Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Пермский национальный исследовательский

политехнический университет»

(ПНИПУ)

Кафедра вычислительной математики и механики

Лабораторная работа №1

по дисциплине: «Учебно-исследовательская работа»

по теме: «GUI»

Вариант 1

Выполнил:

Студент группы ИСТ-19-1бзу

Соболь Е.В.

Проверил:

ст.пр кафедры ВМиМ

Сахабутдинова Л. Р.

Пермь

1. г.

Оглавление

[1 Постановка задачи 3](#_Toc72241212)

[2 Математическая составляющая 4](#_Toc72241213)

[2.1 Функция eval 4](#_Toc72241214)

[2.2 Первый аргумент: expression 4](#_Toc72241215)

[2.3 Второй аргумент: globals 6](#_Toc72241216)

[2.4 Третий аргумент: locals 7](#_Toc72241217)

[3 Описание интерфейсной части 9](#_Toc72241218)

[4 Программная часть 10](#_Toc72241219)

[5 Анализ аналогичных приложений 12](#_Toc72241220)

[6 Информационные технологии для реализации 13](#_Toc72241221)

[7 Логическая схема работы приложения 15](#_Toc72241222)

[Приложение 17](#_Toc72241223)

# 1 Постановка задачи

Разработать приложение для построения графиков математических функций. Выбрать библиотеку для интерфейса. График должен строиться с помощью библиотеки matplotlib.

Требования к приложению:

* Наличие блока с арифметическими действиями.
* Наличие блока с тригонометрическими функциями.
* Ввод начала и конца интервала, шага для построения функции.
* Возможность построения нескольких графиков на одном полотне.
* Возможность настройки шага сетки по вертикали и горизонтали.
* Возможность выбора декартовой или полярной системы координат.

# 2 Математическая составляющая

## 2.1 Функция eval

Приложение должно обеспечить формирование графика. Для этого необходимо использовать специальную функцию eval, которая позволяет преобразовать введенное выражение в результат вычислений. Предполагается график только от одной функции с переменной по наименованию x. Давайте рассмотрим что из себя представляет функция eval.

Используется встроенная функция eval() для динамического исполнения выражений из ввода на основе строки или скомпилированного кода. Если вы передаете в eval() строку, то функция анализирует ее, компилирует в байт-код и выполняет как выражение Python.

Сигнатура eval() определена следующим образом:

eval(expression[, globals[, locals]])

Первый аргумент expression содержит выражение, которое необходимо выполнить. Функция также принимает два необязательных аргумента globals и locals.

## 2.2 Первый аргумент: expression

Когда мы вызываем eval(), содержание expression воспринимается интерпретатором как выражение Python.

>>> eval("2 \*\* 8")

256

>>> eval("1024 + 1024")

2048

>>> eval("sum([8, 16, 32])")

56

>>> x = 100

>>> eval("x \* 2")

200

При вызове eval() со строковым выражением в качестве аргумента, функция возвращает значение, полученное в результате оценки входной строки. По умолчанию eval() имеет доступ к глобальным именам.

Чтобы оценить строковое выражение, eval() выполняет следующую последовательность действий:

1. Парсинг выражения.
2. Компилирование в байт-код.
3. Выполнение кода выражения Python.
4. Возвращение результата.

Имя аргумента expression подчеркивает, что функция работает только с выражениями, но не составными конструкциями. При попытке передачи блока кода вместо выражения будет получено исключение SyntaxError:

>>> x = 100

>>> eval("if x: print(x)")

SyntaxError: invalid syntax

Таким образом, в eval() нельзя передать конструкции c if, import, def или class, с циклами for и while. Однако ключевое слово for может использоваться в eval() в случае выражений для генераторов списков.

В eval() запрещены и операции присваивания:

>>> eval("pi = 3.1416")

SyntaxError: invalid syntax

SyntaxError также вызывается в случаях, когда eval() не удается распарсить выражение из-за ошибки в записи:

>>> eval("5 + 7 \*")

SyntaxError: unexpected EOF while parsing

В eval() можно передавать объекты кода (code objects). Чтобы скомпилировать код, который вы собираетесь передать eval(), можно использовать compile(). Это встроенная функция, которая может компилировать строку в объект кода или AST-объект.

## 2.3 Второй аргумент: globals

Аргумент globals опционален. Он содержит словарь, обеспечивающий доступ eval() к глобальному пространству имен. С помощью глобальных переменных можно указать eval(), какие глобальные имена использовать при выполнении выражения.

Глобальные имена – это все те имена, которые доступны в текущей глобальной области или пространстве имен. Вы можете получить к ним доступ из любого места в вашем коде.

Все имена, переданные глобальным переменным в словаре, будут доступны eval() во время выполнения.

>>> x = 100 # Глобальная переменная

>>> eval("x + 100", {"x": x})

200

>>> y = 200 # Другая глобальная переменная

>>> eval("x + y", {"x": x})

NameError: name 'y' is not defined

Любые глобальные имена, определенные вне пользовательского словаря globals, не будут доступны изнутри eval(), будет вызвано исключение NameError.

Вы также можете указать имена, которых нет в текущей глобальной области видимости. Чтобы это работало, нужно указать конкретное значение для каждого имени. Тогда eval() будет интерпретировать эти имена, как если бы это были глобальные переменные:

>>> eval("x + y + z", {"x": x, "y": y, "z": 300})

600

>>> z # самой переменной нет в глобальной области видимости

NameError: name 'z' is not defined

Если вы предоставите eval() пользовательский словарь, который не содержит значения для ключа "\_\_builtins\_\_", то ссылка на словарь встроенных функций всё равно будет автоматически добавлена к ключу "\_\_builtins\_\_", прежде чем выражение будет проанализировано. Это гарантирует, что eval() имеет полный доступ ко всем встроенным именам Python при оценке выражения.

>>> eval("sum([2, 2, 2])", {})

6

>>> eval("min([1, 2, 3])", {})

1

>>> eval("pow(10, 2)", {})

100

Несмотря на переданный пустой словарь ({}), eval() имеет доступ к встроенным функциям.

При вызове eval() без передачи пользовательского словаря в глобальные переменные аргумент по умолчанию будет использовать словарь, возвращаемый globals() в среде, где вызывается eval():

>>> x = 100 # Глобальная переменная

>>> y = 200 # Другая глобальная переменная

>>> eval("x + y")

300

Таким образом, передача словаря в аргументе globals служит как способ намеренно ограничить область видимость имен для функции eval().

## 2.4 Третий аргумент: locals

Аргумент locals также является необязательным аргументом. В этом случае словарь содержит переменные, которые eval() использует в качестве локальных имен при оценке выражения.

Локальными называются те имена (переменные, функции, классы и т.д.), которые мы определяем внутри данной функции. Локальные имена видны только изнутри включающей функции.

>>> eval("x + 100", {}, {"x": 100})

200

>>> eval("x + y", {}, {"x": 100})

NameError: name 'y' is not defined

Обратите внимание, что для передачи словаря locals сначала необходимо предоставить словарь для globals. Передача по ключу в случае eval() не работает:

>>> eval("x + 100", locals={"x": 100})

TypeError: eval() takes no keyword arguments

Главное практическое различие между globals и locals заключается в том, что Python автоматически вставит ключ "\_\_builtins\_\_" в globals, если этот ключ еще не существует. Cловарь locals остается неизменным во время выполнения eval().

# 3 Описание интерфейсной части

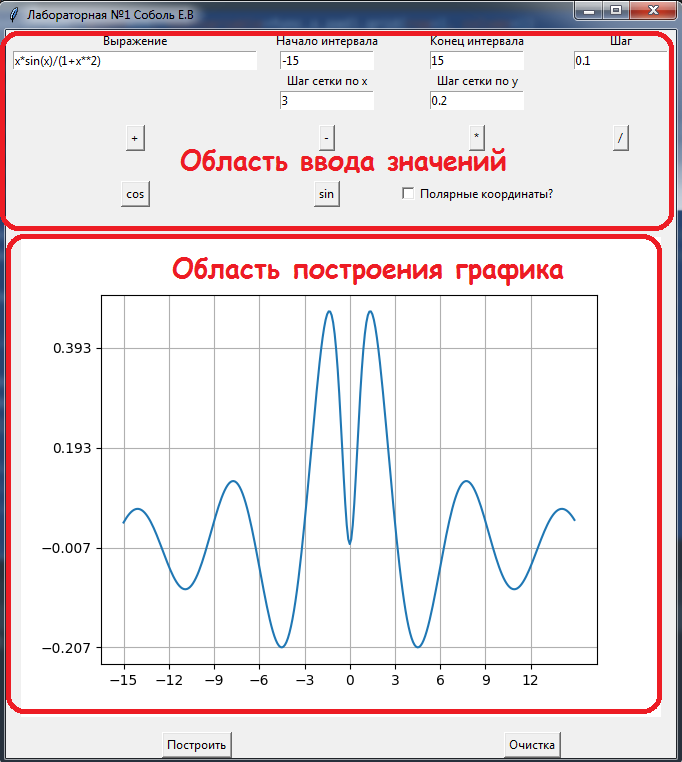


Рисунок 1 – Интерфейс

Интерфейс состоит из ряда кнопок, текстовых полей для ввода соответствующих значений и чек- бокса для работы с полярными координатами. Каждое поле подписано надписью, у каждой кнопки есть текст и возможность кликнуть по ней. По центру приложения расположен холст, в котором рисуется график.

# 4 Программная часть

Описание программной части содержит код с подробными комментариями.

import tkinter as tk # использование функций библиотеки TKinter для пользовательского интерфейса

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg # использование FigureCanvasTkAgg из библиотеки matplotlib для отрисовки графика на холсте

import matplotlib.pyplot as plt # использование plt из библиотеки matplotlib для формирования графика

import matplotlib # использований функций библиотеки matplotlib для взаимодействия с графиком

from math import \* # использование всей библиотеки math для неявной работы с математическими выражениями

import numpy as np # использование библиотеки numpy для работы с вещественными диапазонами

# обновление графика

def update\_graph(canvas, start, end, step, x\_pad, y\_pad, function\_expression, is\_polar):

try: # попытка отрисовать график

x\_array = [x for x in np.arange(start, end, step)] # формируем массив значений x

y\_array = [eval(function\_expression) for x in x\_array] # формируем массив значений y

if is\_polar: # если нужен график в полярных координатах

plt.polar(x\_array, y\_array) # отрисовываем в полярных координатах

else: # если нужен график в декартовых координатах

plt.plot(x\_array, y\_array) # отрисовываем в декартовых координатах

plt.xticks(np.arange(start, end, x\_pad)) # устанавливаем сетку по x

plt.yticks(np.arange(min(y\_array), max(y\_array), y\_pad)) # устанавливаем сетку по y

canvas.draw() # рисуем график на холсте

except Exception as e: # произошла ошибка

tk.messagebox.showinfo("Ошибка вычисления", e) # выводим информацию о ошибке

# головная функция

def main():

def draw\_graph(): # команда отрисовки графика

update\_graph(canvas, # холст

float(func\_start.get()), # начало диапазона

float(func\_end.get()), # конец диапазона

float(func\_step.get()), # шаг функции

float(func\_x\_pad.get()), # шаг сетки по x

float(func\_y\_pad.get()), # шаг сетки по y

func\_expression.get(), # выражение функции

func\_is\_polar.get()) # необходимость полярных координат

def clear\_graph(): # команда очистки графика

plt.clf() # очистка графика

plt.grid() # установка координатной сетки

canvas.draw() # отрисовка холста

root = tk.Tk() # создаем приложение с графическим интерфейсом

plt.grid() # создаем график

canvas = FigureCanvasTkAgg(plt.figure(1), master=root) # создаем холст

plot\_widget = canvas.get\_tk\_widget() # создает компонент, на котором будет установлен график и который будет встроен в холст

# значения, которые привязаны к текстовым полям

# расшифровка значений находится на строке №29

func\_expression = tk.StringVar()

func\_start = tk.StringVar()

func\_end = tk.StringVar()

func\_step = tk.StringVar()

func\_x\_pad = tk.StringVar()

func\_y\_pad = tk.StringVar()

func\_is\_polar = tk.IntVar()

# установка начальных значений для всех параметров графика

func\_start.set('-15')

func\_end.set('15')

func\_step.set('0.1')

func\_x\_pad.set('3')

func\_y\_pad.set('0.2')

func\_is\_polar.set(0)

func\_expression.set('x\*sin(x)/(1+x\*\*2)')

#region UI

tk.Label(root, text="Выражение").grid(row=0, column=0)

tk.Entry(root, width=40, textvariable=func\_expression).grid(row=1, column=0)

tk.Label(root, text="Начало интервала").grid(row=0, column=1)

tk.Entry(root, width=15, textvariable=func\_start).grid(row=1, column=1)

tk.Label(root, text="Конец интервала").grid(row=0, column=2)

tk.Entry(root, width=15, textvariable=func\_end).grid(row=1, column=2)

tk.Label(root, text="Шаг").grid(row=0, column=3)

tk.Entry(root, width=15, textvariable=func\_step).grid(row=1, column=3)

tk.Label(root, text="Шаг сетки по x").grid(row=2, column=1)

tk.Entry(root, width=15, textvariable=func\_x\_pad).grid(row=3, column=1)

tk.Label(root, text="Шаг сетки по y").grid(row=2, column=2)

tk.Entry(root, width=15, textvariable=func\_y\_pad).grid(row=3, column=2)

tk.Button(root, text="+", command=lambda: func\_expression.set(func\_expression.get() + '+')).grid(row=4, column=0, pady=15)

tk.Button(root, text="-", command=lambda: func\_expression.set(func\_expression.get() + '-')).grid(row=4, column=1)

tk.Button(root, text="\*", command=lambda: func\_expression.set(func\_expression.get() + '\*')).grid(row=4, column=2)

tk.Button(root, text="/", command=lambda: func\_expression.set(func\_expression.get() + '/')).grid(row=4, column=3)

tk.Button(root, text="cos", command=lambda: func\_expression.set(func\_expression.get() + 'cos')).grid(row=5, column=0, pady=15)

tk.Button(root, text="sin", command=lambda: func\_expression.set(func\_expression.get() + 'sin')).grid(row=5, column=1)

tk.Checkbutton(text="Полярные координаты?", variable=func\_is\_polar).grid(row=5, column=2, pady=15)

plot\_widget.grid(row=6, column=0, columnspan=4, padx=15, pady=15)

tk.Button(root, text="Построить", command=draw\_graph).grid(row=7, column=0, columnspan=2)

tk.Button(root, text="Очистка", command=clear\_graph).grid(row=7, column=2, columnspan=2)

#endregion

draw\_graph() # рисуем график

root.title("Лабораторная №1 Соболь Е.В")

root.mainloop() # запускаем приложение

main() # запускаем головную функцию

# 5 Анализ аналогичных приложений

Веб-ресурс <https://www.mathway.com/ru/graph>: позволяет строить несколько графиков, при этом каждый график представляет собой отдельную задачу. Имеется масштабирование.

Веб-ресурс <http://www.yotx.ru/>: Данный сервис может строить обычные графики (вида y = f(x)), заданные параметрически, графики по точкам и графики функций в полярной системе координат, при этом, не заставляя, как многие программы, искать результат по всей двумерной плоскости. Сервис автоматически подберет масштаб графика и интервалы по осям координат так, чтобы график оказался перед вами. Одновременно на одной плоскости вы можете строить несколько графиков. Есть возможность отправить ссылку на получившийся график друзьям или поделиться с ними через социальные сети. Кроме того, вы даже сможете вставить интерактивный виджет с получившимся графиком. Также есть функции печати результата и загрузки графика в виде статичного изображения.

# 6 Информационные технологии для реализации

Рассмотрим примеры ИТ, позволяющие выполнить поставленную задачу. Для реализации графического интерфейса можно использовать фреймворки Kivy или PyQt.

**Kivy** был создан в 2011 году. Данный кросс-платформенный фреймворк Python работает на Windows, Mac, Linux и Raspberry Pi. В дополнение к стандартному вводу через клавиатуру и мышь он поддерживает мультитач. Kivy поддерживает ускорение GPU своей графики, что во многом является следствием использования OpenGL ES2. Во время разработки приложения через Kivy создается интуитивно понятный интерфейс (Natural user Interface), или NUI. Его главная идея в том, чтобы пользователь мог легко и быстро приспособиться к программному обеспечению без чтения инструкций. Kivy не задействует нативные элементы управления, или виджеты. Все его виджеты настраиваются. Это значит, что приложения Kivy будут выглядеть одинаково на всех платформах. Тем не менее, это также предполагает, что внешний вид приложения будет отличаться от нативных приложений пользователя. Это может стать как преимуществом, так и недостатком, в зависимости от целей.

**PyQt** — это набор инструментов для графического интерфейса. Это интерфейс Python для Qt , кроссплатформенной библиотеки графического интерфейса. PyQt был разработан RiverBank Computing Ltd. PyQt совместим со всеми популярными операционными системами, включая Windows, Linux и Mac OS. Существует двойная лицензия, доступная под GPL, а также коммерческая лицензия.

Для работы с графиками помимо выбранной мною библиотеки matplotlib: plotly существуют и другие, например, Seaborn или Leather.

**Seaborn** — это одна из Python-библиотек, которая построена на основе Matplotlib, но более высокоуровневая. В результате, все, что можно построить в Matplotlib, можно построить в Seaborn. К тому же имеет дополнительные инструменты, например, heatmap и violin plots, а также встроенные темы оформления.

**Leather** — это библиотека Python для создания графиков. Leather работает с любыми данными. Поскольку эта библиотека относительно новая, над некоторыми частями документации еще ведется работа. Графики, которые можно создать при помощи Leather, самые базовые, но это и было целью автора библиотеки.

Отмечу, что наиболее популярная библиотека среди всех это matplotlib: plotly. У неё более простой и красочный построитель графиков, чем у аналогов.

# 7 Логическая схема работы приложения

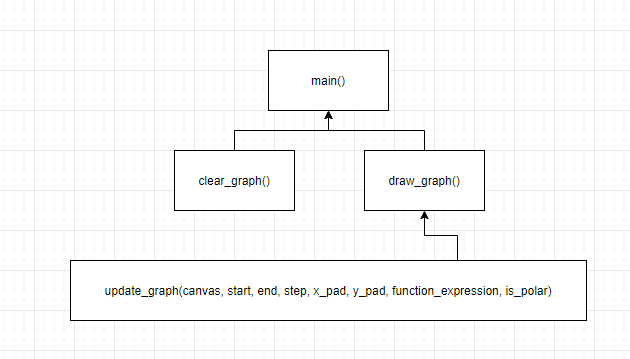


Рисунок 2 - Логическая схема

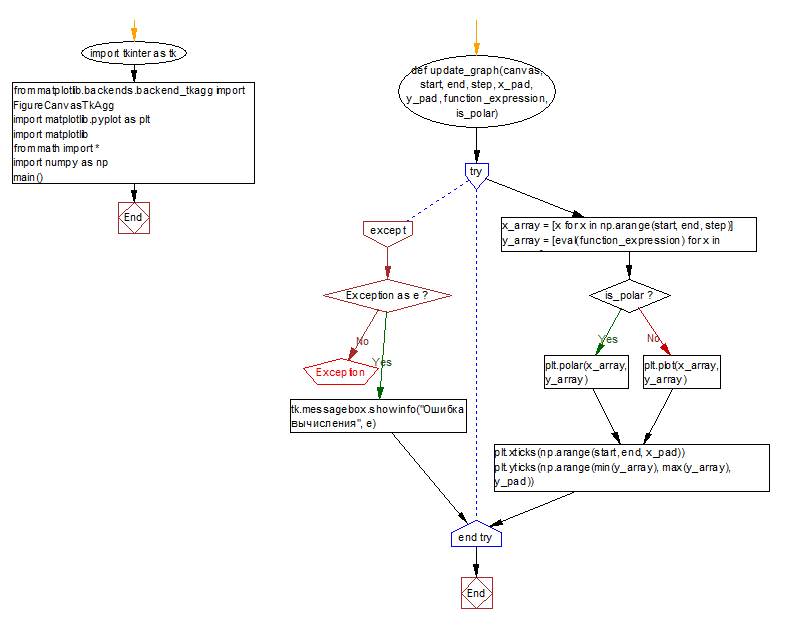


Рисунок 3 - Логическая схема (1)

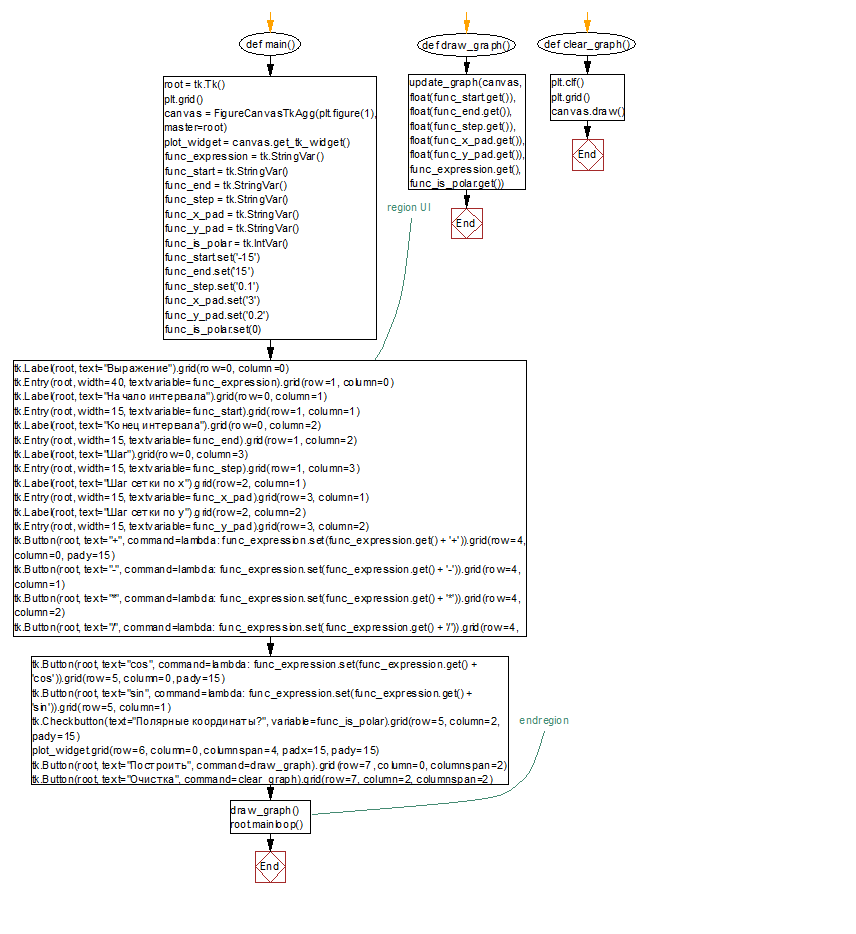


Рисунок 4 - Логическая схема (1)

# Приложение

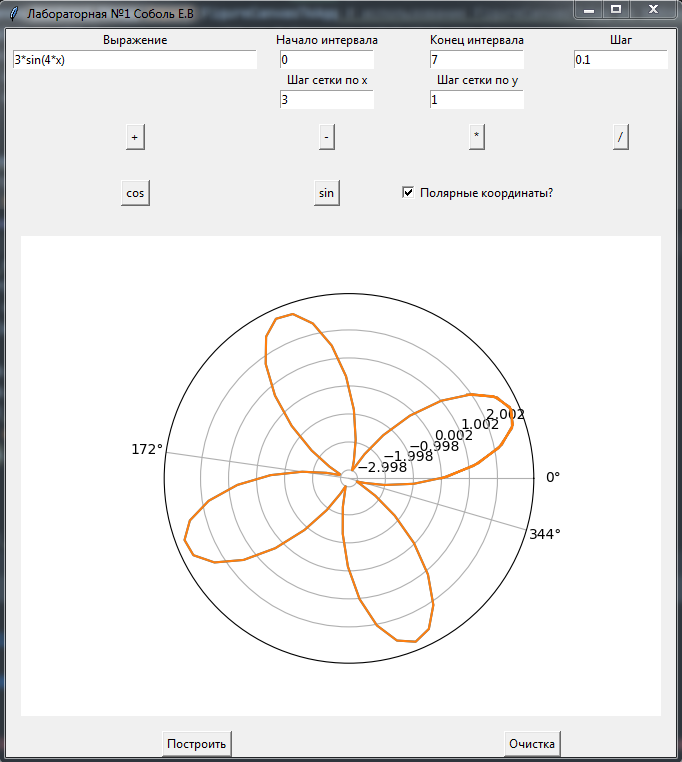


Рисунок 5 - 3(sin4x) полярные

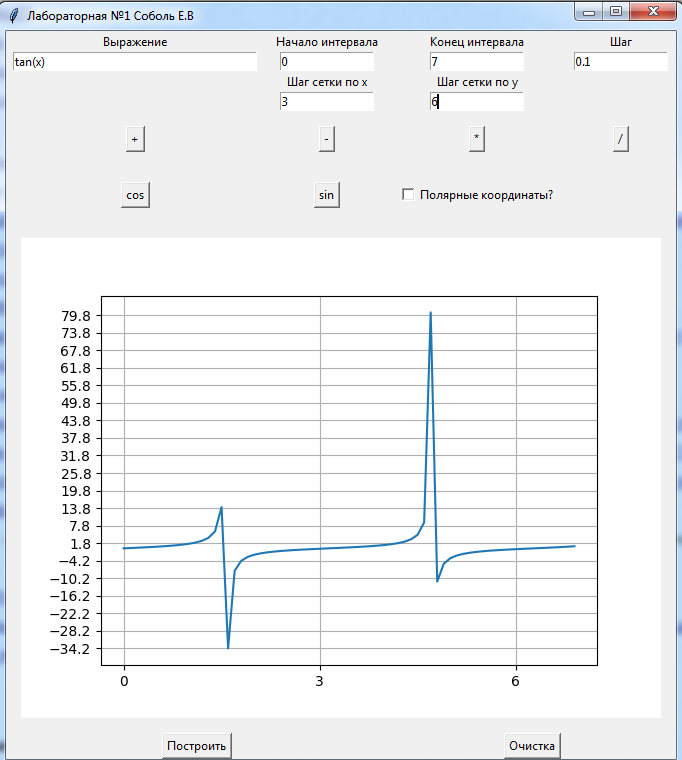


Рисунок 6 - tgx

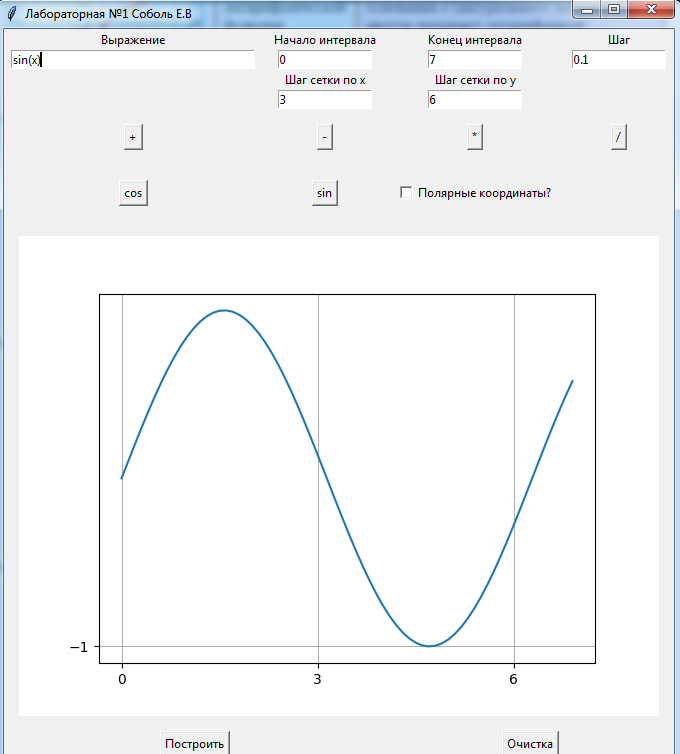


Рисунок 7 - sinx

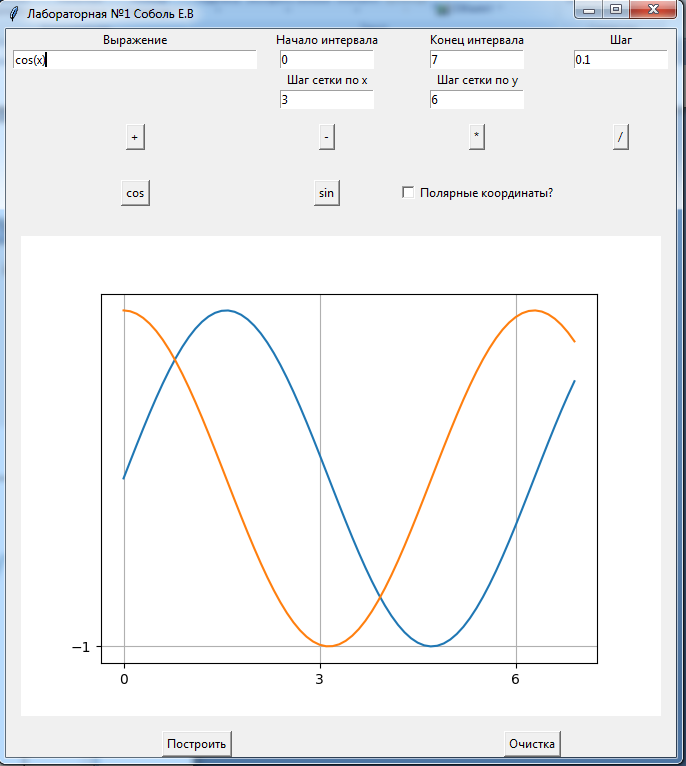


Рисунок 8 - sinx и cosx

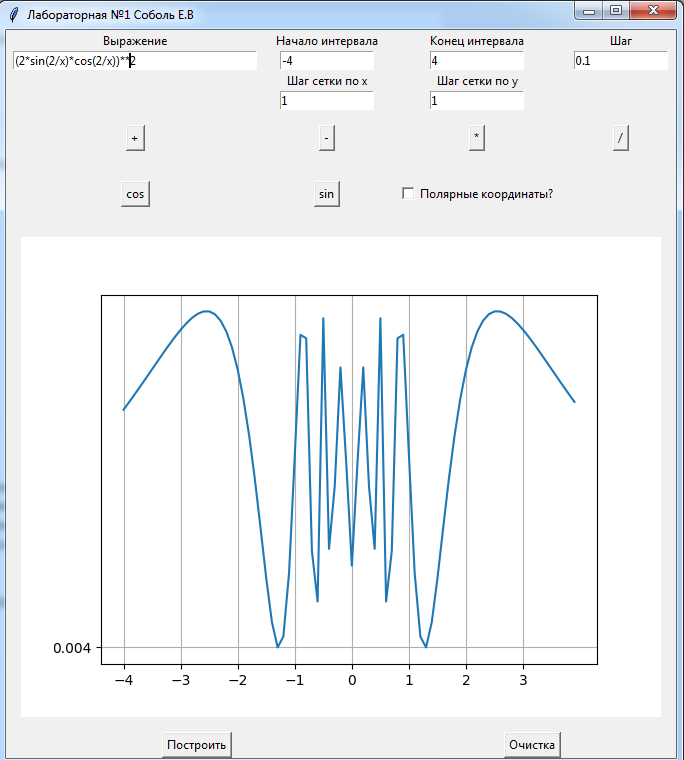


Рисунок 9 - (2sin(2/x)cos(2/x))^2

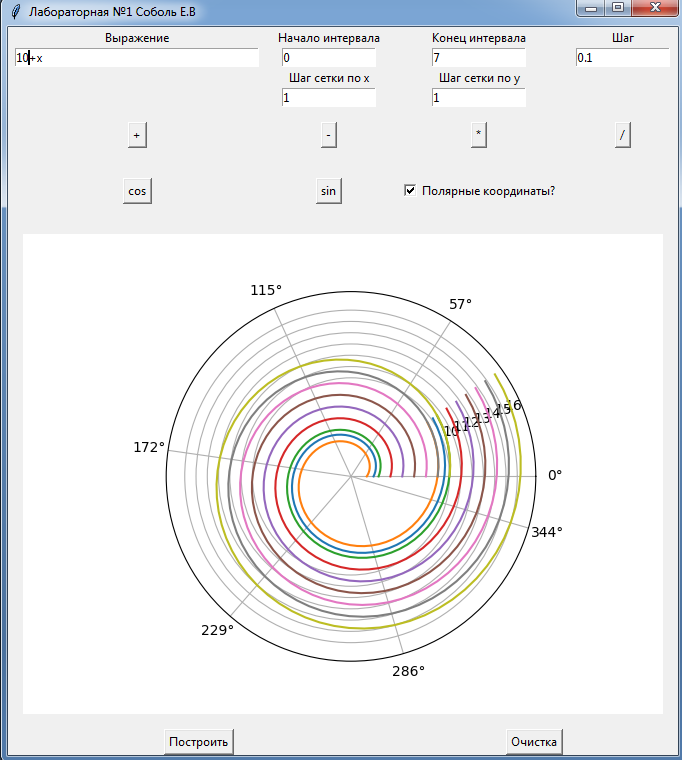


Рисунок 10 - 2+x ... 10+x полярные