Муниципальное бюджетное общеобразовательное учреждение Злынковская средняя общеобразовательная школа №1

Проектная работа

Математический помощник

Выполнил

Марухленко Иван, 11 класс

Руководитель

Степуро А. А., учитель физики и информатики

Злынка

2022 год

Содержание

Введение

На уроках физики часто приходится работать с формулами: выводить значения отдельный переменных, определять зависимость одних от других, на основе этого строить графики и т.д. Это может занять довольно много времени. Один из вариантов решения проблемы – возложить эту работу на ЭВМ. Здесь и возникают основные трудности: большинство таких программ функционирует в онлайн-режиме, и при проблемах с подключением к сети они станут недоступными; программы же, устанавливаемые непосредственно на компьютер, выполняют только одну из необходимых нам функций.

Построить график или решить уравнение, конечно, можно самому. Но никто из нас не застрахован от ошибки. При этом (если говорить о графиках) необходимо визуализировать данные, чтобы легче их воспринимать при решении задач. У компьютера не бывает случайных ошибок, он всегда работает точно, поэтому эту работу лучше поручить ему, а чтобы легко дать ему понять, что нужно пользователю, необходимо разработать качественное ПО, специализируемое на данной проблеме.

Идея проекта – создать компьютерную программу-помощник для школьников, объединяющую в себе три функции:

1. Решение уравнений и систем уравнений

2. Построение графиков функций

3. Упрощение математических выражений

Перед началом стоит обсудить используемые мной ресурсы:

Материальные ресурсы: ПК с установленной ОС Windows; нематериальные ресурсы: ПО для ОС Windows, язык программирования (ЯП) «Python» и дополнительные библиотеки, знание ЯП, документации к языку и библиотекам, интернет-форум «StackOverflow».

Используемое мной в данном проекте программное обеспечение распространяется свободно и бесплатно. Компьютер с операционной системой

использовался долгое время (несколько лет) до начала реализации проекта. Знания о ЯП были получены из курсов на платформе Stepik (https://stepik.org/course/58852).

Риски минимальны, т.к. решение большинства проблем, связанных с написанием программы, описаны в статьях на форуме «StackOverflow». Из трудностей стоит отметить лишь возможную сложность выбранных библиотек, из-за чего может понадобиться сменить их на другие, и недостаточные знания языка программирования.

Цель проекта.

Создание программы-помощника, способной решать уравнения и строить графики функций, а также производить упрощение выражений

Задачи:

1. Выбрать язык программирования и среду разработки для написания кода программы

2. Изучить дополнительные библиотеки, необходимые для создания графического интерфейса программы

3. Составить консольную версию приложения с базовым функционалом

4. Добавить графическую оболочку к коду

5. Исправить недочеты приложения

6. Улучшить производительность программы и читабельность кода

7. Испытать готовое приложение на разных устройствах

Подготовка к проекту

В начале реализации проекта мне было необходимо выбрать, с помощью каких средств я буду создавать своё приложение. Уже долгое время я учу язык программирования Python. Он достаточно прост в понимании и написании кода. Мой выбор пал на него. Теперь нужно разобраться со средой разработки. Необходимый минимум – подсветка синтаксиса ЯП. В ряд полезных функций среды разработки можно также отнести запуск кода напрямую, возможность отладки или быстрая

установка доп. библиотек. В моём случае идеальное решение - PyCharm - интегрированная среда разработки для языка программирования Python, разработанная компанией JetBrains. Она предоставляет средства для анализа кода, графический отладчик, инструмент для запуска юнит-тестов. У программы есть бесплатная версия, функционала которой достаточно для написания моего «Помощника».

Далее необходимо придумать, как реализовать свою идею. Python имеет множество дополнительных библиотек, с помощью которых можно сэкономить много времени, не создавая решения с нуля.

Во-первых, необходимо сделать возможным решение разновидовых уравнений одной программой. В школе на уроках информатике нам приходилось составлять программы для решения квадратных и линейных уравнений. Но они занимали по 15 – 30 строк кода, а ввод некорректных данных не предполагался. К тому же, если результат являлся иррациональным числом, происходило его округление, что порождало неточности. Решение: использование символьных значений. Так корни будут оставаться корнями, дроби – дробями, а также станет возможным объявление символьных переменных (например, x, y, z), так что теперь программа приобретёт возможность принимать на вход уравнения, а не отдельные значения коэффициентов. Я смог найти единственную библиотеку, способную на такое – это SymPy. Помимо вышеперечисленных функций, она обладает возможностью решения уравнений, построения графиков, упрощения выражений и др.

Попробовав построить график с помощью SymPy, я обнаружил, что он состоит из точек, соединённых отрезками. Количество этих точек изменить было нельзя, поэтому графики почти во всех случаях получались в виде ломаных. В поисках решения проблемы, я наткнулся на библиотеку Matplotlib, специализированную на построении графиков и диаграмм, которые строятся по схожему с SymPy принципу – из массива точек, но в отличие от SymPy их количество регулируемо. Для их создания достаточно правильно создать этот массив и передать его функции для построения.

И одна из самых главных частей – это добавление программе графического интерфейса (GUI). До проекта я знал только одну библиотеку, способную его

реализовать, - Kivy, и я решил использовать её, но возникло две проблемы. Первая из них – документация библиотеки была слишком ёмкой, и в ней было трудно разобраться. Вторая же проблема, более серьёзная – Matplotlib для построения графика создавал отдельное окно, и я не смог найти, как объединить его с окном Kivy. Теперь мне пришлось искать библиотеку для создания GUI по этим критериям. Ей оказалась стандартная библиотека Tkinter. Не найдя альтернативы, я стал использовать её.

Создание программы

Разобравшись с библиотеками, я приступил к написанию кода. Было решено не составлять консольную версию приложения, чтобы потом не тратить время на изменение его под графическую оболочку, и я сразу приступил к созданию графики. По моей задумке программа должна была быть представлена в виде окна, занимающего четверть экрана (чтобы его можно было расположить над окнами других приложений, например, вкладки с формулами в браузере), основной интерфейс расположен на нескольких вкладках, которые можно либо закрыть, либо добавить несколько одинаковых, сверху – меню для их добавления. Я начал писать код. Для создания вкладок был объявлен отдельный класс (TabMaker), где каждый метод отвечал за создание своего вида вкладок.

Первая вкладка отвечала за решение уравнений. Её внешний вид был достаточно простой: одно поле для ввода данных, кнопка для отправки этих данных функции-решателю и Label для вывода корней. Изначально в решателе был сделан цикл, отвечающий за «ловлю русских символов», и, если они обнаруживались, вместо корней выводилось сообщение «Недопустимы русские символы»; но в ходе ввода случайных значений (в их числе была переменная, написанная с заглавной буквы кириллицы) я обнаружил, что это не приводит к ошибке в программе, а переменные могут представлять из себя хоть целые слова. После такой находки я удалил этот цикл, и функция решателя стала меньше. Устранив ещё несколько неточностей, я добавил возможность решения систем уравнений: на вкладку было добавлено ещё две кнопки – добавить и удалить уравнения, образующие систему. Я решил ограничить количество уравнений в системе – их максимум составил 3 уравнения (системы

уравнений, которые проходят в школе, состоят только из 2 уравнений, чтобы был некий «запас», я сделал именно такой максимум). Написав необходимый код, я задумался о так называемых исключениях – ошибках, которые могут возникать, если пользователь отправит на ввод неправильное уравнение или пустое поле. Чтобы программа не зависала от таких «сюрпризов», была реализована ловля этих исключений. Теперь, если ввод вызывал ошибку, программа выводила сообщение «Некорректный ввод». Некоторые из таких порождали исключение о невозможности решения вследствие отсутствия функции, обратной данной (например, факториал). Я не хотел, чтобы они воспринимались как обычные ошибки, поэтому до кода с поимкой всех исключений, я написал ещё несколько строк, ответственных за ловлю именно этого исключения, и в случае его поимки выводилось сообщение «Не могу решить».

Первый вид вкладок сделан, я приступил ко второму – построение графиков функций. Всё доступное пространство окна приложения было разделено на две части: на левой производился ввод функции, а также диапазон X (что обязательно для matplotlib, т.к. массив точек создаётся в ограниченном промежутке, а не генерируется в зависимости от положения графика), на правой – холст tkinter с полем для построения графиков. Вычисление вводимых значений производилось уже не с помощью SymPy, а посредством библиотеки, являющейся некоторым дополнением matplotlib – numpy, благодаря ей же создаются массивы точек. С этой вкладкой возникло уже гораздо больше проблем. Во-первых, проявилась проблема с построением графика функции обратной пропорциональности (y = k/x), которая состояла из двух частей. Matplotlib реализован так, что точки из массивов просто соединяются отрезками, и вместо уходящих в бесконечность гипербол отображалась фигура, состоящая из соединённых косой прямой веток этих гипербол. Решение проблемы оказалось достаточно простым: количество точек должно было быть нечётным, благодаря чему в массиве значений x появлялся 0, и здесь y заменялся на infinity (бесконечность). Другая проблема носила схожий характер – график тангенсоиды. Здесь график уходил в бесконечность на значениях, которые представлены числами, связанными с pi, поэтому как бы не составлялся массив – он никогда не будет включать в себя иррациональные числа. Решение – ограничить

график по y, а чтобы не было соединительных линий, как в графике обратной пропорциональности, нужно большие значения y заменять на бесконечность, и теперь график стал строиться правильно. Ещё одна небольшая проблема: иногда значения y в несколько раз превосходили x, и график, например, функции y = 1/x^2 выглядел как две перпендикулярные прямые. Решение – уменьшить кол-во точек в массивах (с 6001 до 2001). Из-за этого уменьшается точность графика, но теперь график выглядит, как нужно. После реализации основной части вкладки, я, как и в решателе, сделал ловлю исключений для защиты от некорректного ввода.

Остался последний вид вкладок, предназначенный для упрощения выражений. Это была самая простая вкладка из всех, и в её работе не было никаких проблем.

Я реализовал все свои задумки по вкладкам. Меню с возможностью добавления вкладок было решено не создавать по одной причине – в Tkinter для вкладок не предусмотрена возможность их закрытия, а альтернативное решение для меня было непонятно и слишком громоздко. Поэтому при старте программы создаётся только 3 вкладки каждого вида.

Позже я объединил вкладку решателя уравнений и вкладку для упрощения выражений, т.к. на них оставалось слишком много свободного места, а уменьшать окно было нельзя – это повлияло бы и на вкладку для построения графиков. И в конце была написана небольшая инструкция с условными обозначениями математических функций и некоторых констант. На этом основная часть работы была окончена.

Тестирование готового продукта

Компиляция кода

Написание кода программы завершено. Теперь необходимо позаботиться о её дальнейшем использовании: необходимо обеспечить работоспособность «Математического помощника» на любом компьютере. Сейчас для запуска кода необходим интерпретатор языка Python, что создаёт дополнительные сложности с установкой программы. Необходимо поместить код в исполняемый файл, способный работать без дополнительного ПО. Проще говоря, из файла .py нужно создать файл .exe. Самым популярным способом реализации этого служит утилита pyinstaller.

Готовый файл может запуститься на устройстве, даже если на нём не будет python-интерпретатора. Но такой вариант компиляции имеет несколько минусов: 1) Подобный файл занимает достаточно много для своего функционала памяти компьютера; 2) Скорость выполнения файла очень низкая, что связано с самим Пайтоном. На этом этапе я стал понимать, что мой выбор ЯП был не лучшим – подобные программы лучше писать на компилируемых языках программирования. Но начинать всю работу заново было бы нерентабельно. Поэтому я стал искать, как решить мою заминку.

Был найден следующий способ: посредством более быстрого интерпретатора Пайтона – PyPy – и ряда дополнительных программ преобразовывался в машинный и упаковывался в .exe файл, занимающий в разы меньше места и заметно скорее выполняющий команды пользователя. Но я так и не смог осуществить это: после ряда неудачных попыток установки необходимых пакетов, смены устройства и ошибок компиляции я обнаружил критический недостаток этого способа: код, посредством которого должна была производиться упаковка в .exe файл, был написан на python версии 2.7 и мог компилировать только программы, написанные на той же версии. Я не знал всех различий python2 и python3, поэтому этот единственный метод, который мог бы решить проблему, стал бесполезным. Оставалось только установить на все компьютеры в классе информатики необходимое ПО для запуска программы. Это был самый органичный вариант из всех оставшихся. После того, как я удостоверился, что на всех устройствах всё работает нормально, проект можно было считать завершённым.

Заключение

Итак, мне удалось полностью реализовать свою задумку. Подведу итог своих результатов:

- программа способна решить уравнения и строить графики, необходимые для уровня 11 класса;

- ей легко можно управлять интуитивно, имеются подсказки для написания выражений и защита от некорректного ввода;

- файл приложения не занимает много места и достаточно быстро выполняет нужные операции, вполне подходит для компьютеров в моей школе;

Также стоит упомянуть недостатки некоторые недостатки проекта:

1) В уравнениях вида 0*x = 0 программа не находила корни
2) Корни уравнений с тригонометрическими функциями находились в промежутке от 0 до 2*pi, т.е. отсутствует периодичность корней
3) Уравнения, в которых присутствовали функции, у которых нет обратной им функции, не могли быть решены
4) При построении графиков некоторых функций с тангенсами график обрывался
5) Некоторые графики функций строились некорректно

Но в целом, я доволен своей работой. С тех пор, как я увлёкся программированием, это был мой самый крупный проект, который я реализовал. Надеюсь, что он станет полезным

Список источников использованной информации

Электронные ресурсы:

1. Stack Overflow на русском. Сайт вопросов и ответов для программистов. Джефф Атвуд, Джоэл Спольски. [Электронный ресурс] URL: https://ru.stackoverflow.com

2. SymPy 1.6.2 documentation. Документация библиотеки Python SymPy. SymPy Development Team. [Электронный ресурс] URL: https://docs.sympy.org

3. Tkinter GUI Python. Плейлист с видеоуроками по программированию графического интерфейса на Python. Канал «Источник Знаний». [Электронный ресурс] URL: https://www.youtube.com/playlist?list=PLjRuaCofWO0MVr1xkRXiHR3OAZ4M0C2n

Приложения

Ссылка для скачивания:

https://drive.google.com/drive/folders/1gWW00zBiDeo11d_tHpWOzLfKTO49PfJx?usp=sharing

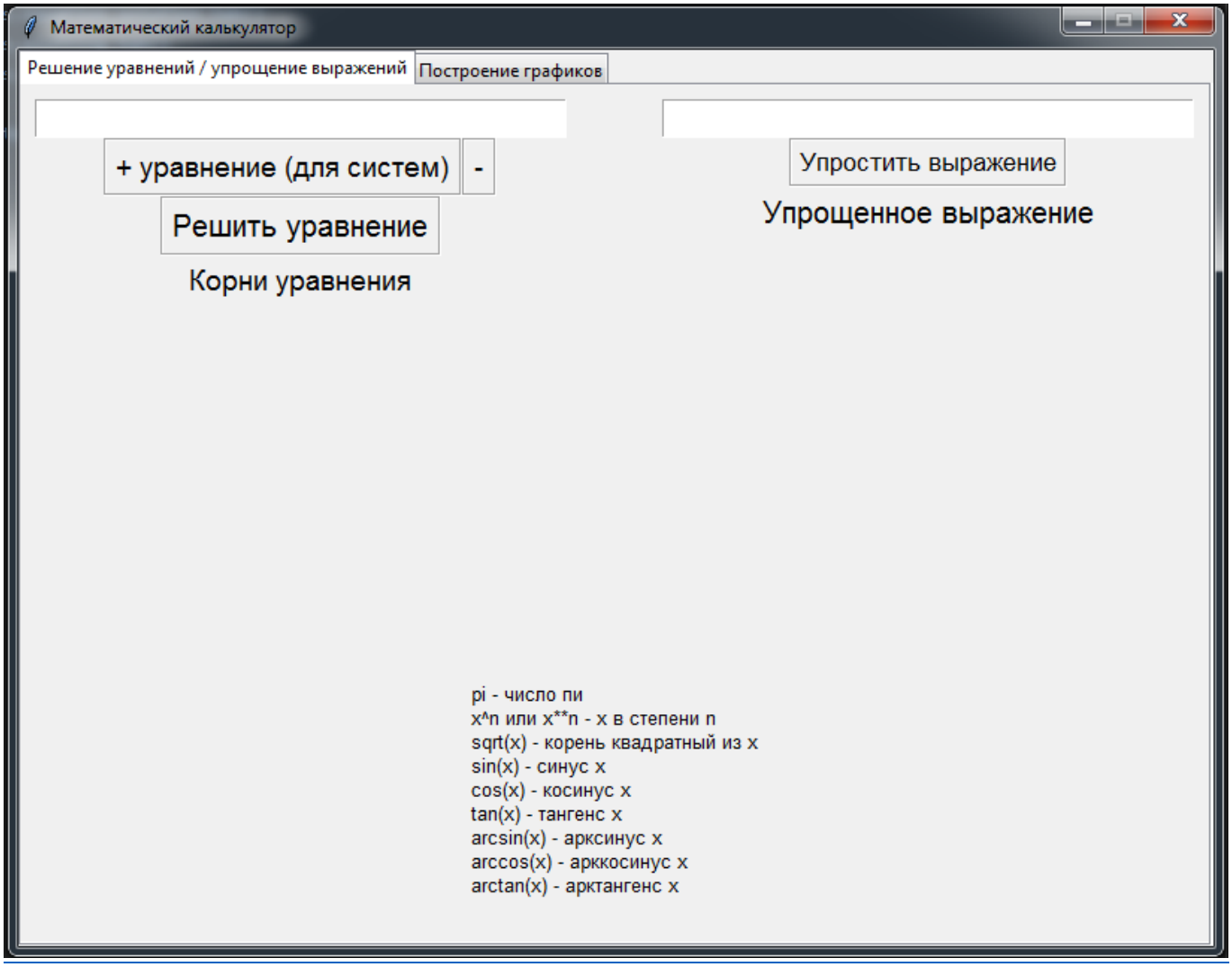

Рисунок 1. Первая вкладка

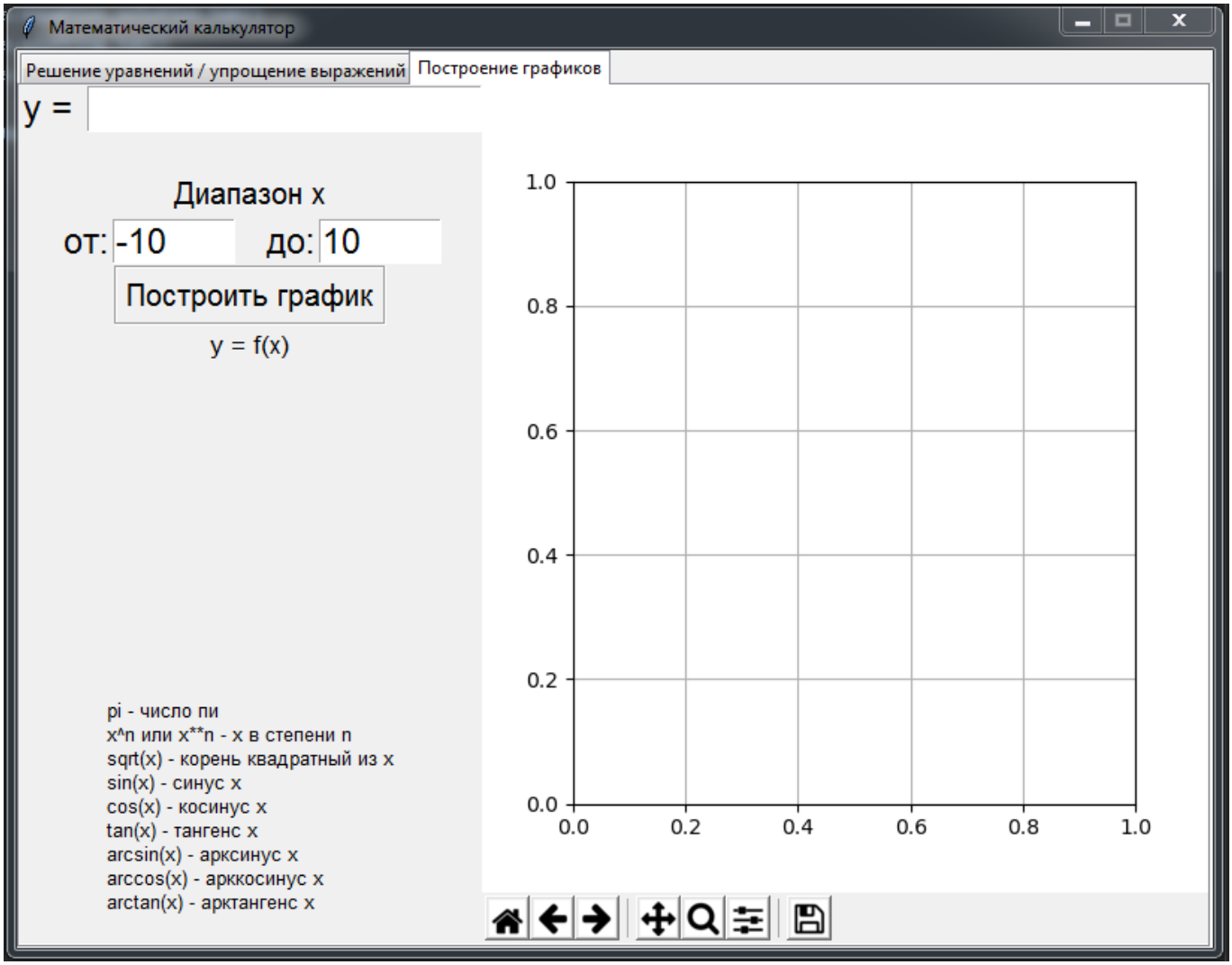

Рисунок 2. Вторая вкладка

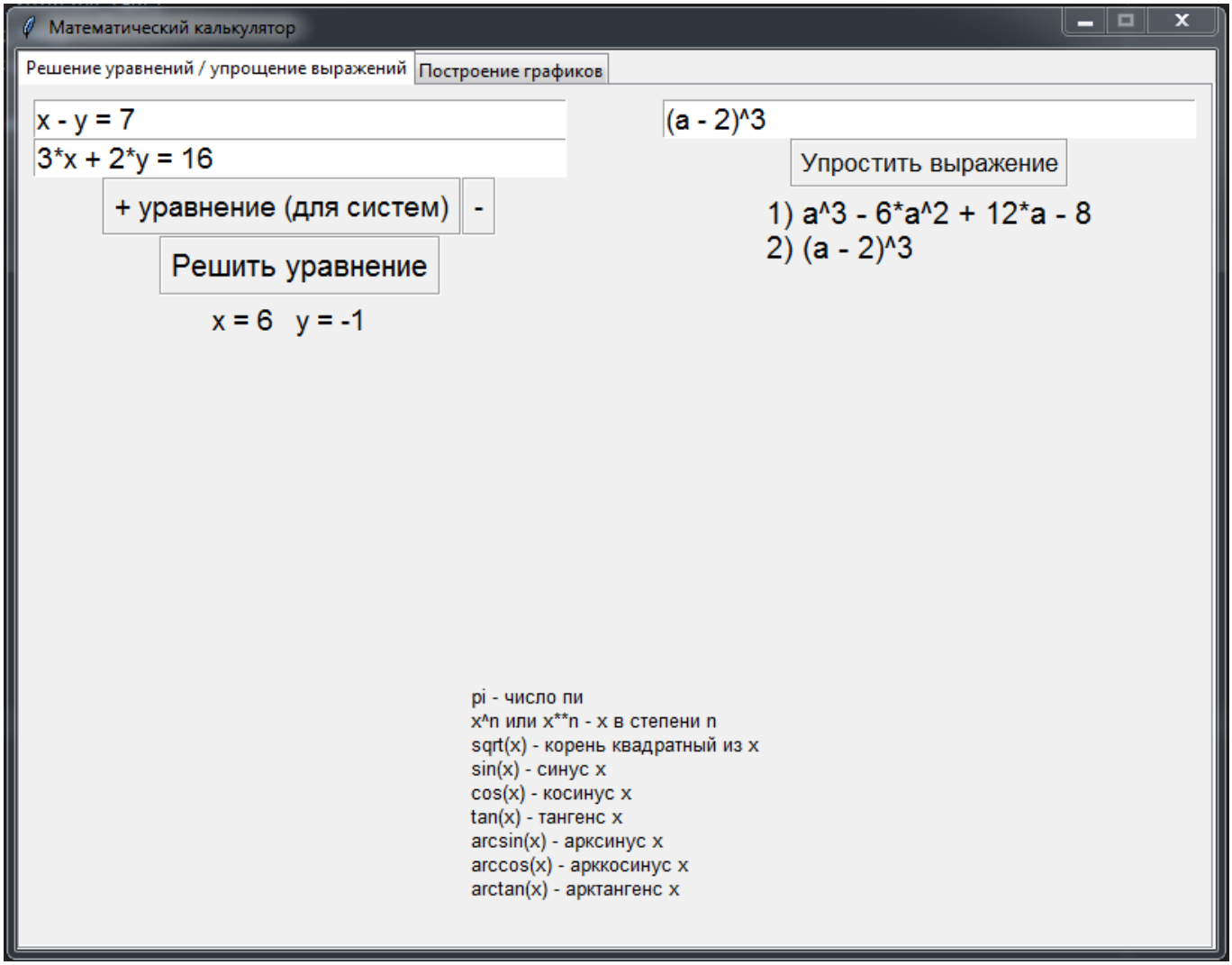

Рисунок 3. Решение системы уравнений и упрощение выражения

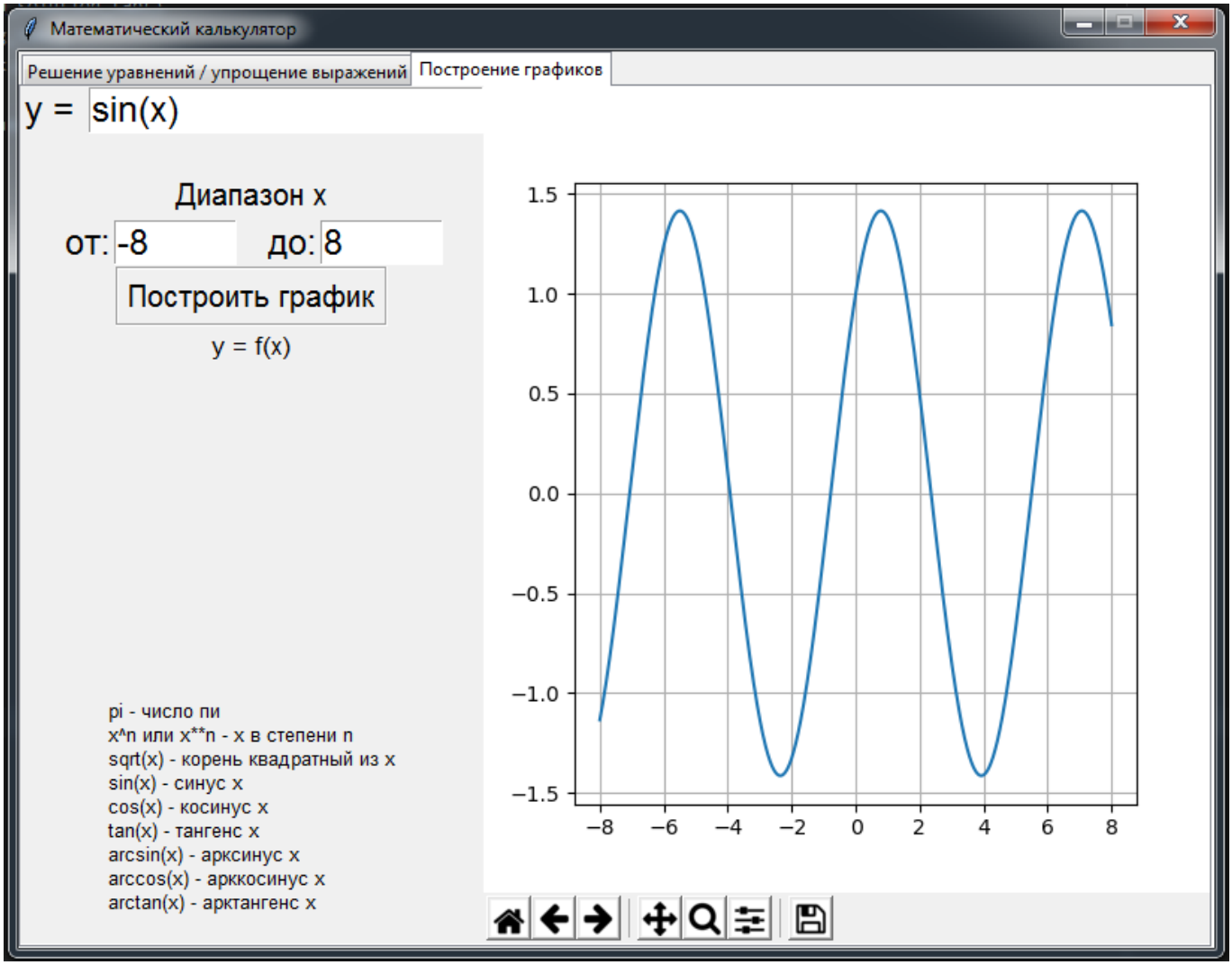

Рисунок 4. Построение синусоиды

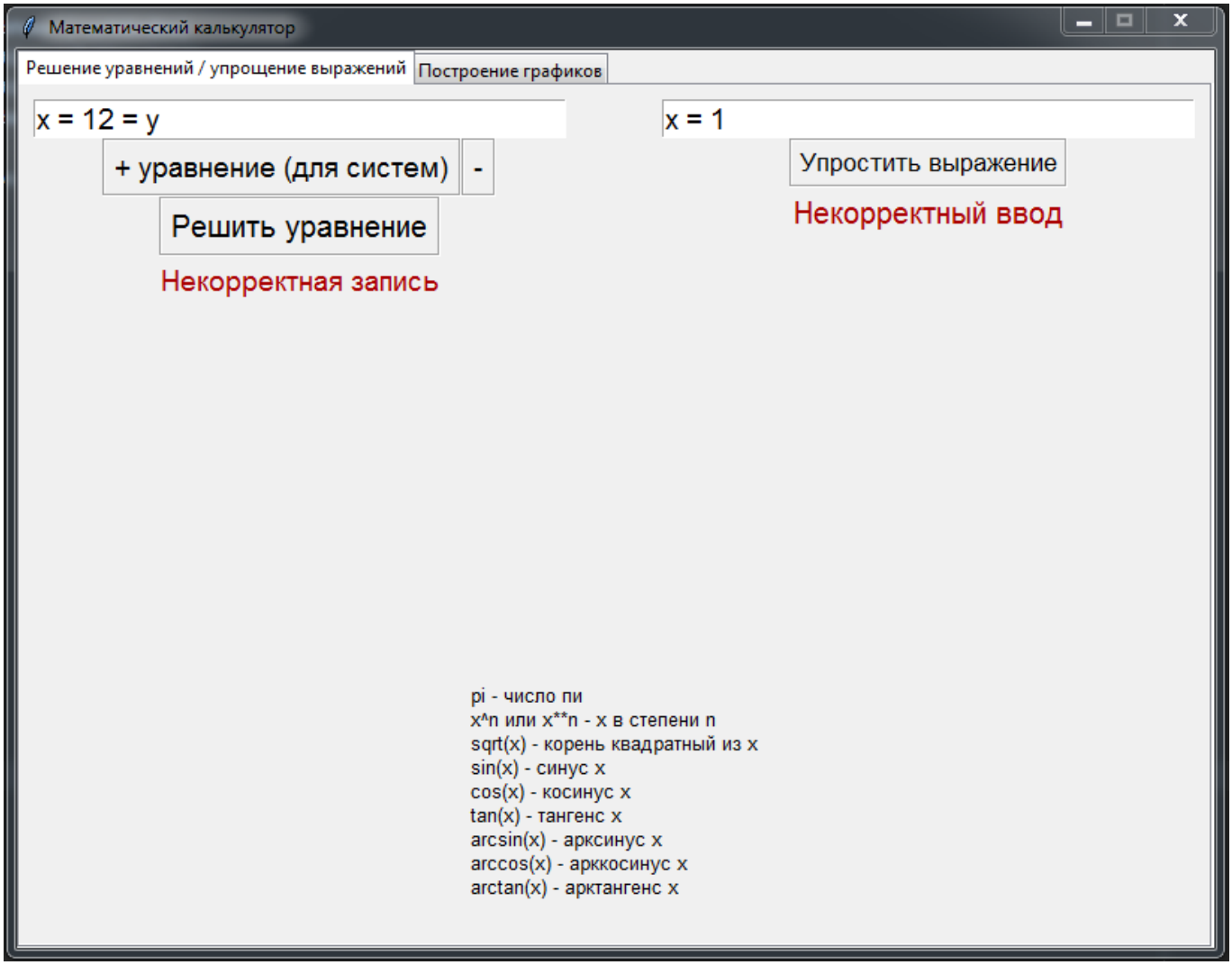

Рисунок 5. Вывод ошибки о некорректном вводе

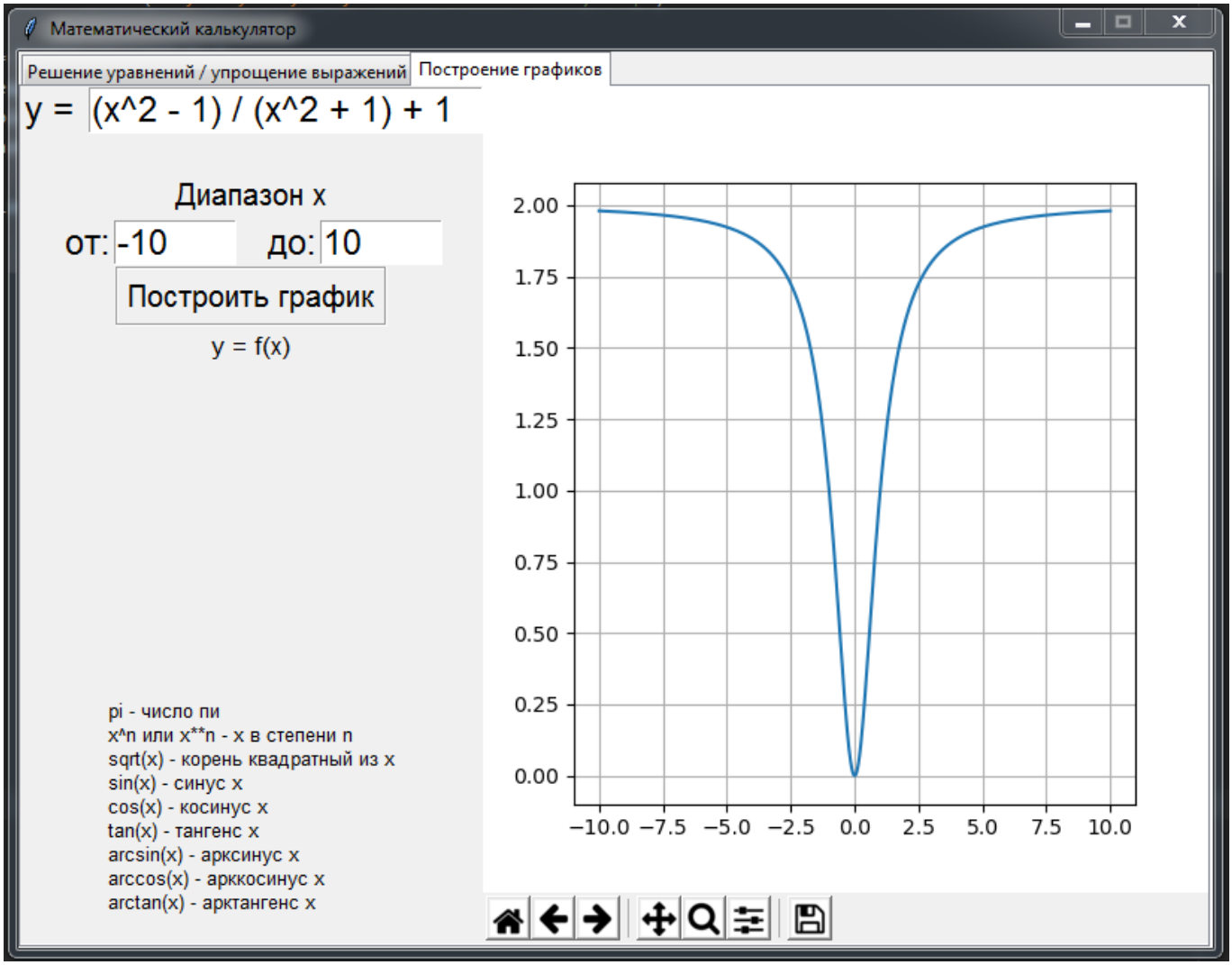

Рисунок 6. Пример построения графика

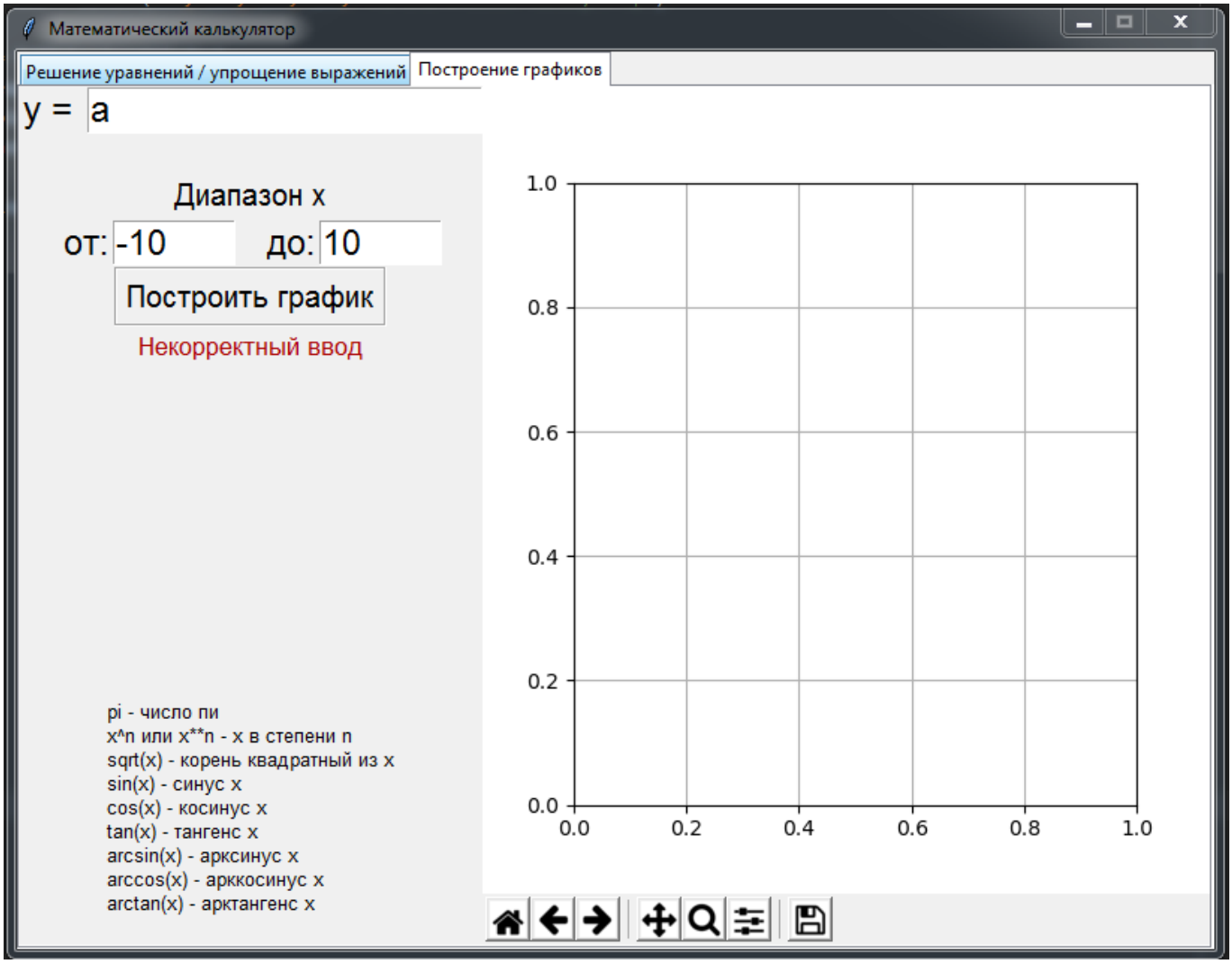

Рисунок 7. Вывод ошибки при некорректном вводе на вкладке построения графиков

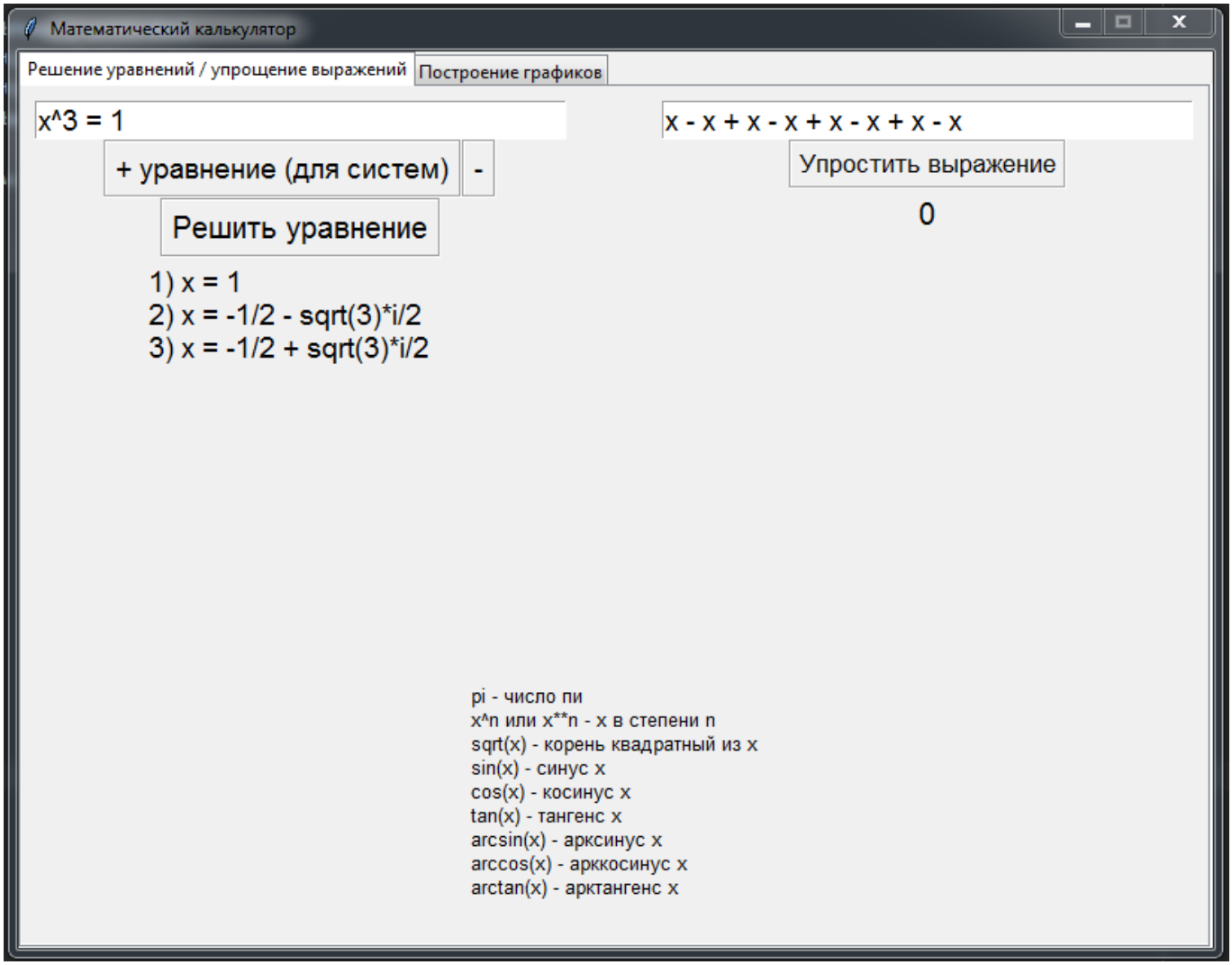

Рисунок 8. Решение уравнения в комплексных числах

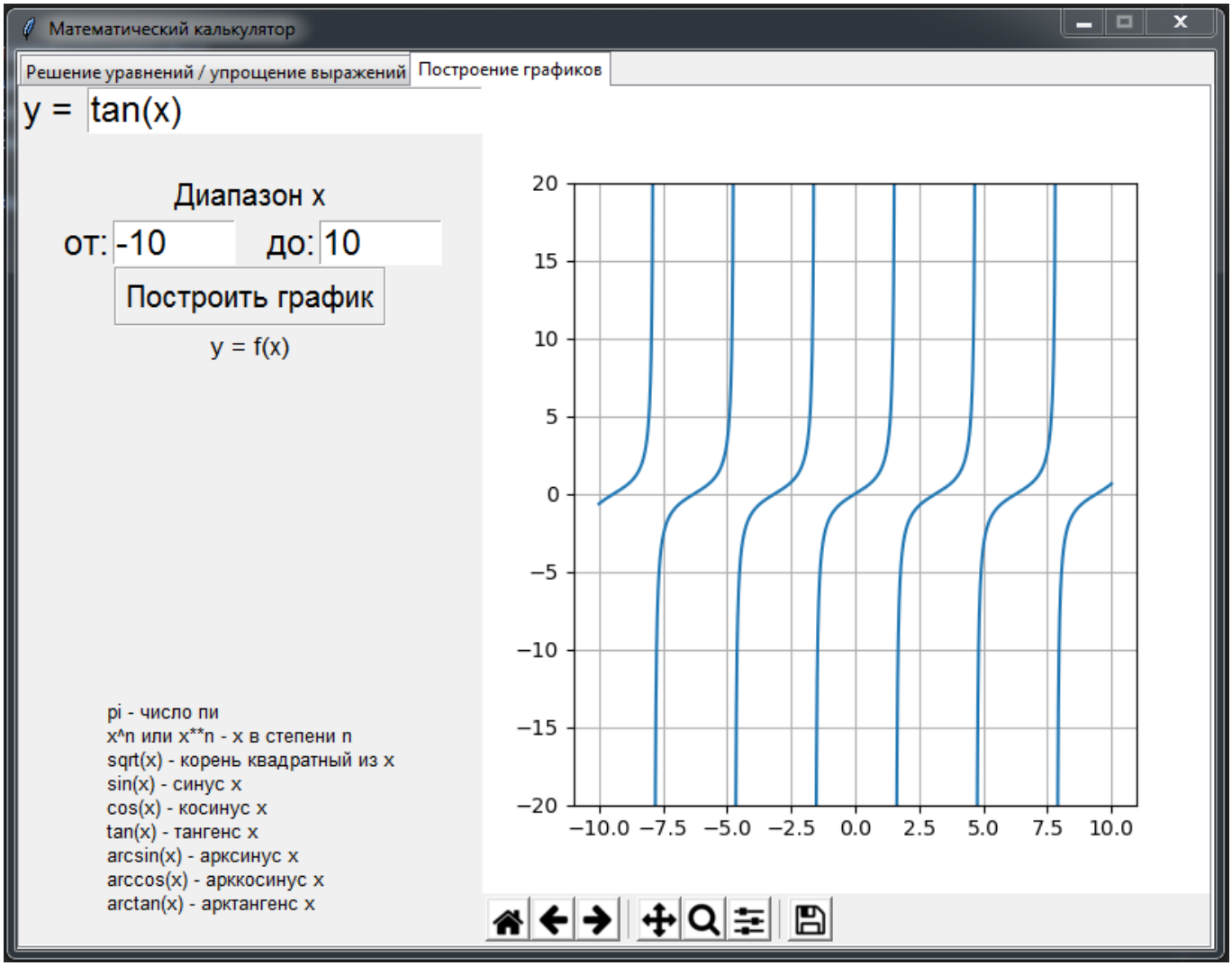

Рисунок 9. Построение тангенсоиды

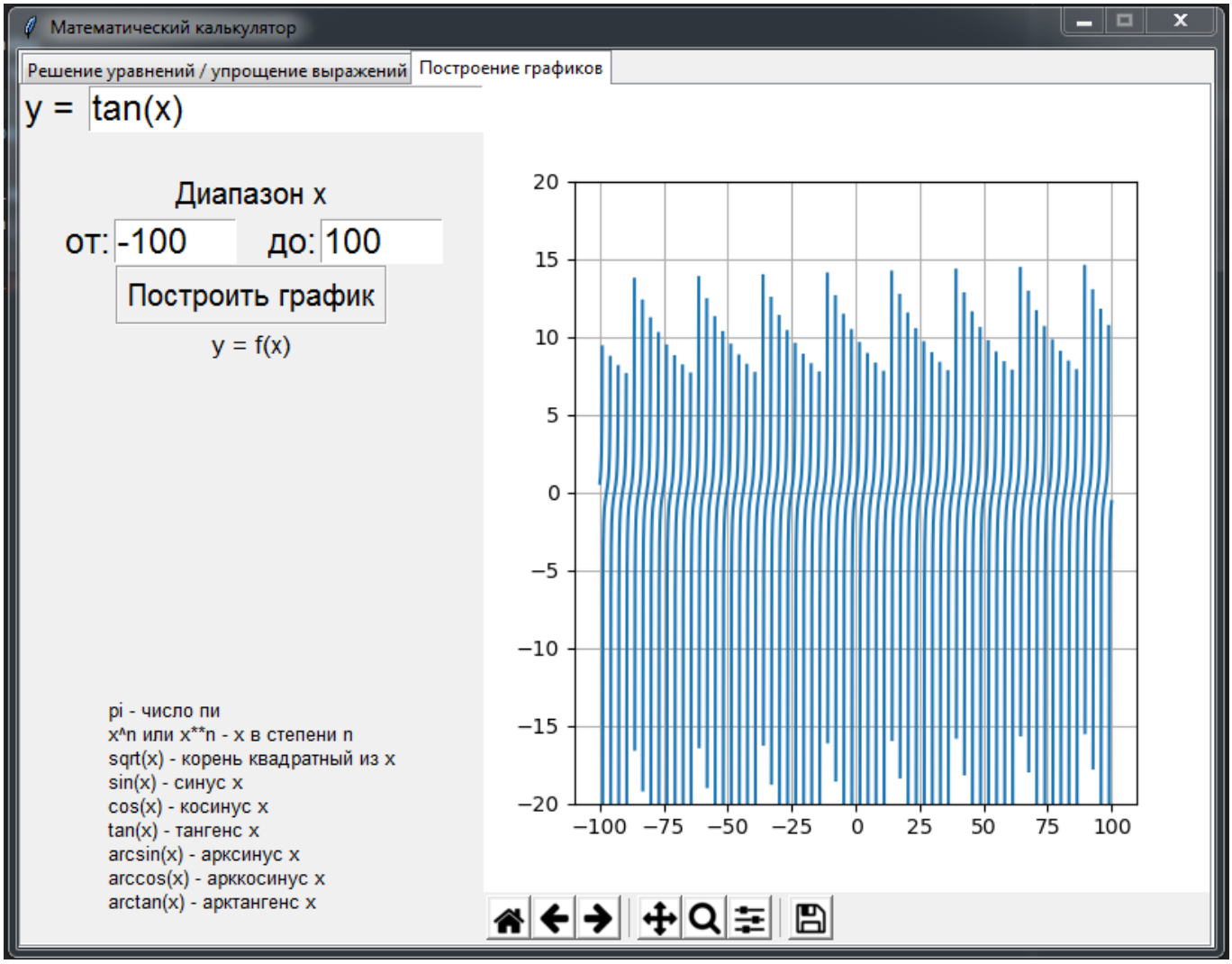

Рисунок 10. Ошибка в построении тангенсоиды при большом диапазоне x

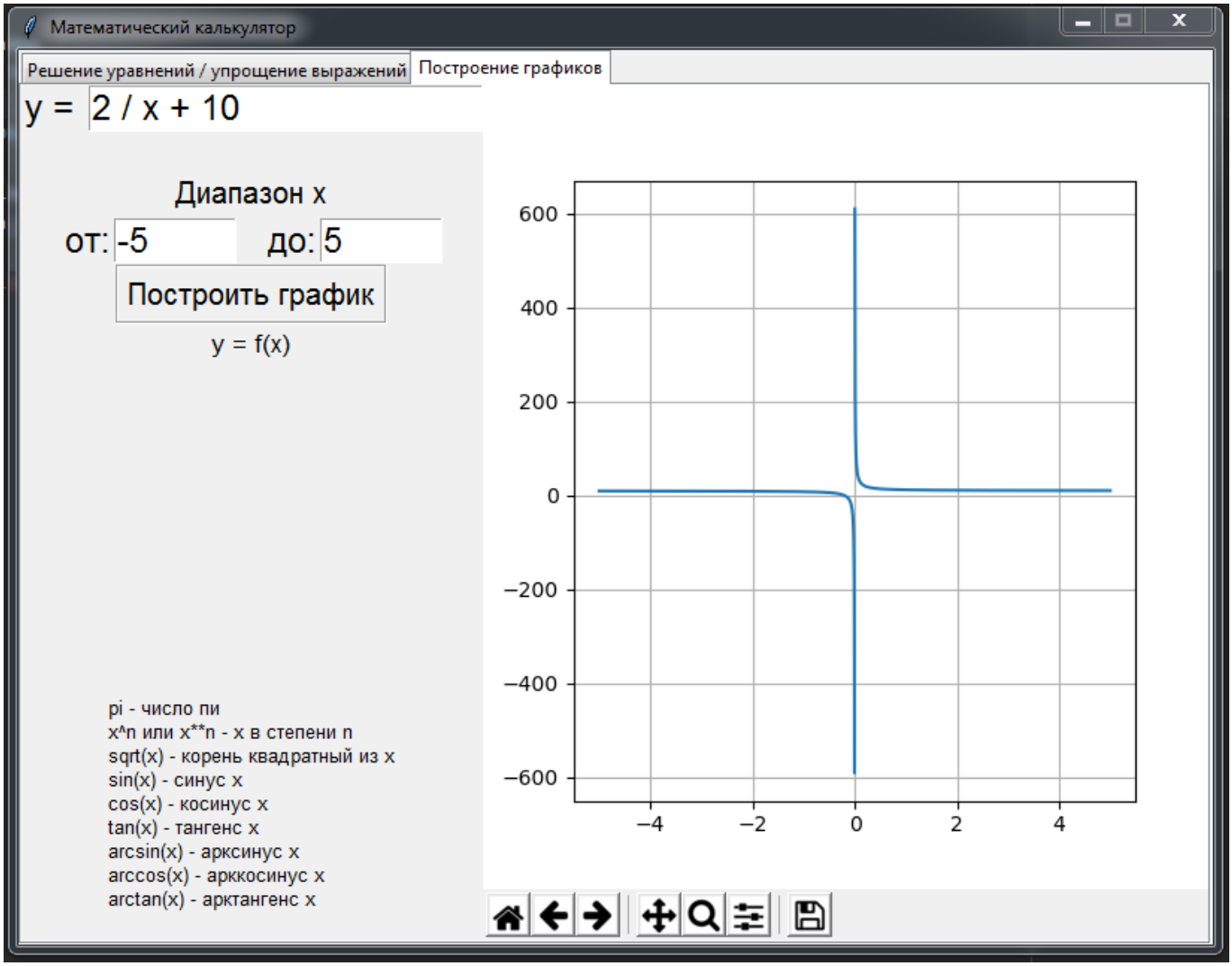

Рисунок 11. Построение графика функции с точкой разрыва

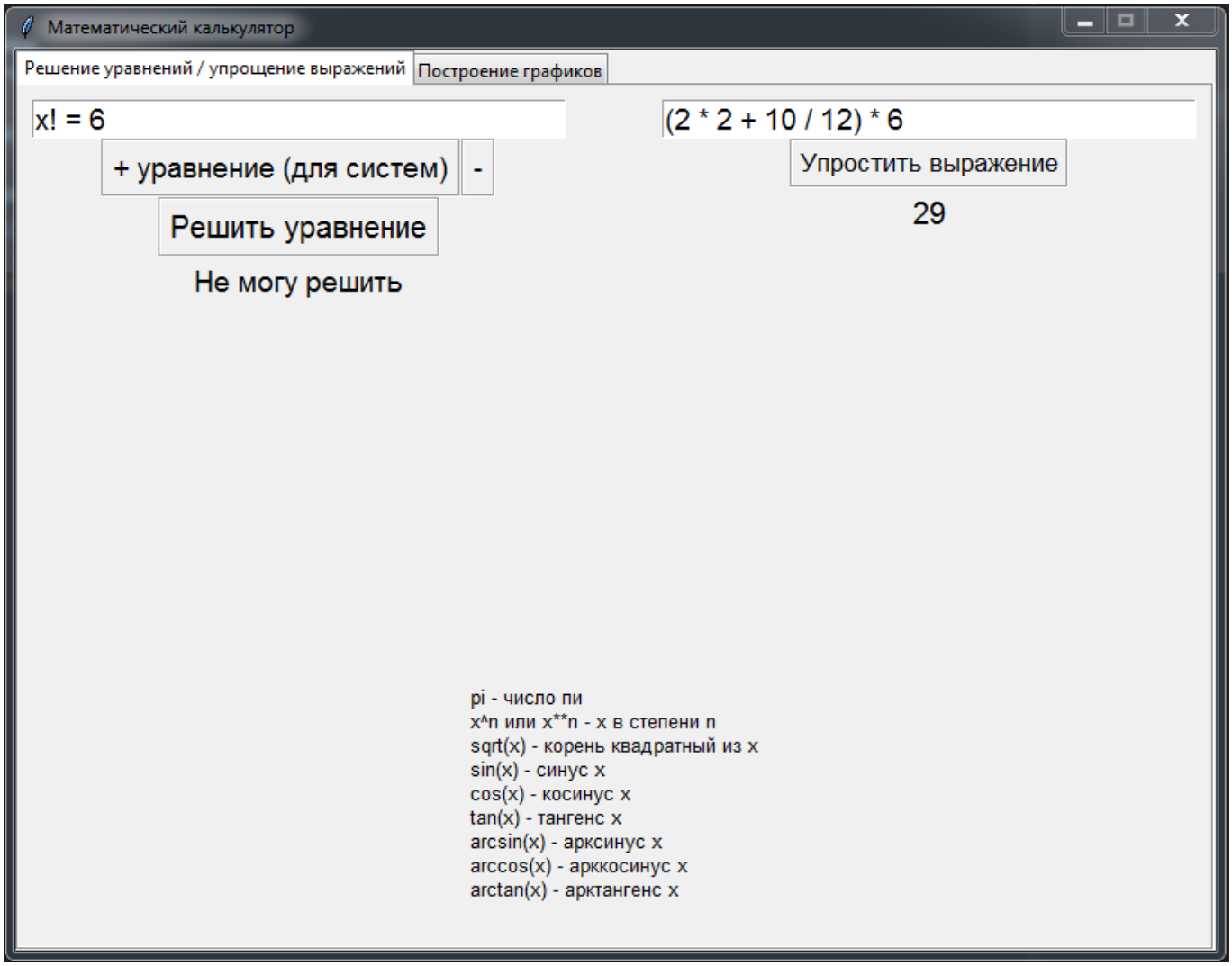

Рисунок 12. Вывод сообщения в случае неспособности решить уравнение. Использование вкладки с упрощением выражения как калькулятора