную времени;

Если все действия проделаны правильно, на заднем плане отрисовывается график функции.

$$\begin{aligned} \psi(t) &= [?] \\ \psi(t) &= [?] + [?] & (1) \\ \psi(t) &= \sin([?]) + [?] & (2) \\ \psi(t) &= \sin(\theta) + [?] & (3) \\ \psi(t) &= \sin(\theta) + [?] * [?] & (4) \\ \psi(t) &= \sin(\theta) + [?] * [?] & (5) \\ \psi(t) &= \sin(\theta) + 2 * [?] & (6) \\ \psi(t) &= \sin(\theta) + 2 * \theta & (7) \end{aligned}$$

Рис. Типичный ввод формулы по этапам функции $\sin(t)+2*t$.

2.2.2. Метод ввода траектории движения робота графически

Если пользователь выбирает графический способ ввода траектории, перед ним открывается поле для поточечного ввода произвольной траектории. Изначально поле представляется чистым листом на который можно нанести неограниченное количество точек. Каждая из точек имеет три характеристики (координаты на плоскости и угол поворота).

Ввод данных организован при помощи двух совокупных плоскостей $\xi O\psi$, $\eta O\psi$, переключение между которыми производится нажатием на перекрестие в левом нижнем углу. Скорость движения робота в этом случае принимается постоянной на каждом отрезке ломанной.

При редактировании траектории мгновенно происходит расчет управления и управляемого движения и отображается результат математического моделирования.

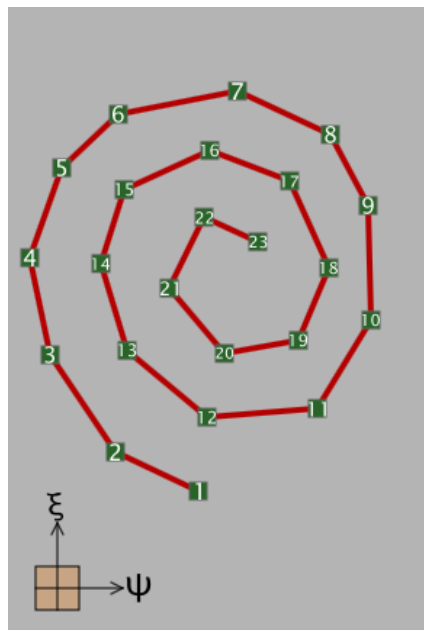


Рис. Внешний вид модуля ввода траектории.

Редактор является авторской разработкой и предоставляет возможность поточечного ввода траектории. Модуль позволяет вводить любые значения от $-1.7e+308$ до $1.7e+308$ с точность до $1e-12$.

Для более подробного ознакомления с модулем ввода траектории см. Листинг 3.

Основные инструменты редактора:

- удалить траекторию;
- добавление точки по клику в любой части рабочей области;
- вставить точку в произвольную часть ломаной;
- переместить точку без разрыва траектории;
- удалить узловую точку без разрыва траектории;
- замкнуть траекторию путем соединения

Благодаря векторной графике, лежащей в основе проектирования приложения, в данном модуле поддерживается масштабирование ломаной в широких пределах, что позволяет пользователю редактировать степень излома траектории. В случае, если траектория скрывается из рабочей области, на границах плоскости появляются ключевые точки, предоставляющие информацию о положении ломаной за пределами рабочей области.

2.2.3. Метод ввода файлом массивом данных

При выборе этого метода ввода, пользователю необходимо загрузить в программу файл с уже сформированным управлением. Данный программный продукт позволяет провести импорт из Microsoft Excell. На данный момент программный продукт поддерживает два расширения файла:

- cvs;
- msr - внутреннее расширение.

Выберите файл измерений...			
t	ξ	ψ	η

Рис. Внешний вид формы для ввода массива данных.

Данные полученные из массива итерационно подставляются в формулы.

Формат расширения msg:

Для импорта данных из других продуктов можно воспользоваться внутренним форматом данных с открытым протоколом. Данные следуют непрерывным потоком в следующем порядке:

- момент времени t_i ;
- значение $\xi(t_i)$;
- значение $\psi(t_i)$;
- значение $\eta(t_i)$;
- символ перехода на новую строку + символ сдвига каретки;

В качестве разделителя данных используется символ «;». Дробные числа записываются через символ «,».

Пример:

0,5;1;2;3;

1;4;5;6;

2.3. Расчетная часть

Для детального ознакомления с реализацией расчета см. Листинг 1.

После задания траектории движения робота одним из описанных способов, в правой части приложения выводятся следующие графики

1. графики $\xi(t), \xi_0(t)$ траектории центра масс платформы
2. графики $\eta(t), \eta_0(t)$ координаты центра масс платформы
3. графики $\psi(t), \psi_0(t)$ угла поворота платформы
4. графики $\dot{\xi}(t), \dot{\xi}_0(t)$ скорости центра масс платформы
5. графики $\dot{\eta}(t), \dot{\eta}_0(t)$ скорости координаты центра масс платформы
6. графики $\dot{\psi}(t), \dot{\psi}_0(t)$ угла поворота платформы
7. графики траектории платформы на фазовой плоскости
 $\xi(\eta(t)), \xi_0(\eta(t))$ угла поворота платформы
8. графики разностей $\dot{\xi}(t) - \dot{\xi}_0(t), \dot{\eta}(t) - \dot{\eta}_0(t), \dot{\psi}(t) - \dot{\psi}_0(t)$

При построении графиков использовался пятиточечный метод численного дифференцирования.

По графикам можно судить, что рассматриваемая система двигается ли вдоль отслеживаемой траектории на расстоянии, не превышающем погрешности слежения.

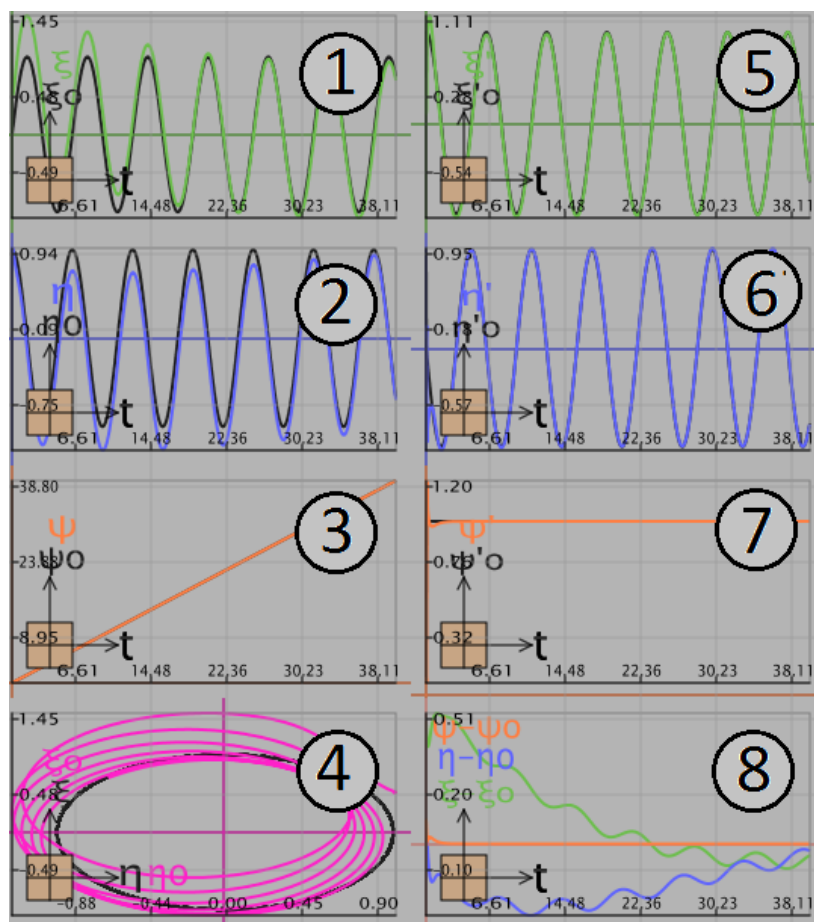


Рис. Внешний вид расчетной части

Листинг 1. Исходный код класса GroupPlotter.java Класс реализует ядро расчетов и вывода графиков

```
package ru.jcup.saa.science.trowh.gui.plotter;

import java.awt.Color;
import java.awt.GridLayout;
import java.util.ArrayList;

import javax.swing.JProgressBar;

import ru.jcup.saa.science.trowh.SelectPanel;
import ru.jcup.saa.science.trowh.gui.input.file.Measurement;
import ru.jcup.saa.science.trowh.gui.input.formula.Point2D;
import ru.jcup.saa.science.trowh.utils.arithmetic.value.LinkedException;

/**
 * Основной класс расчетов
 * @author Katq
 */

public class GroupPlotter extends SelectPanel{

    private static final long serialVersionUID = 1L;

    private Plotter plotter1;
    private Plotter plotter2;
    private Plotter plotter3;

    private Plotter plotter4;
    private Plotter plotter5;
    private Plotter plotter6;

    private Plotter plotter7;
    private Plotter plotter8;

    public GroupPlotter() {
        plotter1 = new Plotter();
        plotter2 = new Plotter();
        plotter3 = new Plotter();
        plotter4 = new Plotter();
        plotter5 = new Plotter();
        plotter6 = new Plotter();
        plotter7 = new Plotter();
        plotter8 = new Plotter();

        this.setLayout(new GridLayout(4,2));
        this.add(plotter1);
        this.add(plotter4);
        this.add(plotter2);
        this.add(plotter5);
        this.add(plotter3);
```

```

        this.add(plotter6);

        this.add(plotter7);
        this.add(plotter8);
    }

    private ArrayList<Measurement> points;
    private PlotterThread plotterThread = new PlotterThread();

    public void setGraph(ArrayList<Measurement> points) {
        this.points = points;
        if (plotterThread.isAlive()) {
            plotterThread.interrupt();
        }
        plotterThread = new PlotterThread();
        plotterThread.start();
    }

    private JProgressBar bar;

    public void setProgressBar(JProgressBar bar) {
        this.bar = bar;
    }

    public static LinkedDouble a= new LinkedDouble(0.2d);
    public static LinkedDouble k1= new LinkedDouble(1d);
    public static LinkedDouble k2= new LinkedDouble(0.3d);
    public static LinkedDouble k3= new LinkedDouble(0.5d);
    public static LinkedDouble k4= new LinkedDouble(0.8d);
    public static LinkedDouble m= new LinkedDouble(1d);
    public static LinkedDouble h= new LinkedDouble(1.6d);
    public static LinkedDouble md= new LinkedDouble(3d);
    public static LinkedDouble ind= new LinkedDouble(0.1d);

    //initial conditions
    public static LinkedDouble ksiP0 = new LinkedDouble(0.5d);
    public static LinkedDouble ettaP0 = new LinkedDouble(1d);
    public static LinkedDouble psiP0 = new LinkedDouble(0.1d);
    public static LinkedDouble ksiP00 = new LinkedDouble(0.1d);
    public static LinkedDouble ettaP00 = new LinkedDouble(0.1d);
    public static LinkedDouble psiP00 = new LinkedDouble(0.1d);

    class PlotterThread extends Thread{

        public void run() {

            plotter1.removeAllGraph();
            plotter2.removeAllGraph();
            plotter3.removeAllGraph();
            plotter4.removeAllGraph();
            plotter5.removeAllGraph();
            plotter6.removeAllGraph();

```

```

        plotter7.removeAllGraph();
        plotter8.removeAllGraph();

        if (bar!=null) {
            bar.setMinimum(0);
            bar.setMaximum(points.size()-1);
        }

        if (bar!=null) {
            bar.setValue(bar.getMaximum());
        }

        compute();
    }

    private void compute() {
        int count = points.size();

        double delta = points.get(1).getTetta()-points.get(0).getTetta();

        double[] ksi0 = new double[count];
        double[] psi0 = new double[count];
        double[] etta0 = new double[count];

        //initialization
        for(int i=0; i<points.size(); i++) {
            Measurement measure = points.get(i);
            ksi0[i] = measure.getKsi();
            psi0[i] = measure.getPsi();
            etta0[i] = measure.getEtta();

            if (Thread.interrupted()) return;
        }

        //compute derivatives
        double[] ksi0Derivative = getDerivative5(ksiP0.get(), delta, ksi0);
        double[] psi0Derivative = getDerivative5(psiP0.get(), delta, psi0);
        double[] etta0Derivative = getDerivative5(ettaP0.get(), delta, etta0);

        double[] ksi0SecondDerivative = getDerivative5(ksiP00.get(), delta,
            ksi0Derivative);
        double[] psi0SecondDerivative = getDerivative5(psiP00.get(), delta,
            psi0Derivative);
        double[] etta0SecondDerivative = getDerivative5(ettaP00.get(), delta,
            etta0Derivative);

        //variable values
        double[] ksi = new double[ksi0.length];
        double[] psi = new double[psi0.length];
        double[] etta = new double[etta0.length];

        double[] ksiDerivative = new double[ksi0.length];

```

```
double[] psiDerivative = new double[psi0.length];
double[] ettaDerivative = new double[etta0.length];
```

```
ksi[0] = ksiP0.get();
psi[0] = psiP0.get();
etta[0] = ettaP0.get();
```

```
ksiDerivative[0] = ksiP00.get();
psiDerivative[0] = psiP00.get();
ettaDerivative[0] = ettaP00.get();
```

```
for(int i=0; i<psi0.length-1; i++) {
    if (Thread.interrupted()) return;
```

```
    double[][] matrixP = new double[3][3];
    matrixP[0][0] = Math.sin(psi[i]);
    matrixP[1][0] = Math.sin(psi[i]+2d*Math.PI/3d);
    matrixP[2][0] = Math.sin(psi[i]+4d*Math.PI/3d);
    matrixP[0][1] = -Math.cos(psi[i]);
    matrixP[1][1] = -Math.cos(psi[i]+2d*Math.PI/3d);
    matrixP[2][1] = -Math.cos(psi[i]+4d*Math.PI/3d);
    matrixP[0][2] = -a.get();
    matrixP[1][2] = -a.get();
    matrixP[2][2] = -a.get();
```

```
    double[][] matrixP0 = new double[3][3];
    matrixP0[0][0] = Math.sin(psi0[i]);
    matrixP0[1][0] = Math.sin(psi0[i]+2d*Math.PI/3d);
    matrixP0[2][0] = Math.sin(psi0[i]+4d*Math.PI/3d);
    matrixP0[0][1] = -Math.cos(psi0[i]);
    matrixP0[1][1] = -Math.cos(psi0[i]+2d*Math.PI/3d);
    matrixP0[2][1] = -Math.cos(psi0[i]+4d*Math.PI/3d);
    matrixP0[0][2] = -a.get();
    matrixP0[1][2] = -a.get();
    matrixP0[2][2] = -a.get();
```

```
    double[][] matrixP0Inverse = new double[3][3];
    matrixP0Inverse[0][0] = 2d/3d*(Math.sin(psi0[i]));
    matrixP0Inverse[1][0] = -2d/3d*(Math.cos(psi0[i]));
    matrixP0Inverse[2][0] = -(1d/(3d*a.get()));
    matrixP0Inverse[0][1] = 1d/3d*(Math.cos(psi0[i])*Math.sqrt(3d)
        -Math.sin(psi0[i]));
    matrixP0Inverse[1][1] = 1d/3d*(Math.sin(psi0[i])*Math.sqrt(3d)
        +Math.cos(psi0[i]));
    matrixP0Inverse[2][1] = -(1d/(3d*a.get()));
    matrixP0Inverse[0][2] = -1d/3d*(Math.cos(psi0[i])*Math.sqrt(3d)
        +Math.sin(psi0[i]));
    matrixP0Inverse[1][2] = 1d/3d*(Math.cos(psi0[i])
        -Math.sin(psi0[i])*Math.sqrt(3d));
    matrixP0Inverse[2][2] = -(1d/(3d*a.get()));
```

```

/* PU=P(U0+Ux)=P(ksi)(P(-1)(ksi0)L0 + P(-1)(ksi)V) = P(ksi)P(-1)(ksi0)L0 + V */
double[] vectorL0 = new double[3];
vectorL0[0] = m.get()*ksi0SecondDerivative[i]+h.get()*ksi0Derivative[i]
             +md.get()*etta0Derivative[i]*psi0Derivative[i];
vectorL0[1] = m.get()*etta0SecondDerivative[i]
             +h.get()*etta0Derivative[i]
             -md.get()*ksi0Derivative[i]*psi0Derivative[i];
vectorL0[2] = ind.get()*psi0SecondDerivative[i]
             +2*a.get()*a.get()*h.get()*psi0Derivative[i];

double[] vectorU0 = matrixMultiplyVector(matrixP0Inverse, vectorL0);

double[] vectorV = new double[3];
vectorV[0] = -k1.get()*(ksiDerivative[i]-ksi0Derivative[i])-k2.get()
             *(ksi[i]-ksi0[i]);
vectorV[1] = -k1.get()*(ettaDerivative[i]-etta0Derivative[i])-k2.get()
             *(etta[i]-etta0[i]);
vectorV[2] = -k3.get()*(psiDerivative[i]-psi0Derivative[i])-k4.get()
             *(psi[i]-psi0[i]);

double[] vectorPU0 = matrixMultiplyVector(matrixP, vectorU0);

double[] vectorOdeRight = vectorPlusVector(vectorPU0, vectorV);

ksiDerivative[i+1] = (vectorOdeRight[0] - h.get()*ksiDerivative[i]
                    - md.get()*psiDerivative[i]*ettaDerivative[i])
                    *delta/m.get() + ksiDerivative[i];
ettaDerivative[i+1] = (vectorOdeRight[1] - h.get()*ettaDerivative[i]
                    + md.get()*psiDerivative[i]*ksiDerivative[i])
                    *delta/m.get() + ettaDerivative[i];
psiDerivative[i+1] = (vectorOdeRight[2]
                    - 2*a.get()*a.get()*h.get()*psiDerivative[i])
                    *delta/ind.get() + psiDerivative[i];

ksi[i+1] = ksi[i] + delta*ksiDerivative[i];
etta[i+1] = etta[i] + delta*ettaDerivative[i];
psi[i+1] = psi[i] + delta*psiDerivative[i];

}

//plot graph
plotter1.putGraph("ksi0", getGraphTettaY(delta, ksi0, "ξo",
new Color(30, 30, 30)));
plotter1.putGraph("ksi", getGraphTettaY(delta, ksi, "ξ",
new Color(108, 191, 76)));

plotter2.putGraph("etta0", getGraphTettaY(delta, etta0, "ηo",
new Color(30, 30, 30)));
plotter2.putGraph("etta", getGraphTettaY(delta, etta, "η",

```

```

        new Color(95, 100, 255)));
    plotter3.putGraph("psi0", getGraphTettaY(delta, psi0, " $\psi_0$ ",
        new Color(30, 30, 30)));
    plotter3.putGraph("psi", getGraphTettaY(delta, psi, " $\psi$ ",
        new Color(255, 127, 67)));
    plotter4.putGraph("ksi0Derivative", getGraphTettaY(delta, ksi0Derivative, " $\xi'_0$ ",
        new Color(30, 30, 30)));
    plotter4.putGraph("ksiDerivative", getGraphTettaY(delta, ksiDerivative, " $\xi$ ",
        new Color(108, 191, 76)));
    plotter5.putGraph("etta0Derivative", getGraphTettaY(delta, etta0Derivative,
        " $\eta'_0$ ", new Color(30, 30, 30)));
    plotter5.putGraph("ettaDerivative", getGraphTettaY(delta, ettaDerivative, " $\eta$ ",
        new Color(95, 100, 255)));
    plotter6.putGraph("psi0Derivative", getGraphTettaY(delta, psi0Derivative,
        " $\psi'_0$ ", new Color(30, 30, 30)));
    plotter6.putGraph("psiDerivative", getGraphTettaY(delta, psiDerivative, " $\psi$ ",
        new Color(255, 127, 67)));
    plotter7.putGraph("ettaKsi", getGraphXY(etta0, ksi0, " $\eta$ ", " $\xi$ ",
        new Color(30, 30, 30)));
    plotter7.putGraph("etta0Ksi0", getGraphXY(etta, ksi, " $\eta_0$ ", " $\xi_0$ ",
        new Color(255, 33, 201)));
    plotter8.putGraph("diffKsi", getGraphTettaY(delta, getDiff(ksi, ksi0), " $\xi-\xi_0$ ",
        new Color(108, 191, 76)));
    plotter8.putGraph("diffEtta", getGraphTettaY(delta, getDiff(etta, etta0), " $\eta-\eta_0$ ",
        new Color(95, 100, 255)));
    plotter8.putGraph("diffPsi", getGraphTettaY(delta, getDiff(psi, psi0), " $\psi-\psi_0$ ",
        new Color(255, 127, 67)));

```

```

    plotter1.repaint();
    plotter2.repaint();
    plotter3.repaint();
    plotter4.repaint();
    plotter5.repaint();
    plotter6.repaint();
    plotter7.repaint();
    plotter8.repaint();

```

```

}

```

```

private double[] getDiff(double[] vector1, double[] vector2) {
    int count = vector1.length;
    double[] diffVector = new double[count];
    for (int i=0; i<count; i++) {
        diffVector[i] = vector1[i]-vector2[i];
    }
    return diffVector;
}

```



```

private Graph getGraphTettaY(double delta,double[]vector, String vectorName,
    Color color) {
    Graph graph = new Graph(vectorName, "t", color);
    double time = 0;
    for (Double v : vector) {
        if (v.isNaN()) v=0d;
        if (v.isInfinite()) v=0d;
        graph.add(new Point2D(time, v));
        time += delta;
    }
    return graph;
}

```

```

private Graph getGraphXY(double[] x, double[] y, String xName, String yName,
    Color color) {
    Graph graph = new Graph(yName, xName, color);
    for (int i=0; i<x.length; i++) {
        graph.add(new Point2D(x[i], y[i]));
    }
    return graph;
}

```

```

private double[] getDerivative5(double startPoint, double delta, double[] vector) {
    double returnVector[] = new double[vector.length];
    returnVector[0] = startPoint;
    returnVector[1] = (vector[1] - vector[0]) / delta;
    returnVector[vector.length-1] = (vector[vector.length-1]
        - vector[vector.length-2]) / delta;;
    returnVector[vector.length-2] = (vector[vector.length-2]
        - vector[vector.length-3]) / delta;;
    for(int i=2; i<vector.length-2; i++) {
        returnVector[i] = (vector[i-2] - 8*vector[i-1] + 8*vector[i+1]
            - vector[i+2]) / (12*delta);
    }
    return returnVector;
}

```

```

private double[] matrixMultiplyVector(double[][] matrix, double[] vector) {
    int dimension = vector.length;
    double[] returnVector = new double[dimension];
    for (int y=0; y<dimension; y++) {
        double result = 0d;
        for (int x=0; x<dimension; x++) {
            result += matrix[x][y] * vector[x];
        }
        returnVector[y] = result;
    }
    return returnVector;
}

```

```

private double[] vectorPlusVector(double[] vector1, double[] vector2) {

```

```

        int dimension = vector1.length;
        double[] returnVector = new double[dimension];
        for (int x=0; x<dimension; x++) {
            returnVector[x] = vector1[x] + vector2[x];
        }
        return returnVector;
    }
}
}

```

Листинг 2. Ядро матпакета

```

package ru.jcup.saa.science.trowh.utils.arithmetic;

import java.util.regex.Pattern;

import ru.jcup.saa.science.trowh.utils.arithmetic.value.Value;

public class Arithmetic {

    private static final String BRACKET_SUFFIX = "bc";
    private static final String BRACKET_START = "(";
    private static final String BRACKET_END = ")";

    private static final Object[] OPERAND = new Object[5];

    private GlobalVar<String> globalVars = new GlobalVar<String>();

    public Arithmetic() {
        //init operands
        //операнды перечислены по уровню значимости
        OPERAND[0] = new byte[] {'('};
        OPERAND[1] = new byte[] {'>'};
        OPERAND[2] = new byte[] {'^'};
        OPERAND[3] = new byte[] {'*', '/'};
        OPERAND[4] = new byte[] {'+', '-'};
    }

    public Value compile(String expression) {
        splitToSequence(expression);
        Value valueTree = new Value(globalVars);
        //globalVars.showAllVars();
        globalVars.clear();
        return valueTree;
    }

    /*
     * Выражение в скобках заменяем на переменные
     */
    private int splitToSequence(String expression) {
        int varCount=0;
    }

```

```

int end, begin;
while (countOperand(expression, 0)>0) {
    end = expression.indexOf(BRACKET_END);
    begin = expression.lastIndexOf(BRACKET_START, end);
    String bracket = expression.substring(begin+1, end);

    String lastSimpleVar = splitToSimpleActions(bracket, varCount);
    String newVarName;
    boolean isUserVar = (countOperand(bracket)==0);
    if (isUserVar) {
        newVarName = bracket;
    } else {
        newVarName = BRACKET_SUFFIX + varCount;
        addNewVar(newVarName, lastSimpleVar);
    }

    String searchAction = BRACKET_START + bracket + BRACKET_END;
    expression = expression.replaceFirst(Pattern.quote(searchAction),
        newVarName);
    if (!isUserVar) {
        varCount++;
    }
}

expression = splitToSimpleActions(expression, varCount);
addNewVar(BRACKET_SUFFIX + varCount, expression);

return varCount;
}

/*
 * Разрезаем выражение на простые операции
 */
private String splitToSimpleActions(String expression, int varCount) {
    if (countOperand(expression)>1) {
        int exCount = 0;
        for (int level=1; level<OPERAND.length; level++) {
            while (countOperand(expression, level)>0) {
                int index = getOperandIndex(expression, level);
                String simpleAction = getSimpleAction(expression, index);

                String newVarName = BRACKET_SUFFIX + varCount + "_"
                    + exCount;
                addNewVar(newVarName, simpleAction);
                expression = expression.replaceFirst(
                    Pattern.quote(simpleAction),
                    newVarName);

                exCount++;

                if (countOperand(expression)==1) return expression;
            }

```

```

        }
    }
    return expression;
}

/*
 * Выбираем простое выражение типа <var><операнд><var> по индексу операнда
 */
private String getSimpleAction(String expression, int index) {
    int indexLeft = getOperandIndexLeft(expression, index);
    int indexRight = getOperandIndexRight(expression, index);
    return expression.substring(indexLeft, indexRight);
}

/*
 * Поиск любого операнда слева относительно заданного
 */
private int getOperandIndexLeft(String expression, int index) {
    int indexLeft=0;
    for (int level=1; level<OPERAND.length; level++) {
        byte[] operand = (byte[]) OPERAND[level];
        for (int i=0; i<operand.length; i++) {
            int indexTemp = expression.lastIndexOf(operand[i], index-1);
            if (indexTemp!=-1) {
                indexLeft = Math.max(indexLeft, indexTemp+1);
            }
        }
    }
    return indexLeft;
}

/*
 * Поиск любого операнда справа относительно заданного
 */
private int getOperandIndexRight(String expression, int index) {
    int indexLeft=expression.length();
    for (int level=1; level<OPERAND.length; level++) {
        byte[] operand = (byte[]) OPERAND[level];
        for (int i=0; i<operand.length; i++) {
            int indexTemp = expression.indexOf(operand[i], index+1);
            if (indexTemp!=-1) {
                indexLeft = Math.min(indexLeft, indexTemp);
            }
        }
    }
    return indexLeft;
}

/*
 * Поиск операнда по заданному уровню операнда
 */
private int getOperandIndex(String expression, int levelOperand) {

```

```

        byte[] operand = (byte[]) OPERAND[levelOperand];
        for (int i=0; i<operand.length; i++) {
            int index = expression.indexOf(operand[i]);
            if (index!=-1) return index;
        }
        return -1;
    }

    /*
     * Добавляем новые переменные в массив GlobalVar
     */
    private void addNewVar(String varName, String varValue) {
        globalVars.add(varName, varValue);
    }

    /*
     *
     */
    private int countOperand(String expression, int level) {
        byte[] operand = (byte[]) OPERAND[level];
        int count = 0;
        byte[] expressionByte = expression.getBytes();
        for (byte partExp : expressionByte) {
            for (byte operandPart : operand) {
                if (partExp == operandPart) {
                    count++;
                }
            }
        }
        return count;
    }

    /*
     * То-же, только поиск по всем уровням
     */
    private int countOperand(String expression) {
        int count=0;
        for (int level=0; level<OPERAND.length; level++) {
            count+=countOperand(expression, level);
        }
        return count;
    }
}

```

Листинг 2.1. Реализация переменных в математическом пакете

```

package ru.jcup.saa.science.trowh.utils.arithmetic;

import java.util.HashMap;
import java.util.Map;

```

```

public class GlobalVar<E> {

    private HashMap<String, E> hm = new HashMap<String, E>();
    private String lastAddVariable = null;

    public void clear() {
        lastAddVariable = null;
        hm.clear();
    }

    public void add(String variable, E value) {
        //System.out.println("! " + variable + " = " + value.toString());
        lastAddVariable = variable;

        if (!isset(variable)) {
            hm.put(variable, value);
        } else {
            hm.remove(variable);
            hm.put(variable, value);
        }
    }

    public E get(String variable) {
        if (isset(variable)) {
            return hm.get(variable);
        } else {
            hm.put(variable, null);
            return null;
        }
    }

    public boolean isset(String variable) {
        return hm.containsKey(variable);
    }

    public void showAllVars() {
        System.out.println("Global variables:");
        for (Map.Entry<String, E> entry : hm.entrySet()) {
            String var = entry.getKey();
            E value = entry.getValue();
            System.out.println(" " + var + " =\t" + value + ";");
        }
    }

    public String getNameLastAddVar() {
        return lastAddVariable;
    }

    public HashMap<String, E> getCollection() {
        return hm;
    }
}

```

```
}
```

Листинг 2.2. Реализация частей формулы в математическом пакете
--

```
package ru.jcup.saa.science.trowh.utils.arithmetic.value;

import java.util.HashMap;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import ru.jcup.saa.science.trowh.utils.arithmetic.GlobalVar;
import ru.jcup.saa.science.trowh.utils.arithmetic.value.functions.CosValue;
import ru.jcup.saa.science.trowh.utils.arithmetic.value.functions.CtgValue;
import ru.jcup.saa.science.trowh.utils.arithmetic.value.functions.ExpValue;
import ru.jcup.saa.science.trowh.utils.arithmetic.value.functions.LnValue;
import ru.jcup.saa.science.trowh.utils.arithmetic.value.functions.SinValue;
import ru.jcup.saa.science.trowh.utils.arithmetic.value.functions.TgValue;
import ru.jcup.saa.science.trowh.utils.arithmetic.value.functions.ToradValue;
import ru.jcup.saa.science.trowh.utils.arithmetic.value.numbers.DoubleValue;
import ru.jcup.saa.science.trowh.utils.arithmetic.value.numbers.LiveValue;
import ru.jcup.saa.science.trowh.utils.arithmetic.value.primitives.DecValue;
import ru.jcup.saa.science.trowh.utils.arithmetic.value.primitives.DivValue;
import ru.jcup.saa.science.trowh.utils.arithmetic.value.primitives.IncValue;
import ru.jcup.saa.science.trowh.utils.arithmetic.value.primitives.MulValue;
import ru.jcup.saa.science.trowh.utils.arithmetic.value.primitives.SqrtValue;

public class Value extends ValueAbstract {

    private GlobalVar<String> globalAutoVar;
    //private GlobalVar<Double> globalUserVar;
    private GlobalVar<LinkedDouble> globalUserVar;
    private ValueAbstract treeValue;

    public Value(GlobalVar<String> globalVar) {
        this.globalAutoVar = globalVar;
        //this.globalUserVar = new GlobalVar<Double>();
        this.globalUserVar = new GlobalVar<LinkedDouble>();
        compile();
    }

    private void compile() {
        String lastVariableName = this.globalAutoVar.getNameLastAddVar();
        treeValue = getType(this.globalAutoVar.get(lastVariableName));
    }

    public ValueAbstract getType(String phrase) {
```

```

String numberRegex = "[0-9]+((\\.)?[0-9]+)?";
String varRegex = "[a-z][a-z0-9_]*";
String varOrNumberRegex = "(" + numberRegex + ")|(" + varRegex + ")";
String expressintSeparatorRegex = "((\\+)|(\\-)|(\\V)|(\\*)|(\\^))";

String expressionRegex = "^" + varOrNumberRegex + expressintSeparatorRegex
    + varOrNumberRegex + "$";
String valueRegex = "^" + numberRegex + "$";
String variableRegex = "^" + varRegex + "$";
String functionRegex = "^[a-z][a-z0-9_]*\\>" + varOrNumberRegex + "$";

//Для переменных
if (isMatchRegex(variableRegex, phrase)) {
    String valuePhrase = this.globalAutoVar.get(phrase);
    if (valuePhrase != null) {
        phrase = valuePhrase;
    } else {
        //Ссылка на живую переменную
        //ProxyLiveValue liveValue = new ProxyLiveValue(globalUserVar,
            phrase);
        //liveValue.setThisLink(liveValue);
        //LiveNoValue liveValue = new LiveNoValue(globalUserVar, phrase);
        LiveValue liveValue = new LiveValue(globalUserVar, phrase);
        return liveValue;
    }
}

}

//Для выражений
if (isMatchRegex(expressionRegex, phrase)) {
    String expressionLeft = getValueFromExpression(phrase, LEFT);
    String expressionRight = getValueFromExpression(phrase, RIGHT);

    if (phrase.indexOf('+') != -1) {
        return new IncValue(expressionLeft, expressionRight, this);
    }
    if (phrase.indexOf('-') != -1) {
        return new DecValue(expressionLeft, expressionRight, this);
    }
    if (phrase.indexOf('/') != -1) {
        return new DivValue(expressionLeft, expressionRight, this);
    }
    if (phrase.indexOf('*') != -1) {
        return new MulValue(expressionLeft, expressionRight, this);
    }
    if (phrase.indexOf('^') != -1) {
        return new SqrtValue(expressionLeft, expressionRight, this);
    }
}

}

//Для чисел

```



```

    if (isMatchRegex(valueRegex, phrase)) {
        return new DoubleValue(Double.parseDouble(phrase));
    }

    //Для функций
    if (isMatchRegex(functionRegex, phrase)) {
        String expressionRight = getValueFromExpression(phrase, RIGHT);
        //System.err.println(">>>" + phrase);
        if (phrase.indexOf("sin")!=-1) {
            return new SinValue(expressionRight, this);
        }
        if (phrase.indexOf("cos")!=-1) {
            return new CosValue(expressionRight, this);
        }
        if (phrase.indexOf("ctg")!=-1) {
            return new CtgValue(expressionRight, this);
        }
        if (phrase.indexOf("tg")!=-1) {
            return new TgValue(expressionRight, this);
        }
        if (phrase.indexOf("exp")!=-1) {
            return new ExpValue(expressionRight, this);
        }
        if (phrase.indexOf("ln")!=-1) {
            return new LnValue(expressionRight, this);
        }
        if (phrase.indexOf("torad")!=-1) {
            return new ToradValue(expressionRight, this);
        }
        //if (phrase.indexOf("log")!=-1) {
        //    return new LogValue(expressionRight, this);
        //}
    }

    return null;
}

```

```

public static final int LEFT = 0;
public static final int RIGHT = 1;

```

```

private static String getValueFromExpression(String phrase, int direction) {
    String expressintSeparatorRegex = "((\\+)|(\\-)|(\\V)|(\\*)|(\\^)|(\\>))";
    int actionIndex = findMatch(phrase, expressintSeparatorRegex);
    if (direction==LEFT) {
        return phrase.substring(0, actionIndex);
    } else {
        return phrase.substring(actionIndex+1, phrase.length());
    }
}

```

```

private static int findMatch(String phrase, String regex) {
    Pattern pattern = Pattern.compile(regex);

```

```

Matcher matcher = pattern.matcher(phrase);
matcher.find();
return matcher.start();
}

```

```

private boolean isMatchRegex(String regex, String text) {
Pattern p = Pattern.compile(regex);
Matcher m = p.matcher(text);
return m.matches();
}

```

```

@Override
public Double calc() {
return treeValue.calc();
}

```

```

public void setUserVar(String name, LinkedDouble value) {
globalUserVar.add(name, value);
}
public void setUserVar(String name, Double value) {
globalUserVar.add(name, new LinkedDouble(value));
}
public void setUserVar(String name, int value) {
globalUserVar.add(name, new LinkedDouble(value*1D));
}

```

```

public LinkedDouble getUserVar(String name) {
return globalUserVar.get(name);
}

```

```

public void showUserVars() {
globalUserVar.showAllVars();
}

```

```

/*
* Импортируем коллекцию извне
*/

```

```

public void setUserVars(GlobalVar<Double> newGlobalVar) {
this.globalUserVar.clear();
HashMap<String, Double> newCollection = newGlobalVar.getCollection();
for (Map.Entry<String, Double> entry : newCollection.entrySet()) {
String key = entry.getKey();
Double value = entry.getValue();
this.globalUserVar.add(key, new LinkedDouble(value));
}
}

```

```

}

```

Листинг 2.3. Реализация функций математического пакета.

```
package ru.jcup.saa.science.trowh.utils.arithmetic.value.functions;

import ru.jcup.saa.science.trowh.utils.arithmetic.value.Value;
import ru.jcup.saa.science.trowh.utils.arithmetic.value.ValueAbstract;

public class CosValue extends ValueAbstract {

    public CosValue(String value1, Value startValue) {
        //System.out.println(String.format("new cos(%s)", value1));
        this.argument1 = startValue.getType(value1);
    }

    @Override
    public Double calc() {
        return Math.cos(this.argument1.calc());
    }

}
```

Листинг 2.4. Реализация операций математического пакета.

```
package ru.jcup.saa.science.trowh.utils.arithmetic.value.primitives;

import ru.jcup.saa.science.trowh.utils.arithmetic.value.Value;
import ru.jcup.saa.science.trowh.utils.arithmetic.value.ValueAbstract;

public class DecValue extends ValueAbstract {

    public DecValue(String value1, String value2, Value startValue) {
        //System.out.println(String.format("new dec(%s, %s)", value1, value2));
        this.argument1 = startValue.getType(value1);
        this.argument2 = startValue.getType(value2);
    }

    @Override
    public Double calc() {
        return this.argument1.calc() - this.argument2.calc();
    }

}
```

Листинг 3. Реализация модуля ввода траектории

```
package ru.jcup.saa.science.trowh.gui.input.trajectory;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Cursor;
```

```
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Polygon;
import java.awt.Shape;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import java.awt.event.MouseWheelEvent;
import java.awt.event.MouseWheelListener;
import java.util.ArrayList;
import java.util.Iterator;
```

```
import javax.swing.JMenuItem;
import javax.swing.JPanel;
import javax.swing.JPopupMenu;
```

```
import ru.jcup.saa.science.trowh.Settings;
import ru.jcup.saa.science.trowh.gui.input.InputListener;
import ru.jcup.saa.science.trowh.gui.input.Inputable;
import ru.jcup.saa.science.trowh.gui.input.file.Measurement;
import ru.jcup.saa.science.trowh.utils.GraphicsQuality;
import ru.jcup.saa.science.trowh.utils.font.ShapeFont;
```

```
public class TrajectoryInput extends JPanel implements MouseMotionListener, MouseListener,
MouseWheelListener, Inputable {
```

```
    private static final float LINE_WIDTH = 4f;
    private static final int PSET_WIDTH = 12;
```

```
    private static final long serialVersionUID = 1L;
```

```
    private ArrayList<Point3D> points = new ArrayList<Point3D>();
```

```
    private final int YX_CANVAS = 0;
    private final int YZ_CANVAS = 1;
    private int canvasMode = YX_CANVAS;
```

```
    //crosshair standarts
    private int crosshairSize = 30;
    private int crosshairOffset = 20;
    private int arrowSize = 8;
    private int fontSize = 20;
```

```
    public static final ShapeFont TRAJECTORY_FONT = new
ShapeFont(Settings.SYSTEM_FONT, 0);
```

```
    private boolean isDebug = false;
    private double mouseX, mouseY;
    private double mouseXpixel, mouseYpixel;
    private boolean isLeftDown, isCenterDown, isRightDown;
```

```

private int fieldWidht;
private int fieldHeight;
private int fieldDepth;
private double scaleX;
private double scaleY;
private double scaleZ;
private double scaleMaster = 1;
private int offsetX = 0;
private int offsetY = 0;
private int offsetZ = 0;

private int pointSelected;
private boolean isPointDragging = false;

private boolean isCanvasDragging = false;
private double moveFromMouseX, moveFromMouseY;
private int currentOffsetX, currentOffsetY, currentOffsetZ;

private JPanel thisPanel = this;

public TrajectoryInput(int widht, int height, int depth) {
    this.fieldWidht = widht;
    this.fieldHeight = height;
    this.fieldDepth = height;

    this.addMouseListener(this);
    this.addMouseMotionListener(this);
    this.addMouseWheelListener(this);

    if (isDebug) {
        points.add(new Point3D(10,10,10));
        points.add(new Point3D(100,10,100));
        points.add(new Point3D(150,150,150));
        points.add(new Point3D(170,170,170));
    }
}

@Override
public void paint(Graphics g) {
    Graphics2D g2d = (Graphics2D) g;
    GraphicsQuality.setHiQuality(g2d);

    int componentWidth = this.getWidth();
    int componentHeight = this.getHeight();

    g2d.setColor(Settings.MOUSE_INPUT_BACKGROUND);
    g2d.fillRect(0, 0, componentWidth, componentHeight);

    scaleY = 1d*componentHeight/this.fieldHeight;
    scaleX = 1d*componentWidth/this.fieldWidht;
    scaleZ = 1d*componentWidth/this.fieldDepth;

```

```

if (points.size()>0) {
    //Draw lines
    g2d.setStroke(new BasicStroke(LINE_WIDTH));
    Point3D pointBegin = points.get(0);
    Point3D pointEnd;
    for (int i=1; i<points.size(); i++) {
        pointEnd = points.get(i);

        double y1 = (pointBegin.getY() + offsetY) * scaleY*scaleMaster;
        double y2 = (pointEnd.getY() + offsetY) * scaleY*scaleMaster;

        double x1;
        double x2;

        if (canvasMode == YX_CANVAS) {
            x1 = (pointBegin.getX() + offsetX) * scaleX*scaleMaster;
            x2 = (pointEnd.getX() + offsetX) * scaleX*scaleMaster;
        } else {
            x1 = (pointBegin.getZ() + offsetZ) * scaleZ*scaleMaster;
            x2 = (pointEnd.getZ() + offsetZ) * scaleZ*scaleMaster;
        }

        g2d.setColor(Settings.LINE);
        g2d.drawLine((int)x1, (int)y1, (int)x2, (int)y2);

        pointBegin = pointEnd;
    }
}

//Draw coordinate crosshair
int crooshairX = crosshairOffset;
int crosshairY = componentHeight - crosshairOffset - crosshairSize;
g2d.setStroke(new BasicStroke(1f));
if (canvasMode == YX_CANVAS) {
    g2d.setColor(new Color(199, 165, 132));
} else {
    g2d.setColor(new Color(132, 165, 199));
}
g2d.fillRect(crooshairX, crosshairY, crosshairSize, crosshairSize);
g2d.setColor(Color.BLACK);
g2d.drawRect(crooshairX, crosshairY, crosshairSize, crosshairSize);
//up arrow
g2d.drawLine(crooshairX + crosshairSize/2, crosshairY - crosshairSize,
             crooshairX + crosshairSize/2, crosshairY + crosshairSize);
g2d.drawLine(crooshairX + crosshairSize/2, crosshairY - crosshairSize,
             crooshairX + crosshairSize/2 - arrowSize/2, crosshairY - crosshairSize
             + arrowSize);
g2d.drawLine(crooshairX + crosshairSize/2, crosshairY - crosshairSize,
             crooshairX + crosshairSize/2 + arrowSize/2, crosshairY - crosshairSize
             + arrowSize);
//right arrow

```

```

g2d.drawLine(crooshairX, crosshairY + crosshairSize/2,
              crooshairX + crosshairSize*2, crosshairY + crosshairSize/2);
g2d.drawLine(crooshairX + crosshairSize*2, crosshairY + crosshairSize/2,
              crooshairX + crosshairSize*2 - arrowSize, crosshairY + crosshairSize/2
              - arrowSize/2);
g2d.drawLine(crooshairX + crosshairSize*2, crosshairY + crosshairSize/2,
              crooshairX + crosshairSize*2 - arrowSize, crosshairY + crosshairSize/2
              + arrowSize/2);
g2d.setStroke(new BasicStroke(1f));
g2d.fill(TRAJECTORY_FONT.getFont("ξ", crooshairX + crosshairSize/2 + arrowSize
- fontSize, crosshairY - crosshairSize - fontSize - 1, fontSize, fontSize));
String horizontalAxisName;
if (canvasMode == YX_CANVAS) {
    horizontalAxisName = "ψ";
} else {
    horizontalAxisName = "η";
}
g2d.fill(TRAJECTORY_FONT.getFont(horizontalAxisName, crooshairX
+ crosshairSize*2 + 1, crosshairY + crosshairSize/2 - fontSize/2, fontSize, fontSize));

//Draw psets
for (int i=0; i<points.size(); i++) {
    g2d.setStroke(new BasicStroke(1f));

    Point3D point = points.get(i);
    double y = (point.getY() + offsetY) * scaleY*scaleMaster;

    double x;
    if (canvasMode == YX_CANVAS) {
        x = (point.getX() + offsetX) * scaleX*scaleMaster;
    } else {
        x = (point.getZ() + offsetZ) * scaleZ*scaleMaster;
    }

    int pointX = (int)x;
    int pointY = (int)y;

    g2d.setColor(Settings.PSET_BACKGROUND);
    g2d.fillRect(pointX - PSET_WIDTH/2, pointY - PSET_WIDTH/2,
                 PSET_WIDTH, PSET_WIDTH);
    g2d.setColor(Settings.PSET_FRAME);
    g2d.drawRect(pointX - PSET_WIDTH/2, pointY - PSET_WIDTH/2,
                 PSET_WIDTH, PSET_WIDTH);

    //draw pset number
    g2d.setColor(Color.WHITE);
    g2d.fill(TRAJECTORY_FONT.getFont(""+(i+1), pointX - PSET_WIDTH/2,
pointY - PSET_WIDTH/2,
PSET_WIDTH, PSET_WIDTH));

    //point cross frame
    boolean isDrawCrossPoint = false;

```

```

        if (pointX<=0) {
            pointX = 0;
            isDrawCrossPoint = true;
        }
        if (pointY<=0) {
            pointY = 0;
            isDrawCrossPoint = true;
        }
        if (pointX>=componentWidth) {
            pointX = componentWidth;
            isDrawCrossPoint = true;
        }
        if (pointY>=componentHeight) {
            pointY = componentHeight;
            isDrawCrossPoint = true;
        }
        if (isDrawCrossPoint) {
            g2d.setColor(Settings.POINT_CROSS_FRAME);
            g2d.fillOval(pointX - PSET_WIDTH/2, pointY - PSET_WIDTH/2,
                PSET_WIDTH,
                PSET_WIDTH);
        }

        //draw cross
        if (pointSelected==i && isPointDragging) {
            g2d.setStroke(new BasicStroke(1f));
            g2d.setColor(Color.black);
            g2d.drawLine((int)x, (int)y - PSET_WIDTH, (int)x, (int)y +
                PSET_WIDTH);
            g2d.drawLine((int)x - PSET_WIDTH, (int)y, (int)x + PSET_WIDTH,
                (int)y);
        }
    }

    if (isDebug) {
        g2d.drawString("offsetX:"+offsetX, 10, 10);
        g2d.drawString("offsetY:"+offsetY, 10, 20);
        g2d.drawString("offsetZ:"+offsetZ, 10, 30);
        g2d.drawString("scaleX:"+scaleX, 10, 40);
        g2d.drawString("scaleY:"+scaleY, 10, 50);
        g2d.drawString("scaleZ:"+scaleZ, 10, 60);
        g2d.drawString("masterX:"+scaleMaster, 10, 70);
        g2d.drawString("mouse:"+mouseX+"x"+mouseY, 10, 80);
        g2d.drawString("mousePixel:"+mouseXpixel+"x"+mouseYpixel, 10, 90);
    }
}

@Override
public void mouseDragged(MouseEvent mouse) {
    if (isDebug) mouseLogUpdate(mouse);

    updateMouseProperties(mouse);
}

```



```

if (isCenterDown) {
    //change offset
    if (!isCanvasDragging) {
        //start dragging. Have to remember the first point
        moveFromMouseX = mouseXpixel;
        moveFromMouseY = mouseYpixel;
        currentOffsetX = offsetX;
        currentOffsetY = offsetY;
        currentOffsetZ = offsetZ;
        isCanvasDragging = true;
    } else {
        //dragging in progress
        if (canvasMode == YX_CANVAS) {
            this.offsetX = currentOffsetX + (int)((mouseXpixel -
                moveFromMouseX) / scaleX / scaleMaster);
        } else {
            this.offsetZ = currentOffsetZ + (int)((mouseXpixel -
                moveFromMouseX) / scaleZ / scaleMaster);
        }

        this.offsetY = currentOffsetY + (int)((mouseYpixel - moveFromMouseY)
            / scaleY / scaleMaster);
        this.setCursor(Cursor.getPredefinedCursor(Cursor.MOVE_CURSOR));
        this.repaint();
    }
} else { //left or right mouse btn clckd
    if (isPointDragging) {
        //point relocation
        if (isLeftDown) {
            if (canvasMode == YX_CANVAS) {
                points.set(pointSelected, new Point3D(mouseX, mouseY,
                    points.get(pointSelected).getZ()));
            } else {
                points.set(pointSelected, new
Point3D(points.get(pointSelected).getX(), mouseY, mouseX));
            }
            makeGapPoint();
            this.repaint();
        }
    } else {
        boolean isComplete = false;

        //point click check
        Point3D point = getPointUnderMouse();
        if (point!=null) {
            pointSelected = points.indexOf(point);
            isPointDragging = true;
            isComplete = true;
        }

        //create middle point
    }
}

```

```

        // !!!double check isPointDragging !!!
        //seems too small area to click
        //if (!isPointDragging && points.size()>0) {
        if (!isComplete) {
            int index = getShapeUnderMouse();
            if (index!=-1) {
                points.add(index, new Point3D(mouseX, mouseY,
                    fieldDepth/2));
                pointSelected = index;
                isPointDragging = true;
            }
        }
    }
}

@Override
public void mouseReleased(MouseEvent arg0) {
    isPointDragging = false;
    isCanvasDragging = false;
    this.setCursor(Cursor.getPredefinedCursor(Cursor.DEFAULT_CURSOR));
    this.repaint();
}

@Override
public void mouseClicked(MouseEvent mouse) {
    if (isDebug) mouseLogUpdate(mouse);

    updateMouseProperties(mouse);

    Point3D point = getPointUnderMouse();
    if (point!=null) {
        if (isRightDown) {
            pointSelected = points.indexOf(point);

            JPopupMenu removePopup = new JPopupMenu();
            JMenuItem removePset = new JMenuItem("Удалить эту точку");
            removePopup.add(removePset);
            removePset.addActionListener(new ActionListener(){
                @Override
                public void actionPerformed(ActionEvent e) {
                    points.remove(pointSelected);
                    thisPanel.repaint();
                }
            });
            removePopup.show(mouse.getComponent(), mouse.getX(),
mouse.getY());
        }
    } else {
        if (isRightDown) {
            JPopupMenu removePopup = new JPopupMenu();

```

```

        JMenuItem removeAllPset = new JMenuItem("Очистить поле");
        removePopup.add(removeAllPset);
        removeAllPset.addActionListener(new ActionListener(){
            @Override
            public void actionPerformed(ActionEvent e) {
                points.clear();
            }
        });
        removePopup.show(mouse.getComponent(), mouse.getX(),
            mouse.getY());
    }

    //ifs for axis changer
    int componentHeight = this.getHeight();
    boolean isHorizontal = (mouseXpixel >= crosshairOffset)
        && (mouseXpixel <= (crosshairOffset + crosshairSize));
    boolean isVertical = (mouseYpixel >= (componentHeight - crosshairSize -
        crosshairOffset))
        && (mouseYpixel <= (componentHeight - crosshairOffset));

    if (isLeftDown && isHorizontal && isVertical) {
        //click to axis changer
        if (canvasMode==YX_CANVAS) {
            canvasMode=YZ_CANVAS;
        } else {
            canvasMode=YX_CANVAS;
        }
        this.repaint();
    } else if (isLeftDown && !isPointDragging) {
        //new point3d
        if (canvasMode == YX_CANVAS) {
            points.add(new Point3D(mouseX, mouseY, fieldDepth/2));
        } else {
            points.add(new Point3D(fieldWidth/2, mouseY, mouseX));
        }
        makeGapPoint();
        this.repaint();
    }
}

@Override
public void mouseMoved(MouseEvent mouse) {
    //this method only set mouse cursor icon
    if (isDebug) mouseLogUpdate(mouse);

    updateMouseProperties(mouse);

    boolean isCursorSet = false;

    int index = getShapeUnderMouse();
    if (index!=-1) {

```

```

        this.setCursor(Cursor.getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));
        isCursorSet = true;
    }

    Point3D point = getPointUnderMouse();
    if (point!=null) {
        this.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
        isCursorSet = true;
    }

    //ifs for axis changer
    int componentHeight = this.getHeight();
    boolean isHorizontal = (mouseXpixel >= crosshairOffset)
        && (mouseXpixel <= (crosshairOffset + crosshairSize));
    boolean isVertical = (mouseYpixel >= (componentHeight - crosshairSize -
        crosshairOffset))
        && (mouseYpixel <= (componentHeight - crosshairOffset));
    if (isHorizontal && isVertical) {
        this.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
        isCursorSet = true;
    }

    if (!isCursorSet) {
        this.setCursor(Cursor.getPredefinedCursor(Cursor.DEFAULT_CURSOR));
    }
}

private final double masterAddition = 0.2;

@Override
public void mouseWheelMoved(MouseWheelEvent wheel) {
    mouseLogUpdate(wheel);

    this.scaleMaster = scaleMaster + masterAddition * wheel.getWheelRotation();
    if (scaleMaster < 0.1) scaleMaster = 0.1;
    if (scaleMaster > 5) scaleMaster = 5;

    double newMouseY = wheel.getPoint().getY() / scaleY / scaleMaster - offsetY;

    offsetY = offsetY + (int)(newMouseY-mouseY);

    double newMouseX;
    if (canvasMode == YX_CANVAS) {
        newMouseX = wheel.getPoint().getX() / scaleX / scaleMaster - offsetX;
        offsetX = offsetX + (int)(newMouseX-mouseX);
    } else {
        newMouseX = wheel.getPoint().getX() / scaleZ / scaleMaster - offsetZ;
        offsetZ = offsetZ + (int)(newMouseX-mouseX);
    }

    this.repaint();
}

```

```

private int getShapeUnderMouse() {
    if (points.size() > 0) {
        int componentWidth = this.getWidth();
        int componentHeight = this.getHeight();

        scaleY = 1d * componentHeight / this.fieldHeight;
        scaleX = 1d * componentWidth / this.fieldWidth;
        scaleZ = 1d * componentWidth / this.fieldHeight;

        Point3D pointBegin = points.get(0);
        Point3D pointEnd;
        for (int i = 1; i < points.size(); i++) {
            pointEnd = points.get(i);
            double y1 = (pointBegin.getY() + offsetY) * scaleY * scaleMaster;
            double y2 = (pointEnd.getY() + offsetY) * scaleY * scaleMaster;

            double x1;
            double x2;
            if (canvasMode == YX_CANVAS) {
                x1 = (pointBegin.getX() + offsetX) * scaleX * scaleMaster;
                x2 = (pointEnd.getX() + offsetX) * scaleX * scaleMaster;
            } else {
                x1 = (pointBegin.getX() + offsetZ) * scaleZ * scaleMaster;
                x2 = (pointEnd.getX() + offsetZ) * scaleZ * scaleMaster;
            }

            double deltaX = x2 - x1;
            double deltaY = y2 - y1;
            double grad = Math.toDegrees(Math.atan2(deltaY, deltaX)) + 90;

            double rad = Math.toRadians(grad);
            int offset = 7;

            int[] nx = {(int)(x1 + Math.cos(rad) * offset),
                        (int)(x1 - Math.cos(rad) * offset),
                        (int)(x2 - Math.cos(rad) * offset),
                        (int)(x2 + Math.cos(rad) * offset)};
            int[] ny = {(int)(y1 + Math.sin(rad) * offset),
                        (int)(y1 - Math.sin(rad) * offset),
                        (int)(y2 - Math.sin(rad) * offset),
                        (int)(y2 + Math.sin(rad) * offset)};

            Shape lineShape = new Polygon(nx, ny, 4);

            //Rectangle rect = lineShape.getBounds();
            //System.out.println("x1:" + rect.getX());
            //System.out.println("y1:" + rect.getY());
            //System.out.println("x2:" + rect.getMaxX());
            //System.out.println("y2:" + rect.getMaxY());

            if (lineShape.contains(mouseXpixel, mouseYpixel)) {

```

```

        return i;
    }
    pointBegin = pointEnd;
}
}
return -1;
}

private Point3D getPointUnderMouse() {
    Iterator<Point3D> it = points.iterator();
    while(it.hasNext()) {
        Point3D point = it.next();

        int x;
        if (canvasMode == YX_CANVAS) {
            x = (int)point.getX();
        } else {
            x = (int)point.getZ();
        }

        int y = (int)point.getY();
        int startX = x - PSET_WIDTH/2;
        int startY = y - PSET_WIDTH/2;
        int endX = x + PSET_WIDTH/2;
        int endY = y + PSET_WIDTH/2;

        if (mouseX >= startX && mouseX <= endX && mouseY >= startY &&
            mouseY < endY) {
            return point;
        }
    }
    return null;
}

private void mouseLogUpdate(MouseEvent mouse) {
    mouseXpixel = mouse.getPoint().getX();
    mouseYpixel = mouse.getPoint().getY();
    mouseY = mouse.getPoint().getY() / scaleY / scaleMaster - offsetY;

    if (canvasMode == YX_CANVAS) {
        mouseX = mouse.getPoint().getX() / scaleX / scaleMaster - offsetX;
    } else {
        mouseX = mouse.getPoint().getX() / scaleZ / scaleMaster - offsetZ;
    }

    this.repaint();
}

private void updateMouseProperties(MouseEvent mouse) {
    isLeftDown = (mouse.getModifiers() & MouseEvent.BUTTON1_MASK) ==
        MouseEvent.BUTTON1_MASK;
}

```

```

        isCenterDown = (mouse.getModifiers() & MouseEvent.BUTTON2_MASK) ==
            MouseEvent.BUTTON2_MASK;
        isRightDown = (mouse.getModifiers() & MouseEvent.BUTTON3_MASK) ==
            MouseEvent.BUTTON3_MASK;
        mouseYpixel = mouse.getPoint().getY();
        mouseXpixel = mouse.getPoint().getX();
        mouseY = mouseYpixel / scaleY / scaleMaster - offsetY;

        if (canvasMode == YX_CANVAS) {
            mouseX = mouseXpixel / scaleX / scaleMaster - offsetX;
        } else {
            mouseX = mouseXpixel / scaleZ / scaleMaster - offsetZ;
        }
    }

    @Override
    public void mouseEntered(MouseEvent arg0) {
    }
    @Override
    public void mouseExited(MouseEvent arg0) {
    }
    @Override
    public void mousePressed(MouseEvent arg0) {
    }

    /* @SuppressWarnings("unused")
    private void printPoint() {
        Iterator<Point3D> it = points.iterator();
        for(int i=0; i<points.size(); i++) {
            Point3D point = it.next();
            double x = point.getX();
            double y = point.getY();

        }
    }
    */

    /*
    * Input finisher
    */

    private final double INPUT_STEP = 1;

    private void makeGapPoint() {
        /*System.out.println("****");
        System.out.println("****");
        System.out.println("****");
        System.out.println("****");
        System.out.println("****");
        System.out.println("****");*/

        if (points.size()>1) {
            ArrayList<Measurement> measurement = new ArrayList<Measurement>();

```

```

double tetta = 0;
double tettaStep = 0.1;

//Iterator<Point3D> it = points.iterator();
for (int i = 1; i<points.size(); i++) {
    Point3D point1 = points.get(i-1);
    Point3D point2 = points.get(i);

    //length 3d vector
    double lx = point2.getX() - point1.getX();
    double ly = point2.getY() - point1.getY();
    double lz = point2.getZ() - point1.getZ();
    double length = Math.sqrt(lx*lx + ly*ly + lz*lz);
    //System.out.println(i + "-" + (i+1) + ": "+length);

    int step = (int) (length / INPUT_STEP);
    double lxStep = lx / step;
    double lyStep = ly / step;
    double lzStep = lz / step;

    //System.out.println("_____ " + point1.getX() + "/" + point1.getY()+ "/"
    + point1.getZ());
    //System.out.println("_____ " + point2.getX() + "/" + point2.getY()+ "/"
    + point2.getZ());

    int currentStep = 0;
    do {
        currentStep++;
        tetta+=tettaStep;
        double lxWithStep = point1.getX() + lxStep*currentStep;
        double lyWithStep = point1.getY() + lyStep*currentStep;
        double lzWithStep = point1.getZ() + lzStep*currentStep;
        if (lxWithStep!=Double.POSITIVE_INFINITY
            && lyWithStep!=Double.POSITIVE_INFINITY
            && lzWithStep!=Double.POSITIVE_INFINITY) {
            measurement.add(new Measurement(tetta, lyWithStep,
lxWithStep, lzWithStep));
        }

        //System.out.println(lxWithStep + "; " + lyWithStep + "; " +
lzWithStep);
    } while(step > currentStep);

    //System.out.println(point1.getX() + "; " + point1.getY() + "; " +
point1.getZ());
}

if (inputable!=null) {
    inputable.inputComplete(measurement);
}
}

```



```

    }

    private InputListener inputable;

    @Override
    public void setInputListener(InputListener inputable) {
        this.inputable = inputable;
    }
}

```

```

package ru.jcup.saa.science.trowh.gui.input.file;

```

```

import java.awt.BorderLayout;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;

```

```

import javax.swing.JPanel;

```

```

import ru.jcup.saa.science.trowh.gui.input.InputListener;
import ru.jcup.saa.science.trowh.gui.input.Inputable;

```

```

public class FileInput extends JPanel implements Runnable, Inputable{

```

```

    private static final long serialVersionUID = 1L;
    private File file;
    private ArrayList<Measurement> measurement = new ArrayList<Measurement>();
    private MeasurementTable table;

```

```

public FileInput() {
    this.setLayout(new BorderLayout());

//measurement table
    table = new MeasurementTable();
    this.add(table, BorderLayout.CENTER);

    //file choose panel
    FileChoose fileChoose = new FileChoose(this);
this.add(fileChoose, BorderLayout.NORTH);
}

/*
 * file load thread
 */

@Override
public void run() {
    //table.generateTable(new ArrayList<Measurement>());
    table.setLoadProcess(0);
    loadFile(file);
}

public void setFile(File file) {
    this.file = file;
}

private static final long UPDATE_LOAD_LATENCY = 300; //ms

private void loadFile(File file) {
    //File file = new File("c:\\file.csv");
    measurement.clear();

    table.setLoadProcess(0);
    updateStartTime();

    try {
        int lineCounter=0;
        FileReader in = new FileReader(file);
        BufferedReader br = new BufferedReader(in);
        while (br.ready() && !Thread.interrupted()) {
            lineCounter++;
            String line = br.readLine();
            parseMeasurementLine(line);
            updateLoad(lineCounter);
        }

        //is not interrupted
        if(!br.ready()) {
            table.generateTable(measurement);
            if (measurement.size()>2) {
                inputable.inputComplete(measurement);
            }
        }
    }
}

```

```

        }
    }

    br.close();
} catch (IOException e) {
    e.printStackTrace();
}
}

private long startTime;

private void updateLoad(int lineCounter) {
    if (System.currentTimeMillis()-startTime > UPDATE_LOAD_LATENCY) {
        table.setLoadProcess(lineCounter);
        updateStartTime();
    }
}

private void updateStartTime() {
    startTime = System.currentTimeMillis();
}

private void parseMeasurementLine(String line) {
    Scanner scanner = new Scanner(line);
    scanner.useDelimiter(";");

    double tetta = scanner.nextDouble();
    double ksi = scanner.nextDouble();
    double psi = scanner.nextDouble();
    double etta = scanner.nextDouble();

    scanner.close();

    //System.out.println(tetta + "/" + ksi + "/" + psi + "/" + etta);
    measurement.add(new Measurement(tetta, ksi, psi, etta));
}

private InputListener inputable;

@Override
public void setInputListener(InputListener inputable) {
    this.inputable = inputable;
}

}

```