**Projekt oprogramowania rysującego wykresy GeoGebry z graficznym interfejsem z użytkownikiem i graficzną prezentacją wyników.**

**1. Wymagania:**

Ideą oprogramowania jest stworzenie prostego kalkulatora graficznego, za pomocą którego możliwe jest przedstawienie wykresów funkcji opisanej odpowiednim równaniem, gdzie aktor ma możliwość wprowadzania zmiennych. Program pozwala zaobserwować dynamicznie z mieniające się krzywe.

Opis równania:

Wykresy są graficznym przedstawieniem funkcji w postaci r (t) = c + a \* cos(it) + b \* sinm(kt), gdzie a, b, c są liczbami rzeczywistymi, z aś i , k, m t o liczby całkowite nieujemne.

Koniec opisu.

**1.1 Funkcjonalność:**

Oprogramowanie użytkowane jest przez jednego aktora (użytkownika). Scenariusze działania aktora:

Scenariusz 1. (Rozpoczęcie)

Aktor uruchamia oprogramowanie. Pojawia się interfejs składający się z panelu wprowadzania danych oraz pola, w którym rysowane są wykresy. Każde miejsce, gdzie aktor wpisuje dane, j est podpisane i informuje go o kolejnych czynnościach. Aktor wpisuje różne wartości zmiennych a, b, c, i , k, m w odpowiednie luki, z aś program oblicza istniejące punkty maksymalne i minimalne, t j. ekstrema funkcji, które pomogą nam w określeniu zakresu danych potrzebnych do wykreślenia krzywych na wykresie.

Jeśli aktor chce zakończyć działanie programu, w każdym momencie jego działania naciska przycisk o nazwie „ Zakończ”.



Scenariusz 2. ( Wprowadzenie danych i wykres)

Aktor wprowadza zmienne a, b, c oraz i , k, m. Ma ponadto możliwość zmiany dziedziny, w której program rysuje krzywą, z a pomocą suwaka. Jeśli dane wprowadzone przez aktora są poprawne i zgadzają się z wcześniej określonymi założeniami, po prawej stronie pojawia się wykres w układzie kartezjańskim. W zależności od ustawionego zakresu ( z miana możliwa j est od wartości 0 do 2π). Program w trakcie działania oblicza również wartości minimalne i maksymalne tworzącej się funkcji w celu lepszej prezentacji wykresu.

Scenariusz 2a.

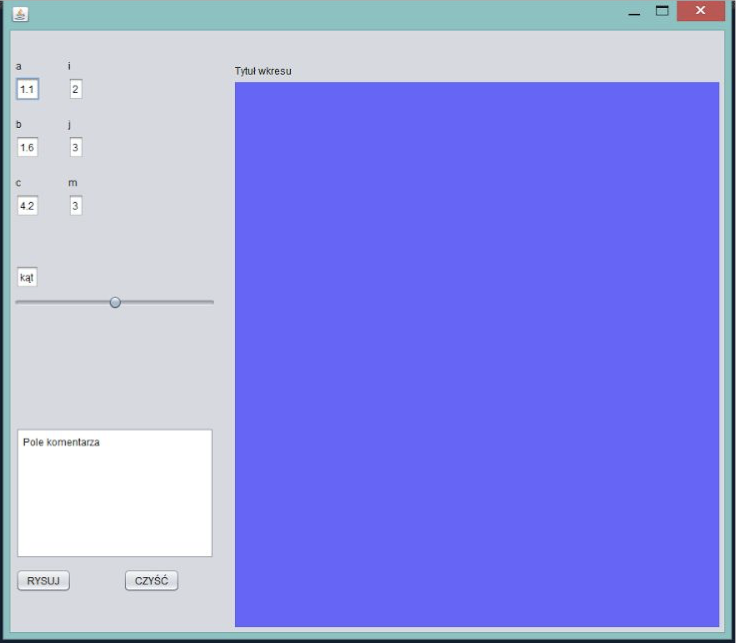
Aktor wprowadza wartości niezgodne z założeniami programu, t j. w pola i,k,m wprowadza wartości niecałkowite l ub całkowite ujemne. Wykres nie pojawia się, w polu komentarza pojawia się komunikat o błędnym wprowadzeniu zmiennych z instrukcją, która mówi aktorowi na j akie wartości należy z mienić wprowadzone dane aby program funkcjonował poprawnie. Aktor poprawia dane, program rysuje wykres.

Scenariusz 3. ( Czyszczenie)  
Aktor naciska przycisk WYCZYŚĆ. Rysunek przedstawiający wykres znika, aktor widzi tylko tło. Znikają wprowadzone wartości, w polach pojawiają się wartości przykładowe t akie same, które pojawiły się po uruchomieniu programu. Pole komentarza pozostaje puste.

Scenariusz 4. (Zakończenie)

Aktor wybiera pole, które umożliwia zakończenie działania aplikacji - aplikacja zamyka się. Interfejs użytkownika znika. Nie są gromadzone dane o działaniu aplikacji.

**1.2 GUI ( Graficzny interfejs oprogramowania):**



**1.3 Testy akceptacyjne**

Test scenariusza 1.  
Dla parametrów a=1.1; b=1.6; c=4.2; i=-2; k=3; m=3, kąt=3 pojawia się informacja o niespełnieniu założeń.

Test scenariusza 2.  
Dla parametrów a=1.1; b=1.6; c=4.2; i=2; k=3; m=3; kąt=3 pojawia się wykres oraz wartości max=6,38896 i min=1.5000

**2.Analiza**

Bazując na wymaganiach można ustalić, że ta prosta aplikacja będzie składała się z trzech klas:

* klasy głównej
* klasy rysującej
* klasy danych.

Klasa główna musi „wiedzieć” o istnieniu klasy rysującej i klasy danych. Klasa rysująca musi „wiedzieć” o istnieniu klasy danych.

**2.1 Klasa główna**Klasa jest odpowiedzialna za:

* Utworzenie okna interfejsu z użytkownikiem, zgodnie ze szkicem zawartym w wymaganiach
* Zamknięcie aplikacji
* Obsługę przycisków RYSUJ oraz CZYŚĆ
* Wprowadzenie danych z pól tekstowych
* Wyświetlenie wyniku obliczeń
* Wyświetlenie komentarzy
* Prezentację rysunku wygenerowanego przez klasę rysującą, zgodnie ze szkicem zawartym w wymaganiach.

Klasa ta może być zbudowana z wykorzystaniem narzędzia JFrame\_Form do tworzenia interfejsu graficznego w środowisku Netbeans.

**2.2 Klasa rysująca wykres**Klasa jest odpowiedzialna za:

* Wygenerowanie wykresu w układzie kartezjańskim
* Utworzeniu tytułu, opisu osi, skali na osiach oraz legendy opisującej przebiegi
* Utworzenie obiektu zgodnego z wymaganiami klasy głównej

Klasa ta może zostać zbudowana z wykorzystaniem publicznego pakietu jFreeChart.

**2.3 Klasa danych**Klasa jest odpowiedzialna za:

* Przechowywanie danych do wygenerowania wykresu. Jest to jedyne miejsce, gdzie przechowywane są dane
* Obliczenia wyniku

Diagram klas dla celów analizy budowanej aplikacji jest prosty, zawiera tylko trzy klasy. Zostanie przedstawiony, jako diagram klas dla celów projektowych, zawierający szczegóły implementacji.

**3. Projekt**

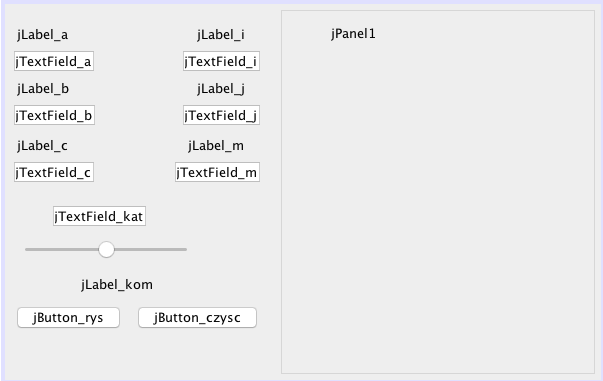
Do utworzenia wykresu w projekcie zostanie wykorzystana biblioteka graficzna *jFreeChart.*

Oprogramowanie narzędziowe wygeneruje kod, który po wykonaniu utworzy odpowiedni obiekt graficzny. Kod ten jest zawarty w metodzie initComponents(), która z kolei jest wywoływana przez konstruktor klasy głównej. Metoda initComponents() jest niemodyfikowalna przez programistę. W omawianym przykładzie klasa główna nosi nazwę Main.   
Ponadto oprogramowanie narzędziowe tworzy metodę main() w klasie Main, która powoduje uruchomienie aplikacji. Powstają też niemodyfikowalne deklaracje obiektów występujących w graficznym interfejsie. Całość oprogramowania znajduje się w pakiecie ggpack.

**3.1 ggpack/Main.java**

Klasa ta posiada znaczną liczbę atrybutów (zmiennych klasowych). Klasa ta zawiera metodę main(), generowaną przez narzędzie graficzne JFrame\_Form. Przyjęto, że większość tych obiektów, które zostaną użyte przez programistę do opisu poszczególnych funkcji otrzymają własne, znaczące nazwy, natomiast pozostałe będą miały nazwy wygenerowane przez oprogramowanie narzędziowe.

Na rysunku poniżej przedstawiono nazwy elementów utworzonych w środowisku graficznym.



Do wprowadzania danych liczbowych służą obiekty jTextField. Są to obiekty klasy javax.swing.JTextField z możliwością edytowania zawartości. Użytkownik wykorzystuje obiekty jButton\_rys oraz jButton\_czysc należące dla klasy javax.swing.JButton do rozpoczęcia odpowiedniego scenariusza. Przy definiowaniu właściwości obiektu należy zażądać wygenerowania nagłówka metody, w której będzie znajdował się kod opisujący poszczególne akcje scenariusza. Dostępne metody dla obiektu znajdują się w sekcji Event. Dla przycisku RYSUJ zostanie wygenerowany nagłówek metody rysujActionPerformed(ActionEvent e), dla przycisku CZYSC zostanie wygenerowany nagłówek metody czyscActionPerformed(ActionEvent e). Wszystkie omówione obiekty zawierają się w obiekcie (kontenerze) jPanel2 klasy javax.swing.JPanel.

**3.2 ggpack/DrawPanel.java**

Klasa DrawPanel wykorzystuję bibliotekę jFreeChart do zbudowania wykresu. Projektowany wykres należy do wykresów, gdzie użytkownik definiuje obie osie.

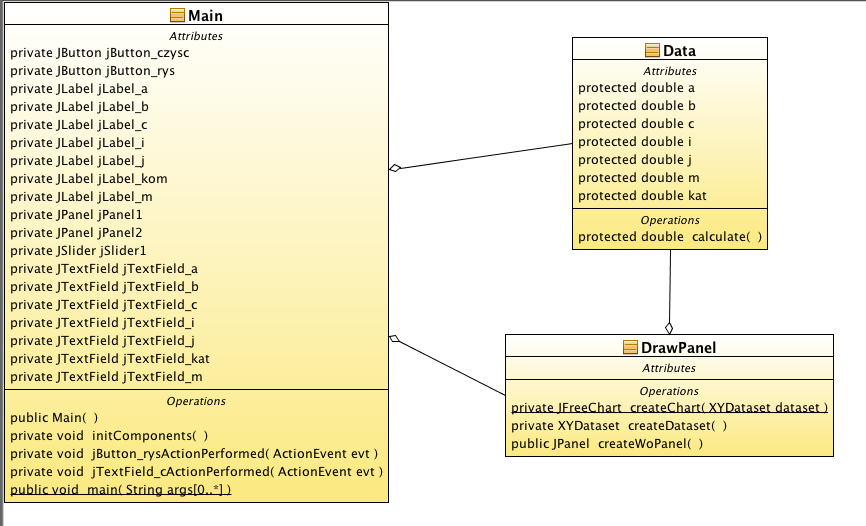
Z biblioteki wykorzystane zostały metody:

* Metoda createDataset(). W tej metodzie użytkownik definiuje dane do kreślenia umieszczając je w odpowiednich obiektach biblioteki.
* Metoda createChart(dataset). Zadaniem tej metody jest utworzenie obiektu przedstawiającego rysunek, zawierający opisy, osie, skale. Tworzony jest obiekt typu JFreeChart.
* Metoda createWoPanel(). Metoda ta wykorzystuje poprzednie dwie metody do utworzenia obiektu JFreeChart. Następnie konwertuje obiekt typu JFreeChart do postaci obiektu, którego przodkiem jest JPanel (JFreeChart dziedziczy od JPanel). Tego typu obiekt można wkleić do przygotowanego interfejsu użytkownika.

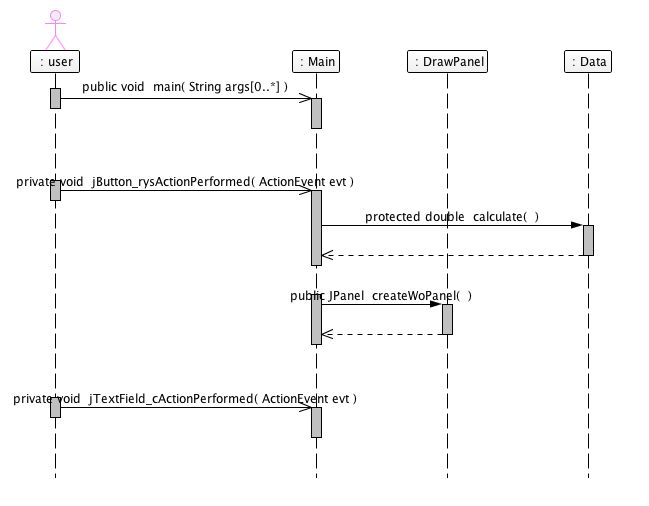
**3.3 ggpack/Data.java**

Klasa ta przechowuje dane niezbędne do przeprowadzenia obliczeń, wprowadzane przez użytkownika. Dane te są dostępne tylko wewnątrz pakietu ggpack (deklaracja static protected). Z danych tych korzystają metody z pozostałych dwóch klas. W tej klasie znajduje się statyczna metoda calculate() wykonująca obliczenia zgodnie z zależnością podaną w wymaganiach.

**3.4 Diagram klas projektu**

****

**3.5 Diagram sekwencji projektu**



**4. Implementacja**

W tym rozdziale zostaną przedstawione kody poszczególnych klas

**4.1 Klasa Main.java**

package ggpack;

import java.awt.BorderLayout;

import javax.swing.JPanel;

/\*\*

\* Klasa <code>Main</code> tworzy okno interfejsu, zamyka aplikację, obsługuje

\* przyciski RYSUJ oraz CZYŚĆ, wprowadza dane z pól tekstowych, wyświetla wyniki

\* obliczeń, komentarze i prezentuje rysunek wygenerowany przez klasę

\* <code>DrawPanel</code>.

\*

\* @author Jacek Sobiecki

\* @author Piotr Szubert

\*/

public class Main extends javax.swing.JFrame {

/\*\*

\* Creates new form GeoGebraUI

\*/

public Main() {

initComponents();

}

/\*\*

\* This method is called from within the constructor to initialize the form.

\* WARNING: Do NOT modify this code. The content of this method is always

\* regenerated by the Form Editor.

\*/

/\*\*

\* Rysuje wykres

\* @param evt obiekt klasy java.awt.event.ActionEvent

\*/

private void jButton\_rysActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

try {

Data.a = Double.parseDouble(jTextField\_a.getText());

Data.b = Double.parseDouble(jTextField\_b.getText());

Data.c = Double.parseDouble(jTextField\_c.getText());

Data.i = Integer.parseInt(jTextField\_i.getText());

Data.k = Integer.parseInt(jTextField\_k.getText());

Data.m = Integer.parseInt(jTextField\_m.getText());

double kat = Double.parseDouble(jTextField\_kat.getText());

Data.kat = (kat \* 180 / Math.PI);

if (Data.a < 0) {

throw new NumberFormatException();

}

if (Data.b < 0) {

throw new NumberFormatException();

}

if (Data.c < 0) {

throw new NumberFormatException();

}

if (Data.i < 0) {

throw new NumberFormatException();

}

if (Data.k < 0) {

throw new NumberFormatException();

}

if (Data.m < 0) {

throw new NumberFormatException();

}

if (Data.kat < 0) {

throw new NumberFormatException();

}

jPanel1.setLayout(new BorderLayout());

JPanel dp = DrawPanel.createWoPanel();

jPanel1.add(dp);

jPanel1.validate();

jLabel\_kom.setText(String.valueOf("<html>Wartosc max :" + Data.max + "<br>Wartosc min :" + Data.min + "</html>"));

jPanel1.setVisible(true);

} catch (NumberFormatException e) {

jPanel1.setVisible(false);

jLabel\_kom.setText("<html>Wprowadzone dane są ujemne <br> Wszystkie dane powinny byc > 0.0 <br> oraz wartosci i,k,m calkowite </html>");

} catch (NullPointerException e) {

jLabel\_kom.setText("Wprowadzone dane nie są liczbami");

}

}

/\*\*

\* Ustawia początkowe wartości parametrów oraz ukrywa wykres.

\* @param evt obiekt klasy java.awt.event.ActionEvent

\*/

private void jButton\_czyscActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

jPanel1.setVisible(false);

jTextField\_a.setText("1.1");

jTextField\_b.setText("1.6");

jTextField\_c.setText("4.2");

jTextField\_i.setText("2");

jTextField\_k.setText("3");

jTextField\_m.setText("3");

jTextField\_kat.setText("3");

jLabel\_kom.setText("");

}

/\*\*

\* Reaguje na zmianę położenia suwaka, ustawia wartość kąta.

\* @param evt obiekt klasy javax.swing.event.ChangeEvent

\*/

private void jSlider1StateChanged(javax.swing.event.ChangeEvent evt) {

// TODO add your handling code here:

Data.kat = 0;

Data.kat = jSlider1.getValue();

jTextField\_kat.setText(String.valueOf((jSlider1.getValue() \* 0.017453)));

}

private void jTextField\_cMousePressed(java.awt.event.MouseEvent evt) {

// TODO add your handling code here:

jTextField\_c.setText("");

}

private void jTextField\_mMousePressed(java.awt.event.MouseEvent evt) {

// TODO add your handling code here:

jTextField\_m.setText("");

}

private void jTextField\_kMousePressed(java.awt.event.MouseEvent evt) {

// TODO add your handling code here:

jTextField\_k.setText("");

}

private void jTextField\_iMousePressed(java.awt.event.MouseEvent evt) {

// TODO add your handling code here:

jTextField\_i.setText("");

}

private void jTextField\_aMousePressed(java.awt.event.MouseEvent evt) {

// TODO add your handling code here:

jTextField\_a.setText("");

}

private void jTextField\_katMousePressed(java.awt.event.MouseEvent evt) {

// TODO add your handling code here:

jTextField\_kat.setText("");

}

private void jTextField\_bMousePressed(java.awt.event.MouseEvent evt) {

// TODO add your handling code here:

jTextField\_b.setText("");

}

/\*\*

\* @param args the command line arguments

\*/

public static void main(String args[]) {

/\* Set the Nimbus look and feel \*/

//<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">

/\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.

\* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html

\*/

try {

for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {

if ("Nimbus".equals(info.getName())) {

javax.swing.UIManager.setLookAndFeel(info.getClassName());

break;

}

}

} catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(Main.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(Main.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(Main.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(Main.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

}

//</editor-fold>

//</editor-fold>

//</editor-fold>

//</editor-fold>

/\* Create and display the form \*/

java.awt.EventQueue.invokeLater(new Runnable() {

public void run() {

new Main().setVisible(true);

}

});

}

// Variables declaration - do not modify

private javax.swing.JButton jButton\_czysc;

private javax.swing.JButton jButton\_rys;

private javax.swing.JLabel jLabel1;

private javax.swing.JLabel jLabel\_a;

private javax.swing.JLabel jLabel\_b;

private javax.swing.JLabel jLabel\_c;

private javax.swing.JLabel jLabel\_i;

private javax.swing.JLabel jLabel\_k;

private javax.swing.JLabel jLabel\_kom;

private javax.swing.JLabel jLabel\_m;

private javax.swing.JPanel jPanel1;

private javax.swing.JPanel jPanel2;

private javax.swing.JSlider jSlider1;

private javax.swing.JTextField jTextField\_a;

private javax.swing.JTextField jTextField\_b;

private javax.swing.JTextField jTextField\_c;

private javax.swing.JTextField jTextField\_i;

private javax.swing.JTextField jTextField\_k;

private javax.swing.JTextField jTextField\_kat;

private javax.swing.JTextField jTextField\_m;

// End of variables declaration

}

**4.2 Klasa DrawPanel.java**

package ggpack;

import java.awt.Color;

import javax.swing.JPanel;

import org.jfree.chart.ChartFactory;

import org.jfree.chart.ChartPanel;

import org.jfree.chart.JFreeChart;

import org.jfree.chart.plot.PlotOrientation;

import org.jfree.chart.plot.XYPlot;

import org.jfree.data.xy.XYDataset;

import org.jfree.data.xy.XYSeries;

import org.jfree.data.xy.XYSeriesCollection;

/\*\*

\* Klasa <code>DrawPanel</code> generuje wykres wraz z tytułem,opisem osi,

\* skalą na osiach oraz legendą opisującą przebiegi.

\*

\* @author Jacek Sobiecki

\* @author Piotr Szubert

\*/

public class DrawPanel {

public DrawPanel() {

}

/\*\*

\* Tworzy wykres i ustala jego parametry( np. kolor, opisy osi...).

\* @param dataset przedstawia opis funkcji w postaci (x,y)

\* @return wykres

\*/

private static JFreeChart createChart(XYDataset dataset){

JFreeChart chart = ChartFactory.createXYLineChart(

"Wykres Geogebry", "Pulsacja [rad]",

"Wartosc [1]", dataset, PlotOrientation.VERTICAL,

true, false, false);

chart.setBackgroundPaint(Color.lightGray);

XYPlot plot = (XYPlot) chart.getPlot();

plot.setBackgroundPaint(Color.white);

plot.setDomainGridlinePaint(Color.black);

plot.setRangeGridlinePaint(Color.black);

return chart;

}

/\*\*

\* Tworzy obiekt przedstawiający opis funkcji w układzie kartezjańskim.

\* @return obiekt przedstawiający opis funkcji w postaci (x,y)

\*/

private static XYDataset createDataset() {

XYSeries s1 = new XYSeries("GeoGebra");

int step =(int)Data.kat;

Data.kat=0;

Data.min=Data.calculate();

Data.max=Data.calculate();

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

Data.kat+=0.01745329;

double x = Data.kat;

double y = Data.calculate();

s1.add(x, y);

double y1=Data.calculate();

double y2=Data.calculate();

Data.max=Math.max(y1,Data.max);

Data.min=Math.min(y2,Data.min);

}

XYSeriesCollection dataset = new XYSeriesCollection();

dataset.addSeries(s1);

return dataset;

}

/\*\*

\* Tworzy obiekt przechowujący wykres.

\* @return obiekt przechowujący wykres

\*/

public static JPanel createWoPanel() {

JFreeChart chart = createChart(createDataset());

return new ChartPanel(chart);

}

}

**4.3 Klasa Data.java**

package ggpack;

import static java.lang.Math.cos;

import static java.lang.Math.pow;

import static java.lang.Math.sin;

/\*\*

\* Klasa <code>Data</code> przechowuje dane podawane przez użytkownika potrzebne

\* do wygenerowania wykresu oraz wykonuje na nich obliczenia.

\*

\* @author Jacek Sobiecki

\* @author Piotr Szubert

\*/

public class Data {

protected static double a;

protected static double b;

protected static double c;

protected static int i;

protected static int k;

protected static int m;

protected static double kat;

protected static double max;

protected static double min;

/\*\*

\* Oblicza funkcję geogebry.

\* @return opis matematyczny funkcji

\*/

protected static double calculate(){

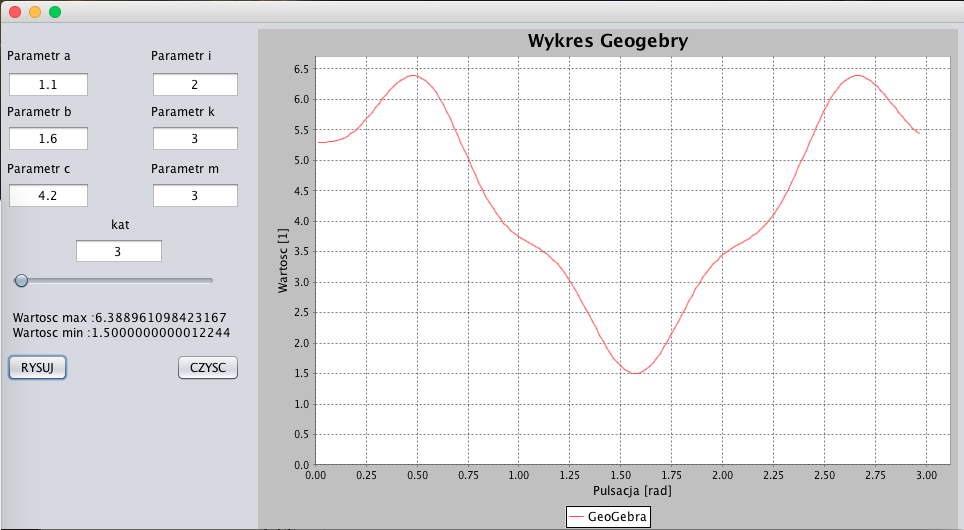
double r = c +( a \* cos(i\*kat)) + (b \* pow(sin(k\*kat),m));

return r;

}

}

**5. Testy**

Przykładowy zrzut ekranu zbudowanego oprogramowania, który realizuje test scenariusza 2.

**6. Dokumentacja użytkownika**

W załączniku