Конференция учащихся

муниципальных образовательных учреждений города Калуги «Старт в науку»

Секция: информатика

Приложение с графическим интерфейсом «Построитель графиков»

Автор работы Бугдиев Амрихудо Илхомович,

Класс: 8 «Г»

Образовательное учреждение: Муниципальное бюджетное общеобразовательное учреждение

“Средняя общеобразовательная школа № 13 города Калуги”

Научный руководитель: Гущин Глеб Борисович

Должность: Учитель Информатики

Калуга, 2023

**Введение**

График - это отражение изменения величин, каких-то значений. Это очень удобно, что можно наглядным рисунком показать, как быстро меняется величина, прибывает она или убывает. Еще одна замечательная возможность связана с пересечением графиков. Это значит, что у них есть что-то общее.

В школе нас учат строить графики основных функций: линейной и квадратичной и находить точки пересечения. Но эту работу можно автоматизировать.

**Проблема:** операции построения графиков трудоемки и нуждаются в автоматизации.

**Цель:**

Получить приложение, которое сможет строить графики и показывать точки их пересечения.

**Задачи:**

1. Повторить связанные темы из школьного курса алгебры
2. Найти библиотеку для создания приложений на Python
3. Определиться со сценариями пользователей
4. Сделать макет интерфейса
5. Собрать интерфейс в коде
6. Запрограммировать логику приложения
7. Провести тесты.

**Объект** **исследования**: создание приложений с графическим интерфейсом

**Предмет исследования:** использование компонента «холст» для построения графиков функций

**Методы** **исследования**: анализ, моделирование, эксперимент

**Гипотеза**: предположим, что возможностей языка Python достаточно для написания подобного приложения

**Практическая значимость**: продукт данного проекта окажет помощь учителям (для создания примеров) и пригодится для расчетов.

**История функций**

Слово «функция» имеет много значений. Чаще всего мы имеем в виду зависимость одной величины от другой по какому-то правилу, формуле.

Люди работают с функциями уже много тысяч лет. Первые функции – формулы касались правил вычисления размеров фигур, из периметра, площади и объема. Позже мы пришли к квадратным уравнениям. Для них чаще всего самым простым способом выяснить ответ оказывалось построить график функции, описанной в уравнении. Сейчас мы работаем еще и с графиками реальных процессов и явлений, для которых сложно подобрать функцию, но представление в виде графика позволяет успешно анализировать такие явления: запись сейсмических датчиков, кардиограмма, колебания звуковых волн и т.п.

**Построение графика функции на плоскости**

Для построения графика на координатной плоскости мы должны получить набор точек, принадлежащих этому графику. Для этого мы перебираем значения аргумента функции и, проведя вычисления, получаем результат функции. Эти два числа мы рассматриваем как координаты точки. После чего наносим точки на плоскость и соединяем их гладкой линией.

**Построение графика функции в программной среде**

Я изучаю язык программирования Python, потому создавать приложения я решил на нем. Изучив информацию в Интернете, я выяснил, что есть несколько библиотек для моей задачи. Однако, самая быстрая в освоении – tkinter. Она входит в стандартную библиотеку ядра языка Python и содержит стандартные виджеты такие как кнопки, окна, рамки, переключателя, поля ввода, метки вывода и прочее. Также присутствует холст, для которого существует рисователь графических примитивов: окружностей, прямоугольников, многоугольников, линий и т.п. Мы можем построить график с помощью рисования линии на холсте, передав ей вычисленные нами точки.

**Ход работы**

В первую очередь, мы определились с макетом приложения (см приложение1). Наше приложение имеет кнопки вызова своих основных функций, три поля ввода для числовых коэффициентов, разнообразные подписи-метки и поле построения графика (холст). После этого, мы написали код, создающий описанный интерфейс. Затем начали реализовывать логику где и столкнулись с непредвиденными сложностями.

Во-первых начало координат у компьютерных систем, как правило, не посередине пространства, а в левом верхнем углу. Причем ось икс направлена вправо, а ось игрек – вниз. Для того, что бы правильно поставить точку в таком положении, необходимо значение по икс сложить со значением середины длины пространства, а для игрек – вычесть его из «середины».

Во-вторых, пользователь может ввести в поля ввода нечисловые значения или вообще ничего не ввести, что вызовет ошибку программы. Проблема была решена следующим образом: если в ячейке оказывается не число или ячейка оказывается пустой при нажатии на кнопку «нарисовать график», то значение этой ячейки принимается равное нулю.

После окончания разработки и тестирования была проведена компиляция программы в исполняемый файл с помощью утилиты PyInstaller.

**Описание основных разделов программы**

Данный код содержит функции и переменные для создания программы-калькулятора графиков квадратичных и линейных функций с возможностью нахождения точек пересечения между графиками.

Первые несколько строк импортируют модуль tkinter для создания графического интерфейса, устанавливают размеры сетки, шаг сетки, и начальные значения для списков, которые будут содержать графики и точки пересечения.

Функция stop\_the\_program() будет использоваться в кнопке "Выход", которая закроет окно программы. Функция del\_all\_graphs() будет использоваться в кнопке "Удалить все графики", которая удалит все нарисованные графики и точки пересечения.

Функция draw\_show\_intersection\_points() будет вызываться при нажатии на кнопку "Найти точки пересечения". Она проверит, есть ли два или более графика, затем для каждой пары графиков определит точки пересечения. Если точки пересечения найдены, они будут отмечены на холсте и добавлены в список точек пересечения.

Функция draw\_graph() будет вызываться при нажатии на кнопку "Нарисовать график". Она извлекает коэффициенты a, b и c из соответствующих полей ввода, проверяет, являются ли они числами, и преобразует их в целые числа. Затем функция создает список точек, по которым будет нарисован график, используя переданные коэффициенты.

Для рисования графиков используется элемент Canvas из библиотеки tkinter, который позволяет создавать фигуры на холсте. Кроме того, на форме присутствуют поля ввода для коэффициентов a, b и c, кнопки для нахождения точек пересечения и удаления графиков, а также метки для вывода информации о графиках и точках пересечения.

**Заключение**

По итогам работы надо сказать, что цель была достигнута в полном объеме. Мы справились со сложностями и реализовали желаемый функционал, попутно углубив знания языка Python, навыки разработки приложений с графическим интерфейсом и изучив некоторые приемы работы с библиотекой tkinter.

Приложение можно развивать дальше. Например, можно добавить переключать вида функций, добавить больше разных видов графиков и сохранение-загрузку холста с построенными графиками.

**Литература**

Построение графиков функций <https://nauchkor.ru/pubs/postroenie-grafikov-funktsiy-v-sredney-shkole-5c1a51b87966e104f6f85223>

Официальная документация по библиотеке tkinter <https://docs.python.org/3/library/tkinter.html>

Русскоязычный сайт-справочник по tkinter <https://metanit.com/python/tkinter/>

**Приложение 1 – макет приложения**

Прекратить работу

Нарисовать график

Показать точки пересечения

Удалить все графики

А

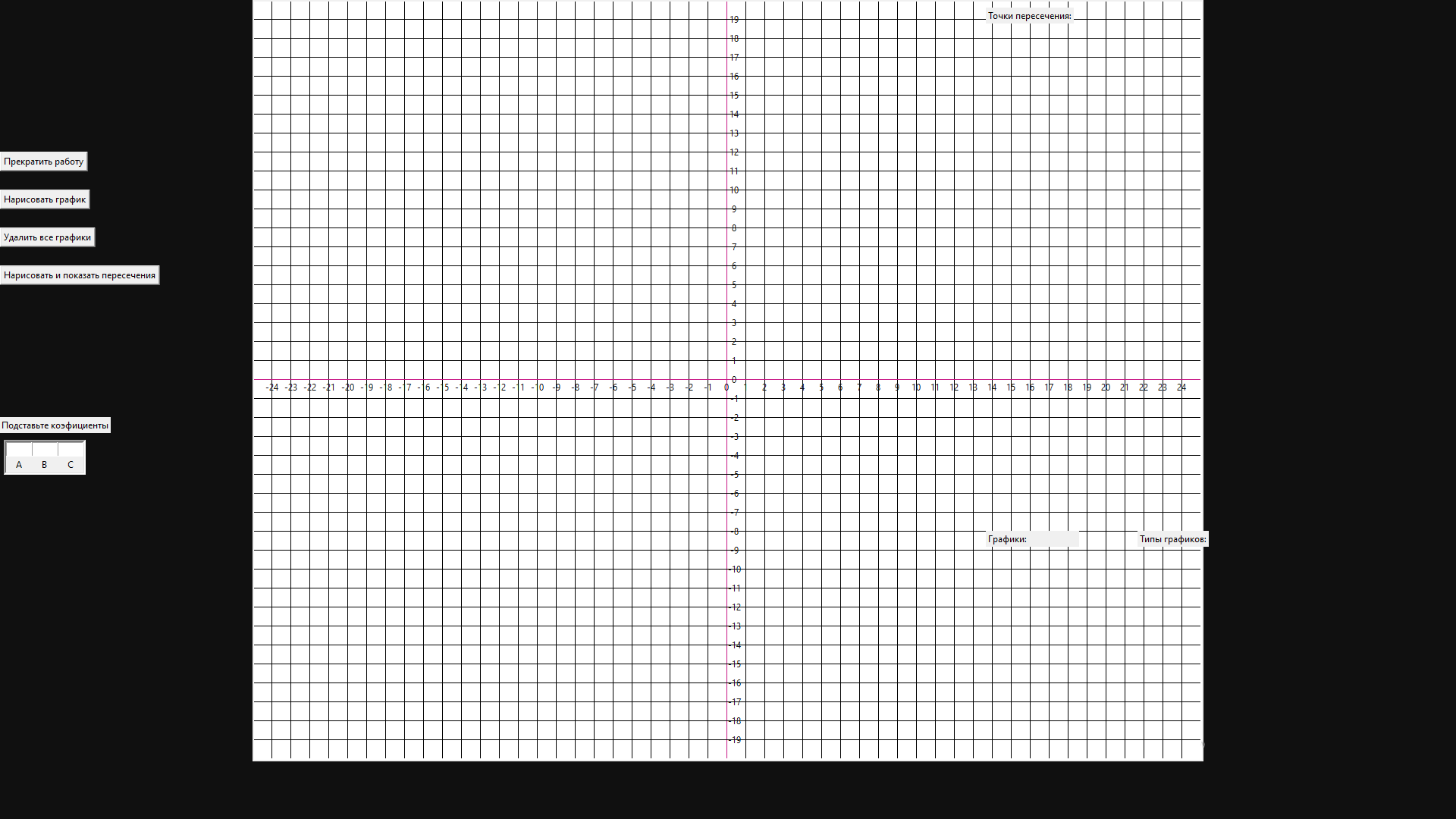
Подставьте коэффициенты

Б

В

Область построения

**Приложение 2. Внешний вид приложения**



**Приложение 3 Инструкция для пользователя**

Данный код создает окно с графическим интерфейсом для рисования и анализа графиков квадратичных и линейных функций.

Окно состоит из нескольких элементов:

1. Поля для ввода коэффициентов a, b, c квадратичной функции
2. Кнопка "Рисовать график" для рисования графика выбранной функции
3. Список графиков, которые были нарисованы, с возможностью их удаления
4. Кнопка "Удалить все графики" для удаления всех нарисованных графиков
5. Кнопка "Найти точки пересечения" для поиска точек пересечения между графиками, которые были нарисованы
6. Список найденных точек пересечения

После запуска окна пользователь может вводить коэффициенты a, b, c и рисовать график, нажимая на кнопку "Рисовать график". График построенной функции будет отображаться на холсте с сеткой размера 50x40 и шагом в 25 пикселей.

Пользователь может нарисовать несколько графиков, их список будет отображаться в соответствующем виджете. Пользователь может удалить отдельный график, выбрав его в списке и нажав кнопку "Удалить выбранный график", или удалить все графики, нажав кнопку "Удалить все графики".

При нажатии на кнопку "Найти точки пересечения" программа находит все точки пересечения между всеми нарисованными графиками. Найденные точки пересечения отображаются на холсте зелеными кругами, их координаты выводятся в соответствующем виджете.

Когда пользователь заканчивает работу с программой, он может нажать на кнопку "Выход" или закрыть окно, чтобы завершить работу программы.

**Приложение 4 Код программы**

# импортируем tkinter

from tkinter import \*

# размеры сетки

N\_X = 50

N\_Y = 40

# сторона квадрата в сетке

step = 25

# начальная формула

graphs = []

intersection\_points = []

intersection\_points\_poses = []

# функция прекращения работы которую будем использовать в кнопке

def stop\_the\_program():

window.destroy()

# функция для удаления всех графиков

def del\_all\_graphs():

global graphs, intersection\_points, intersection\_points\_poses

for graph in graphs:

canvas.delete(graph[0])

graphs = []

text\_graphs.set('Графики:'.ljust(30, ' '))

text\_types\_graphs.set('Типы графиков:')

for point in intersection\_points:

canvas.delete(point)

intersection\_points, intersection\_points\_poses = [], []

text1.set('Точки пересечения:')

# поиск точек пересечения

def draw\_show\_intersection\_points():

# проверяем больше ли 2 кол-во графиков

if len(graphs) > 1:

global intersection\_points\_poses

intersection\_points\_poses = []

text1.set('Точки пересечения:')

# цикл для взятия графика

for i in range(len(graphs) - 1):

# цикл для взятия другого графика и поиска точек пересечения с первым

for j in range(i + 1, len(graphs)):

new\_formula = [0, 0, 0]

# цикл для приравнивания графиков

for coefficient in range(len(new\_formula)):

new\_formula[coefficient] = graphs[i][1][coefficient] - graphs[j][1][coefficient]

poses = []

if new\_formula[0] and new\_formula[1]:

d = (new\_formula[1] \*\* 2) - (4 \* new\_formula[0] \* new\_formula[2])

if d < 0:

continue

else:

for a in range(-1, 2, 2):

poses.append([])

poses[-1].append(((-new\_formula[1]) + a \* (d \*\* 0.5)) / (2 \* new\_formula[0]))

poses[-1].append(

poses[a][0] \*\* 2 \* graphs[i][1][0] +

+ poses[a][0] \* graphs[i][1][1] + graphs[i][1][2])

elif new\_formula[0]:

if -new\_formula[2] / new\_formula[0] >= 0:

for a in range(-1, 2, 2):

poses.append([])

poses[-1].append(a \* (-new\_formula[2] / new\_formula[0]) \*\* 0.5)

poses[-1].append(poses[a][0] \*\* 2 \* graphs[i][1][0] + graphs[i][1][2])

else:

continue

elif new\_formula[1]:

poses.append([])

poses[-1].append(-new\_formula[2] / new\_formula[1])

poses[-1].append(graphs[i][1][1] \* poses[-1][0] + graphs[i][1][2])

for a in range(len(poses)):

if (poses[a][0], poses[a][1]) not in intersection\_points\_poses:

intersection\_points.append(canvas.create\_oval(

(N\_X \* step // 2 - -(poses[a][0] \* step) - 3, (N\_Y \* step) // 2 - (poses[a][1] \* step) - 3),

(N\_X \* step // 2 - -(poses[a][0] \* step) + 3, (N\_Y \* step) // 2 - (poses[a][1] \* step) + 3),

fill='green'))

new\_text = text1.get() + '\n'

text1.set(new\_text + f'{(poses[a][0], poses[a][1])}')

intersection\_points\_poses.append((poses[a][0], poses[a][1]))

# функция для рисования графика

def draw\_graph():

a, b, c = ent\_a.get(), ent\_b.get(), ent\_c.get()

if not a or a.isalpha():

a = 0

if not b or b.isalpha():

b = 0

if not c or c.isalpha():

c = 0

a, b, c = int(a), int(b), int(c)

if a != 0:

type\_f = 'квадратичный'

elif a == 0:

type\_f = 'линейный'

formula = [a, b, c]

# список точек по которым мы будем рисовать график

points = []

# цикл для определения точек

for x in range(-N\_X \* 4, N\_X \* 4):

x /= 4

y = 0

y += x \*\* 2 \* a

y += x \* b

y += c

# добавление позиции точки

points.append((N\_X \* step // 2 - -(x \* step), N\_Y \* step // 2 - y \* step))

# отрисовываем график и добовляем его в список с графиками

graphs.append([canvas.create\_line(\*points, fill='red', width=2), formula])

new\_text = text\_graphs.get() + '\n'

coefficients = ('x^2', 'x', '')

formula\_text = []

for m in range(len(formula)):

if not formula[m] - int(formula[m]):

formula[m] = int(formula[m])

if formula[m] != 0:

if formula[m] < 0:

formula\_text.append(f'({str(formula[m]) + coefficients[m]})')

elif formula[m]:

formula\_text.append(f'{str(formula[m]) + coefficients[m]}')

text\_graphs.set(new\_text + ' + '.join(formula\_text))

text\_types\_graphs.set(text\_types\_graphs.get() + '\n' + type\_f)

# создаём окно

window = Tk()

# меняем цвет заднего фона окна

window['bg'] = '#101010'

text1 = StringVar()

text1.set('Точки пересечения:')

text\_graphs = StringVar()

text\_graphs.set('Графики:'.ljust(30, ' '))

text\_types\_graphs = StringVar()

text\_types\_graphs.set('Типы графиков:')

# создём холст в котором мы и будем отрисовывать график

canvas = Canvas(window, bg='#fff', height=N\_Y \* step, width=N\_X \* step)

# основные кнопки

# создаём кнопку для закрытия окна

Button(window, text='Прекратить работу', command=stop\_the\_program).place(x=0, y=200)

# кнопка для рисования графика

Button(window, text='Нарисовать график', command=draw\_graph).place(x=0, y=250)

# кнопка для удаления всех графиков

Button(window, text='Удалить все графики', command=del\_all\_graphs).place(x=0, y=300)

# кнопка для рисования и показа точек пересечения

Button(window, text='Нарисовать и показать пересечения', command=draw\_show\_intersection\_points).place(x=0, y=350)

# создаём окно ввода

name = Label(text="Подставьте коэфициенты", width=20)

name.place(x=0, y=550)

# отрисовываем линии

for i in range(1, N\_X):

if i == N\_X // 2:

canvas.create\_line(

(i \* step, 0), (i \* step, N\_Y \* step), fill='#C71585')

else:

canvas.create\_line(

(i \* step, 0), (i \* step, N\_Y \* step), fill='black')

# метка для оси Ox

canvas.create\_text(

i \* step, N\_Y // 2 \* step + 10, text=str(i - N\_X // 2), fill='#101010')

# отрисовываем линии

for i in range(1, N\_Y):

if i == N\_Y // 2:

canvas.create\_line(

(0, i \* step), (N\_X \* step, i \* step), fill='#C71585')

else:

canvas.create\_line(

(0, i \* step), (N\_X \* step, i \* step), fill='black')

# метка для оси Oy

canvas.create\_text(

N\_X // 2 \* step + 10, i \* step, text=str(N\_Y // 2 - i), fill='#101010')

# дбавляем холст в окно

canvas.pack()

# Создание фрейма

frame = Frame(bd=3, relief=SUNKEN)

frame.place(x=5, y=580)

# Создание трех полей для ввода

ent\_a = Entry(frame, width=5)

ent\_a.grid(row=0, column=0)

ent\_b = Entry(frame, width=5)

ent\_b.grid(row=0, column=1)

ent\_c = Entry(frame, width=5)

ent\_c.grid(row=0, column=2)

# Создание букв для каждого из полей

lab\_a = Label(frame, text="A")

lab\_a.grid(row=1, column=0)

lab\_b = Label(frame, text="B")

lab\_b.grid(row=1, column=1)

lab\_c = Label(frame, text="C")

lab\_c.grid(row=1, column=2)

# текст

label1 = Label(window, textvariable=text1)

label2 = Label(window, textvariable=text\_graphs)

label3 = Label(window, textvariable=text\_types\_graphs)

# отображаем текст

label1.place(x=1300, y=10)

label2.place(x=1300, y=700)

label3.place(x=1500, y=700)

# следующая строчка отвечает за полноэкранный режим окна

window.attributes('-fullscreen', True)

# основной цикл

window.mainloop()