Лабораторная работа №8: «GUI, модуль Tkinter»

Цель работы:

В данной работе студенты познакомятся:

- с принципами организации событийно-управляемого программирования;
- с основами организации графического интерфейса пользователя (GUI) и виджетами модуля `Tkinter`*);
- со способами построения графиков функций методами этого модуля.

Замечание: Студенты, при решении задач этой лабораторной работы, не ограничиваются выбором модуля `Tkinter`. Приветствуется выбор других модулей или библиотек, например, PyQt.

Постановка задачи

Вывести на экран в графическом режиме графики двух функций на интервале от `Xнач` до `Xкон` с шагом `dx`. Первая функция задана с помощью ряда Тейлора, ее вычисление должно выполняться с точностью ε. Значение параметра `b`, определяющего смещение функции по оси ординат, вводится с клавиатуры. Графики должны быть плавными и различаться цветами.

$$y(x) = 2 \cdot \left(1 - \frac{2^2 \cdot x^2}{3!} + \frac{2^4 \cdot x^4}{5!} - \cdots + (-1)^n \cdot \frac{2^{2 \cdot n} \cdot x^{2 \cdot n}}{(2 \cdot n+1)!} + \cdots\right) \quad -\infty < x < \infty$$

$$z(x) = \frac{Sin(2 \cdot x)}{x} + b$$

Теоретическое введение

Для выполнения такой работы нам необходимо познакомиться с модулями Python, которые можно использовать для построения графика функции и организации интерфейса с пользователем. Кроме этого нам необходимо познакомиться с понятием "событие" и принципами организации программ, использующих эти события для построения GUI.

Событийно-управляемое программирование

Под событием в операционной системе понимается любое действие пользователя при его взаимодействии с программным интерфейсом: нажатие клавиш клавиатуры, одиночные и двойные щелчки кнопками мыши, перемещение мыши в окне. Кроме этих событий в системе существуют события, которые вызываются как самой операционной системой, так и другими приложениями: работа системного таймера, информационный обмен между запущенными процессами.

---

*) В пособии, по тексту, имя модуля пишется с большой буквы, как имя собственное. В скрипте имя модуля следует писать с маленькой буквы (tkinter) для Python 3. В Python 2 имя модуля пишется с большой буквы - Tkinter.

Все события, возникающие в вычислительной системе, воспринимается операционной системой. Эти события преобразуется в сообщение – запись, содержащую необходимую информацию о событии.

Например, это может быть код клавиши и переход её состояния – была клавиша нажата или отпущена. При возникновении события от мыши, в сообщении будет содержаться информация о том, какая кнопка была нажата, был ли это одиночный или двойной клик, а так же координаты маркера мыши в момент клика.

Сообщения поступают в общую очередь, откуда распределяются по очередям приложений.

Каждое приложение имеет свой цикл обработки сообщений, в котором выбирается сообщение из очереди приложения. Затем, через операционную систему, вызывается подпрограмма, предназначенная для обработки этого сообщения.

На рис.8.1 представлена структура программы, управляемой событиями.

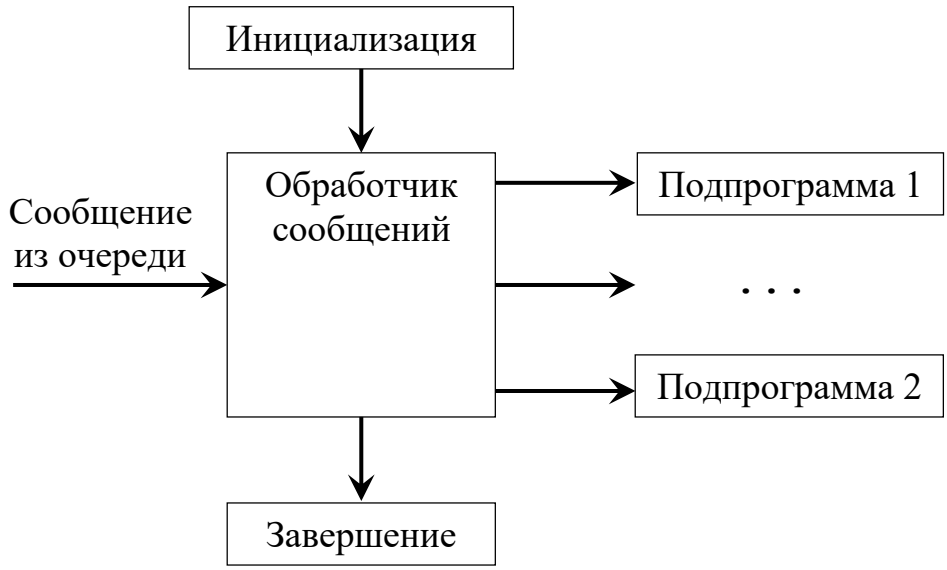

Рис.8.1. – Структура программы, управляемой событиями

Можно сделать следующее заключение – программа состоит из главной части, которая выполняет инициализацию и завершение приложения, цикла обработки сообщений, и набора обработчиков сообщений (подпрограмм).

Модуль Tkinter

В Python реализовано много модулей и библиотек, позволяющих создавать графический интерфейс пользователя (GUI). Вот небольшой перечень: Tkinter, PyQt5, PyGTK, wxPython, Pythonwin, и др.

Стандартным модулем, который устанавливается вместе с Python по умолчанию, является модуль `Tkinter`. У этого модуля понятный и ясный синтаксис и его можно использовать в простых программах с графическим интерфейсом.

Далее будем рассматривать решение нашей задачи, поставленной в начале этой лабораторной работы, в рамках возможностей модуля Tkinter.

Для использования методов и функций модуля необходимо выполнить импорт модуля. Это реализуется строкой:

```
from tkinter import *
```

Другая форма импорта – с назначением псевдонима (алиас – калька с английского, alias – псевдоним):

```
import tkinter as tk
```

Виджеты

В Tkinter визуальные компоненты, используемые для построения пользовательского графического интерфейса, называют виджетами (*widget*, от англ. *window gadget*).

Виджет – лемент интерфейса – примитив графического интерфейса пользователя (GUI), имеющий стандартный внешний вид и выполняющий стандартные действия. Иногда можно встретить и другое название контро (англ. *control*). Вот небольшой перечень виджетов, которыми мы воспользуемся при решении нашей задачи:

– кнопка (`Button`) – используется для запуска команды, или, например, для выбора следующего действия. Например, кнопка "Рисовать" запустит процесс рисования графика, а по кнопке "Выход" скрипт завершит работу;

– однострочное текстовое поле (`Entry`) – этот виджет предназначен для вывода или ввода только одной, как правило, небольшой строки, например, название предмета или фамилия;

– метка (Label) – используется, как правило, для информирования пользователя. Например, метка может быть описанием однострочного текстового поля Entry: назначение поля;

– холст, канва, полотно (Canvas) – область рисования графических изображений;

– окна сообщений (message box) – это диалоговые окна, которые позволяют выводить сообщения, предупреждения или ошибки при взаимодействии с пользователем. Кроме этого имеются диалоговые окна для работы с файлами: окна выбора файла при открытии или сохранении файла.

Все виджеты имеют два набора конфигурационных параметров. Один набор параметров является общим для всех виджетов: размеры, положение на форме.

Второй набор параметров определяется индивидуальными характеристиками виджета: состояние счетчика виджета Spinbox, или индекс значения, выбранного пользователем в виджете Combobox. Ознакомьтесь самостоятельно с этими виджетами (Spinbox, Combobox).

Некоторые виджеты, например Entry, Button, получают фокус, что позволяет связать такой виджет с клавиатурой. Фокус может быть переведён на виджет, кликом мыши на нём или с использованием клавиши Tab.

События, связанные с виджетом (клик мыши, нажатие клавиш клавиатуры) могут быть привязаны к методу (функция), который будет обрабатывать событие. Например, при клике на полотне может быть вызван метод, который позволяет получить координаты курсора мыши в момен клика.

Важно: Виджеты, описываемые в скрипте обязательно должны быть обработаны упаковщиком, который размещает их на форме.

Существуют три метода упаковки виджетов:

Метод pack

В этом методе указывается, как виджет будет заполнять пространство формы: где будет размещаться, и как будет изменяться при изменении размеров формы.

Метод place

Этот метод позволяет программисту контролировать положение виджета путём задания координат, и его размеров. Координаты положения задаются относительно верхнего левого угла окна, к которому привязан виджет. При этом если окно является вложенным и его координаты меняются, то относительные координаты виджета не меняются.

Метод grid

Наиболее часто используемый упаковщик. В этом методе окно делится условной сеткой на ячейки (`grid`). Программист задает ячейку, в которой будет размещаться виджет, через номер столбца и колонки. При этом можно указать, на скольких строках и столбцах будет размещаться виджет.

В нашем скрипте будем использовать этот упаковщик.

Заметим, что не рекомендуется смешивать упаковщики при формировании одной формы. Больше информации о виджетах и методах их упаковки можно получить в соответствующей литературе или в сети Интернет.

Интерфейс программы

Выделим в нашем скрипте две части. В головной части мы поместим описания функций, которые будут выполнять некоторый набор операций, а во второй – описание виджетов, которые будут организовывать интерфейс, для взаимодействия с пользователем.

Нам потребуются следующие виджеты:

- `Canvas` – полотно для графического представления функций;
- `Entry` – поле для ввода данных, например, смещение графика по оси ординат или вывода данных о положении курсора мыши;
- `Button` – кнопка для задания действия: вывод обновлённого графика, после изменения значений, задаваемых через поле Entry; завершение работы программы;
- `Label` – метка, для описания полей Entry.

Уточним техническое задание к интерфейсу скрипта.

Полотно (Canvas) будет занимать основную верхнюю часть формы. Поля ввода, метки с описанием этих полей и кнопки будут размещены в нижней части формы. В скрипте опишем следующие поля:

– поля для вывода координат указателя мыши в момент клика;

– поля для задания границ области координат: абсциссы и ординаты. Задание этих полей позволит просматривать форму графика в разных областях координат;

– поля для задания смещения функции и шага, который будет использоваться при нанесении разметки на осях координат.

– кнопки для вывода графика и завершения работы скрипта.

Макет окна представим в виде рисунка, см. рис.8.2

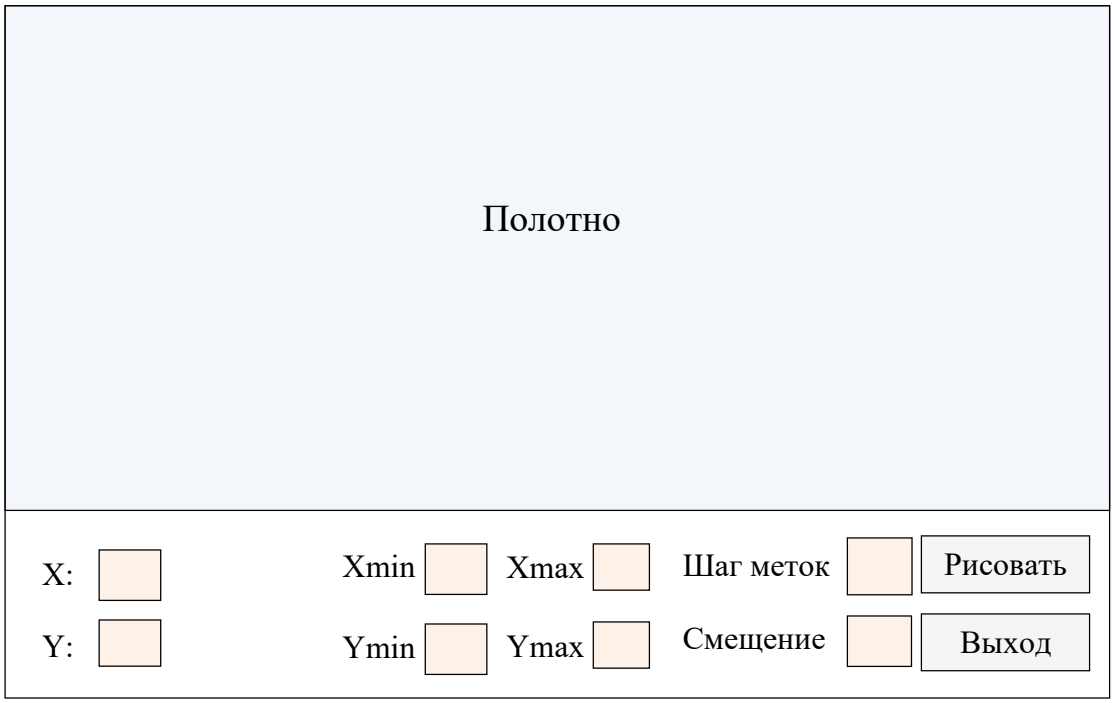

Рис.8.2. - Макет окна программы

Прежде, чем мы начнём описывать наш макет в виде скрипта познакомимся кратко с параметрами используемых виджетов.

Canvas – полотно

Методы виджета Canvas позволяют создавать графические примитивы, с помощью которых формируется рисунок:

– линия – create_line(). Для рисования линии необходимо задать начальные и конечные координаты концов линии, её толщину, цвет, тип (сплошная или пунктир), тип концов (наличие или отсутствие стрелок);

– замкнутый контур – create_poligon(). Для рисования полигональной линии задаётся список координат, через которые пройдёт контур, параметры, описанные для линии, цвет заливки фигуры. Кроме этого можно задать способ проведения линии между точками. В этом случае координаты точек

будут играть весовую роль (линия не будет проходить через точки), но будут влиять на форму получаемой фигуры;

– многоугольник – create_rectangle. Этот примитив позволяет рисовать прямоугольники;

– овал – create_oval. Для рисования эллипса и окружности задаются координаты вершин прямоугольника (квадрата), в который он будет вписан. При этом определяются координаты верхнего левого угла и нижнего правого, цвет заполнения. Также как и для линии можно задавать ширину контура, цвет, и т.п.;

– круг, дуга – create_arc(). Этот примитив используется для рисования дуг, секторов или сегментов (определяется параметром `style`) . Он подобен овалу, но имеет большее число параметров;

– текст – create_text(). Этот примитив используется для вывода текстовой информации в графическом поле, которым представляется полотно. Текст может быть многострочным. В качестве параметров задаются координаты якоря текстового поля (anchor), относительно которого текст выравнивается в этом поле. Можно задавать цвет фона и символов, тип шрифта, размер, …

Следует отметить важную особенность полотна. Все рисуемые примитивы возвращают идентификатор, который можно присвоить переменной, а затем использовать её значение, например, для удаления с полотна изображения примитива.

Система координат полотна

Разберёмся с системой координат, которая реализована на полотне (Canvas).

Отметим, что в этой системе координат единицей измерения является пиксел. В модуле Tkinter нет метода, позволяющего создать собственную пользовательскую систему координат, как это реализовано в модуле Turtle, см. лабораторную работу №7.

Организация системы координат на полотне показана на рис.8.3.

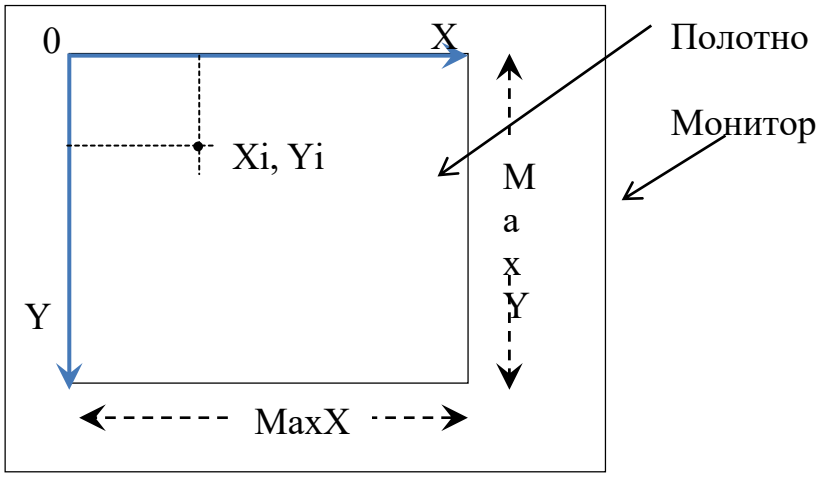

Рис.8.3.– Система координат полотна и параметры окна

На этом рисунке показано, что ось абсцисс направлена с лева направо, а ось ординат – сверху вниз. Следует обратить внимание на то, что размеры полотна (MaxX, MaxY), а так же координаты точки (Xi, Yi) указываются в пикселах.

Учитывая эту особенность модуля, а так же то, что пользователь должен вводить или получать координаты в собственной системе координат, нам придётся преобразовывать данные, введённые пользователем, в пикселы для рисования линий графика и возвращать данные, преобразованные в систему координат пользователя.

Выведем формулы преобразования координат графика, определённых в пользовательской системе измерения, в пикселы. Пусть:

– Xmin, Xmax, Ymin, Ymax – минимальные и максимальные координаты, отображаемые на полотне для осей X и Y, соответственно. Данные представляются в условных единицах измерения длины: сантиметры, метры;

– MaxX, MaxY – размеры полотна по осям X и Y, соответственно в пикселах.

Тогда масштабирующие коэффициенты, показывающие, сколько пикселов приходится на единицу длины пользовательской системы измерения, по осям абсцисс и ординат вычисляются так:

$$Kx = \frac{MaxX}{Xmax - Xmin} \qquad Ky = \frac{MaxY}{Ymax - Ymin}$$

Таким образом, для некоторой точки с координатами Xi, Yi, выраженными в системе координат графика (сантиметры, метры, …), мы можем получить координаты Xp, Yp, которые отображают положение точки в пикселах полотна:

$$Xp = Kx \cdot Xi \; ; \qquad Yp = MaxY - Ky \cdot Yi \; .$$

Во втором выражении учтён, тот факт, что ось ординат в системе координат полотна направлена вниз.

Напишем скрипт, который покажет, как используется виджет Canvas.

```
from tkinter import *  # Импорт модуля
root = Tk() # Создание главного окна
root.title("Графика") # Надпись в заголовке окна
root.resizable(False, False) # Не менять размер окна
                             # по ширине и высоте
Kp = 0.7 # используем 70% экрана
MaxX = root.winfo_screenwidth()* Kp  # Текущие размеры
MaxY = root.winfo_screenheight()* Kp # дисплея монитора
                                     # в пикселах
cv = Canvas(root, width = MaxX, height = MaxY,
            bg = "white")
cv.grid(row = 0, columnspan = 11)
```

В первой строке скрипта выполняется импорт модуля Tkinter.

Обратите внимание на то, что имя модуля записано с маленькой буквы. Затем создаётся главное окно скрипта, и выводится его название в заголовке формы.

Изменение размеров окна потребует дополнительного внимания и обработки, например, перерисовки графиков. Снимем эту проблему, запретив изменение размеров окна. Для этого используется метод:

```
resizable(bW, bH),
```

где параметры `bW` – изменение размера по ширине, и bH – изменение размера по высоте, принимают логическое значение `False` или `True`.

Через методы главного окна можно получить размеры дисплея монитора.

```
winfo_screenwidth()     # Ширина в пикселах
winfo_screenheight()    # Высота в пикселах
```

Эти размеры сильно отличаться для разных мониторов, поэтому зададим размеры полотна на 70% меньше размеров дисплея.

Привязка виджета к форме выполняется при его описании. При этом формируется идентификатор виджета, значение которого может быть присвоено переменной. Обратите внимание на строку, описывающую виджет Canvas:

```
cv = Canvas(root,...)
```

`root` – имя главного окна, `cv` – имя полотна, которое мы будем указывать при рисовании примитивов. Заметим, что на форме может быть создано несколько полотен.

Упаковка виджетов, в нашем примере, будет выполняться методом `grid()`. В качестве аргументов мы передаём номер строки, на которой будет размещаться полотно, и сколько столбцов оно будет занимать. Мы указали нулевую строку, поскольку виджет будет размещаться в верхней части формы. Количество занимаемых столбцов указано 11. Обратитесь к рис.8.2 и посчитайте, сколько виджетов будет размещено по горизонтали в первой строке, если каждый виджет будет занимать одну ячейку воображаемой таблицы. Заметим, что нумерация строк и столбцов начинается с нуля и первая строка занята полотном.

Создайте скрипт с указанным текстом и посмотрите на результат.

Добавим в нашу форму метки (`Label`), которые мы нарисовали на нашем макете, рис.8.3.

Для описания меток используется метод `Label()`, в аргументах которого надо указывать имя окна, к которому относится метка. Дополнительно можно указать текст, ширину окна, в котором надо поместить текст, параметры текста (цвет, тип шрифта и его размер), а так же параметры выравнивания текста (по левому краю, по центру, по правому краю). Значение выравнивания задаётся либо словом CENTER – центр, либо символом или комбинацией символов. Символы – это первые буквы

английских слов, описывающие стороны света: **N**orth – Север, **E**ast – Восток, **S**outh – Юг и **W**est – Запад, рис.8.4.

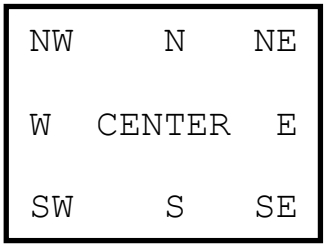

Рис.8.4. – Направления выравнивания текста

Идентификационный код объекта, который возвращается методом `Label()`, присвоим переменным. Опишем все восемь меток нашего макета.

```
lba0 = Label(root, text = "X:", width = 10,
             fg = "blue", font = "Ubutu, 12")
lba0.grid(row = 1, column = 0, sticky='e')
lba1 = Label(root, text = "Y:", width = 10,
             fg = "blue", font = "Ubutu, 12")
lba1.grid(row = 2, column = 0, sticky='e')
lba2 = Label(root, text = "Xmin:", width = 10,
             fg = "blue", font = "Ubutu, 12")
lba2.grid(row = 1, column = 2, sticky='e')
lba3 = Label(root, text = "Xmax:", width = 10,
             fg = "blue", font = "Ubutu, 12")
lba3.grid(row = 1, column = 4, sticky='e')
lba4 = Label(root, text = "Ymin:", width = 10,
             fg = "blue", font = "Ubutu, 12")
lba4.grid(row = 2, column = 2, sticky='e')
lba5 = Label(root, text = "Ymax:", width = 10,
             fg = "blue", font = "Ubutu, 12")
lba5.grid(row = 2, column = 4, sticky='e')
lba6 = Label(root, text = "Шаг меток:", width = 10,
             fg = "blue", font = "Ubutu, 12")
lba6.grid(row = 1, column = 6, sticky='e')
lba7 = Label(root, text = "Смещение:", width = 10,
             fg = "blue", font = "Ubutu, 12")
lba7.grid(row = 2, column = 6, sticky='e')
```

Места размещения меток, указанные в упаковщике `grid()`, были вычислены в соответствии с ранее разработанным макетом, см. рис.8.2. Добавьте этот код к предыдущему и посмотрите результат.

Для завершения формы необходимо добавить поля для ввода и вывода данных – виджеты `Entry`. Эти поля описываются в полной аналогии с описанием меток, например:

```
ent0 = Entry(root, width = 5, font = "Ubuntu, 12")
ent0.grid(row = 1, column = 1)
```

Вставить значение строкового типа в поле `Entry` можно методом `insert()`:

```
ent0.insert(index, s),
```

где строка `S` вставляется перед символом, позиция которого задается параметром `index`.

Удалить символы из виджета можно методом `delete()`:

```
ent0.delete(first, last=None),
```

где параметр `first` задаёт начальную позицию удаляемых символов, а параметр `last` последнюю. При этом позиция, заданная в `last` не включается в удаляемый диапазон. Если `last` не задан, то удаляется только один символ. Если вторым параметром задано значением `END`, то удаляются все сиволы.

Прочитать значение, введённое пользователем в поле, можно методом `get()`:

```
var = float(ent0.get())
```

Прочитанное значение имеет строковый тип. Таким образом, при вставке числового значения его необходимо преобразовать к строковому типу `str()`. При чтении из поля `Entry` числового значения необходимо выполнить обратное преобразование к ожидаемому типу. В нашем скрипте мы будем использовать вещественные значения: `float()`.

Задание:

Добавьте виджеты `Entry` самостоятельно. Их должно быть восемь. Имена переменных, с помощью которых можно обращаться к этим виджетам, составьте по тому же принципу, что и имена для меток. Т.е. `entN`, где `N` – номер виджета.

Два первых поля (`ent0` и `ent1`) предназначены для вывода координат мыши. Инициализируйте их нулевым значением, а оставшиеся поля, предназначенные для ввода, инициализируйте начальными значениями:

```
Xmin = -10.0; Xmax = 10.0 # Границы по абсциссе
Ymin = -5.0; Ymax = 5.0   # Границы по ординате
dX = 50  # Шаг меток по осям
dY = 1.0 # Смещение графика
```

Замечание: Некоторые переменные, использованные в нашем скрипте, будут глобальными, что позволит использовать их в различных функциях без процедуры передачи. Мы поступили так, что бы сократить число параметров, передаваемых в функции.

Назначение переменных глобальными должно делаться более обоснованно.

Проверьте работу скрипта.

Теперь можно добавить кнопки (`Button`). При описании этих виджетов можно задать надпись, которая будет размещена на кнопке, параметры текста (шрифт, цвет) и что очень важно, привязать нажатие

кнопки к вызову функции, которая получит событие от нажатия кнопки и обработает его.

```
btn1 = Button(root, width = 20, bg = "#ccc",
              text = "Рисовать")
btn1.grid(row = 1, column = 10)
btn1.bind("<Button-1>", Draw)
```

В первой строке указан идентификатор кнопки (`btn1`), параметр root показывает, что кнопка находится на форме, остальные параметры задают размер, цвет фона и текст. Метод `grid()` поместит кнопку в первой строке в десятой колонке.

Для обработки события, связанного с кнопкой, будет вызываться функция `Draw()`. Обратите внимание, что переменная `Draw` записана без круглых скобок.

Код для второй кнопки будет аналогичен:

```
btn2 = Button(root, width = 20, bg = "#ccc",
              text = "Выход")
btn2.grid(row = 2, column = 10)
btn2.bind("<Button-1>", Final)
```

Перед тем, как перейти к описанию функций (головной части), добавим связку между событием "клик кнопкой мыши на полотне" с соответствующим обработчиком. Поскольку идентификатор полотна у нас имеет имя `cv`, то связующая строка примет вид:

```
cv.bind('<Button-1>', showXY),
```

где `showXY` функция, которая будет вызвана по событию `'<Button-1>'`. При желании можно в качестве события выбрать двойной клик кнопкой мыши. В этом случае нам следует указать код `<Double-Button-1>`.

Прочитайте больше о событиях и связывании событий в Интернете.

Заключительный шаг для формирования окна скрипта – это организация непрерывного цикла, в котором будут обрабатываться события, см. рис.8.1.

Для организации цикла, в котором будет находиться скрипт в ожидании событий, используется метод `mainloop()`:

```
root.mainloop()
```

Основные шаги по подготовке окна выполнены. В процессе написания скрипта созданы три условия для возникновения событий: кнопка "Рисовать", кнопка "Выход" и событие, связанное с кликом левой кнопки мыши. Реализуем обработку этих событий в виде функций.

```
def Drav(event):
    '''Вызов функций для рисования графиков'''
    pass
#
```

```
def Final(event):
    '''Завершение работы'''
    window_deleted()
#
def window_deleted():
    ''' Завершаем работу по [X]'''
    if askyesno("Выход", "Завершить работу?"):
        root.destroy()
#
def showXY(event):
   '''Вывод координат мыши'''
    global ID1, ID2
    x = event.x; y = event.y
    ent0.delete(0,END)  # Удалим старые
    ent1.delete(0,END)  # значения
# Вычислим и введём новые
    ent0.insert(0, str(round(Xmin + x/Kx, 2)))
    ent1.insert(0, str(round(Ymin + (MaxY - y)/Ky, 2)))
    cv.delete(ID1)  # Удалим старые
    cv.delete(ID2)  # линии
# Нарисуем новые линии
    ID1 = cv.create_line(0,y,MaxX,y, dash = (3,5))
    ID2 = cv.create_line(x,0,x,MaxY, dash = (3,5))
```

Отметте, что в качестве параметра функций указана переменная `event`. Через эту переменную в функцию передаётся описание события.

Функция `Draw()` будет вызывать другие функции. Они нами ещё не разработаны, и мы используем инструкцию pass. Это позволит тестировать скрипт раньше.

Функция `Final()` должна сработать по нажатию кнопки "Выход" и завершить работу скрипта. Поскольку завершение работы скрипта может быть при случайном клике, создадим окно, в котором будет запрос на подтверждение закрытия и две кнопки "Да" и "Нет".

В Tkinter имеется модуль `tkinter.messagebox`, в котором имеется набор окон с разной функциональной направленностью. В нашем случае это:

```
askyesno("Выход", "Завершить работу?")
```

Здесь `askyesno()` метод, формирующий окно запроса с двумя кнопками. В качестве параметров указываются строки. Первая строка – заголовок окна, а вторая – сообщение, которое появится в окне. При нажатии на кнопку возвращается логическое значение: True – нажата кнопка "Да" и False – нажата кнопка "Нет".

Существует ещё один путь закрытия окна – это кнопка [X] в правом верхнем углу формы. Для обработки этого события используется метод `protocol()`, который перехватывает событие, связанное с закрытием окна по кнопкой [X]:

```
root.protocol('WM_DELETE_WINDOW', window_deleted)
```

Здесь `'WM_DELETE_WINDOW'` – наименование протокола, связанного с кнопкой закрытия окна ([X]), а `window_deleted` – функция, обрабатывающая событие. Отметим, что при обработке этого события его параметры в функцию не передаются.

Оформим сказанное в виде двух функций. Одна вызывается при закрытии окна по кнопке [X] и выводит запрос на подтверждение, а вторая вызывает первую. Само закрытие окна происходит при вызове метода `destroy()`:

```
root.destroy()
```

Функция `showXY()` вызывается по событию "одиночный клик левой кнопкой мыши" на полотне и выводит координаты точки на полотне, в которой был сделан клик, в поля `Entry`. Реализация этой функции усложнена по следующим соображениям:

– координаты точки – это два числа отображающие положение точки в пикселах в системе координат полотна. Их необходимо преобразовать в систему координат пользователя. Для этого используем знание о масштабе по каждой из осей. Результат преобразования чисел будет вещественным, поэтому округлим их до двух цифр после запятой. Перед выводом в поле `Entry` числа необходимо преобразовать в строковый тип;

– точку, в которой произведён клик, надо визуализировать. Для этого нарисуем две перпендикулярные пунктирные линии, которые будут проходить через заданную точку параллельно координатным осям. Метод, используемы при рисовании линий, возвращает идентификатор, который мы используем для удаления линий перед новой прорисовкой графика.

Обратитесь к коду функций, приведённому выше, для более полного усвоения этого материала. Заметим, что метод `cv.delete(ID1)` не вызывает ошибки в том случае, когда `ID` не определён: перед первой прорисовкой мы удаляем несуществующий объект.

Координатные оси

Для оценки значений, которые принимает функция, на полотне рисуются координатные линейки с разметкой. Функция `plotXY()` рисует линейки по краям полотна.

Для прорисовки координатной линейки вначале рисуется прямоугольник, а затем на его сторонах наносятся метки в виде коротких линий и текстовые значения, соответствующие этим меткам. При нанесении значения выполняется его вычисление, округление до двух знаков после запятой, а затем преобразование в строковый тип.

В дополнение к этому мы реализовали метод считывания координат полотна по событию, связанному с кликом левой кнопкой мыши.

Рисование функций

Для рисования функций вычисляются значений функции в диапазоне от `Xmin` до `Xmax`. При этом используется пользовательская система координат. Значения абсциссы и ординаты объединяются и формируют список, который возвращается при вызове функции. Поскольку в задании требуется изобразить переходы между вычисляемыми точками функции плавными, вычисления выполняются с шагом в один пиксел по оси X. В пользовательской системе координат шаг будет равен `1/Kx`, где `Kx` – масштабный коэффициент по оси X.

Рисование функций начинается с появления события, связанного с кликом по кнопке "Рисовать". При этом вызывается функция Drav(), в которой последовательно выполняются следующие операции:

Очистка полотна от ранее нарисованных объектов;

Получение данных из полей Entry. Данные, при считывании, контролируются на допустимый диапазон значений. Если на этом шаге возникает ошибка, то выводится предупреждающее сообщение и пользователь может ввести новые данные. Работа скрипта не прерывается;

Рисуется координатная линейка по краям полотна;

Вызывается функция, прорисовывающая график. При вызове, в качестве аргументов задаётся имя функции, которую необходимо нарисовать и цвет, которым будет выполнен график.

Листинг скрипта

Здесь представлен листинг скрипта, решающий поставленную задачу. Он вполне рабочий, но может быть существенно улучшен, например, тем, что основную часть скрипта можно оформить в виде отдельного модуля. Пользователь сможет использовать такой модуль для отображения и других функций в графической форме. При этом ему достаточно будет подготовить собственный скрипт, в который может быть импортирован наш модуль и описаны функции, для которых нужно построить график.

Но эта тема здесь не развивается. Студентам предлагается самостоятельно подумать над возможными улучшениями скрипта.

```
from tkinter import *
from tkinter.messagebox import *
from math import sin, cos, pi, exp
#
def Sin2xDevx(): # Функция из задания
    eps = 0.0001
    Lfun = []
    x = Xmin
    while x <=Xmax:
        an = 1
        Sum = an
        n = 1
```

```
        while (abs(an) > eps):
            an *= -(2**2)*(x**2)/((2*n)*(2*n+1))
            Sum += an
            n += 1
        Lfun.append((x, 2*Sum))
        x += 1 / Kx
    return Lfun

def Sin2xOnX():  # Функция из задания
    Lfun = []
    x = Xmin
    while x <= Xmax:
        if x != 0:
            fun = sin(2*x)/x + dY
        else:
            fun = 2.0
        Lfun.append((x, fun))
        x += 1 / Kx
    return Lfun

def GetData():
    '''Получить данные'''
    global Xmax, Xmin, Ymax, Ymin
    global dX, dY, Kx, Ky
    tmpXmax = float(ent3.get())
    tmpXmin = float(ent2.get())
    tmpYmax = float(ent5.get())
    tmpYmin = float(ent4.get())
    tmpdY = float(ent7.get())
    tmpdX = float(ent6.get())
    if ((tmpXmin >= tmpXmax) or
        (tmpYmin >= tmpYmax) or
        (tmpdX <= 0)):
        showwarning(title = "Ошибка задания границ",
                    message = "Должны выполняться "
                              " неравенства:\n"
                              "Xmax > Xmin;\n"
                              "Ymax > Ymin;\n"
                              "Шаг меток > 2")
    else:
        Xmax = tmpXmax
        Xmin = tmpXmin
        Ymax = tmpYmax
        Ymin = tmpYmin
        dX = tmpdX
        dY = tmpdY
```

```
        Kx = MaxX / abs((Xmax - Xmin))
        Ky = MaxY / abs((Ymax - Ymin))
def SetMark(a, b, LrBt = 1):
    '''Нанесение меток'''
    ax_XY = []
    if LrBt:  # слева и справа
        ax_XY.append((a, b))
        ax_XY.append((a + 10, b))
    else:     # внизу и вверху
        ax_XY.append((a,b))
        ax_XY.append((a, b - 10))
    cv.create_line(ax_XY, fill='black', width = 2)
def plotXY():
    ''' Рисуем координатные линейки'''
# Прямоугольник
    ax_XY = []
    ax_XY.append((5, 5))
    ax_XY.append((MaxX - 5, MaxY - 5))
    cv.create_rectangle(ax_XY, fill="white",
                        outline = "green", width = 2)
#
# Разметка левой и правой сторон
    y = Ymin
    y_pix = MaxY
    flg = False
    while y < Ymax:
        textY = str(round(y, 2))  # Текст метки
        SetMark(0, y_pix, 1)      # Слева
        if flg:   # Надписываем каждую вторую метку
            cv.create_text(15, y_pix, text = textY,
                           anchor = W)
        SetMark(MaxX-10, y_pix, 1) # Справа
        if flg:
            cv.create_text(MaxX - 15, y_pix,
                           text = textY, anchor = E)
        y += dX                    # Ед. пользователя
        y_pix -= dX * Ky           # Ед. в пикселах
        flg = not flg              # Разметка через раз
#
# Разметка сверху и снизу
    x = Xmin
    x_pix = 0
    flg = False
    while x < Xmax:
```

```
        textX = str(round(x, 2))  # Текст метки
        SetMark(x_pix, 0, 0)      # Вверху
        if flg:
            cv.create_text(x_pix, 15, text = textX,
                           anchor = N)
        SetMark(x_pix, MaxY, 0)   # Внизу
        if flg:
            cv.create_text(x_pix, MaxY - 15,
                           text = textX, anchor = S)
        x += dX
        x_pix += dX*Kx
        flg = not flg

def Draw(event):
    '''Подготовка полотна и вызов
       функций для рисования'''
    cv.delete("all") # Очистка полотна
    GetData()        # Получить данные
    plotXY()         # Нарисовать координатные линейки
    Fdraw(Sin2xDevx, 'blue') # Нарисовать функцию 1
    Fdraw(Sin2xOnX, 'red')   # Нарисовать функцию 2
    print('Рисуем')          # Тестовое сообщение

def Fdraw(func, color):
    '''Получение значений функции
       Преобразование в пикселы
       Рисование на полотне'''
    Lxy = func() # Значения функции в ед. пользователя
    Lpix = []    # Значения функции в пикселах
    for xy in Lxy:
        x = Kx * (xy[0] - Xmin)
        if xy[1] != None:
            y = MaxY - Ky * (xy[1] - Ymin)
        else:
            y = 0
        Lpix.append((x,y))
    cv.create_line(Lpix, fill = color)

def Final(event):
    ''' Завершаем работу '''
    window_deleted()

def window_deleted():
    ''' Завершаем работу по [X]'''
    if askyesno("Выход", "Завершить работу?"):
        root.destroy()

def showXY(event):
```

```
    global ID1, ID2
    x = event.x; y = event.y
    ent0.delete(0,END)
    ent1.delete(0,END)
    ent0.insert(0, str(round(Xmin + x/Kx, 2)))
    ent1.insert(0, str(round(Ymin + (MaxY - y)/Ky, 2)))
    cv.delete(ID1)
    cv.delete(ID2)
    ID1 = cv.create_line(0,y,MaxX,y, dash = (3,5))
    ID2 = cv.create_line(x,0,x,MaxY, dash = (3,5))
#
root = Tk()
root.title("Графика")
# Обработчик закрытия окна. Нажата кнопка [X]
root.protocol('WM_DELETE_WINDOW', window_deleted)
root.resizable(False, False)   # Не меняем размер окна
#
Kp = 0.7 # 70% от дисплея
# Начальные параметры полотна. Ед. изм. - пикселы
MaxX = root.winfo_screenwidth() * Kp
MaxY = root.winfo_screenheight() * Kp
#
cv = Canvas(root, width = MaxX,
            height = MaxY, bg = "white")
cv.grid(row = 0, columnspan = 9)
cv.bind('<Button-1>', showXY) # Клик на полотне
#
# Окно графика. Ед. изм. - пользовательские
Xmin = -10.0; Xmax = 10.0
Ymin = -5.0; Ymax = 5.0
Xmid = 0; Ymid = 0 # Положение начала координат
#
# Смещение и шаг аргумента. Ед. изм. - пользовательские
dY = 1.0   # смещение второго графика
dX = 1.0   # шаг меток на координатных линейках
# Идентификаторы курсоров, рисуемых по клику на полотне
ID1 = 0; ID2 = 0
# Масштабные коэфициены: Пиксел / пользов.ед.
Kx = MaxX / abs((Xmax - Xmin))
Ky = MaxY / abs((Ymax - Ymin))
#
lba0 = Label(root, text = "X:", width = 10,
             fg = "blue", font = "Ubutu, 12")
lba0.grid(row = 1, column = 0, sticky='e')
ent0 = Entry(root, width = 5, font = "Ubuntu, 12")
```

```
ent0.grid(row = 1, column = 1, sticky='w')
ent0.insert(0, 0)
lba1 = Label(root, text = "Y:", width = 10,
             fg = "blue", font = "Ubutu, 12")
lba1.grid(row = 2, column = 0, sticky='e')
ent1 = Entry(root, width = 5, font = "Ubuntu, 12")
ent1.grid(row = 2, column = 1, sticky='w')
ent1.insert(0, 0)

lba2 = Label(root, text = "Xmin:", width = 10,
             fg = "blue", font = "Ubutu, 12")
lba2.grid(row = 1, column = 2, sticky='e')
ent2 = Entry(root, width = 5, font = "Ubuntu, 12")
ent2.grid(row = 1, column = 3)
ent2.insert(0, Xmin)

lba3 = Label(root, text = "Xmax:", width = 10,
             fg = "blue", font = "Ubutu, 12")
lba3.grid(row = 1, column = 4, sticky='e')
ent3 = Entry(root, width = 5, font = "Ubuntu, 12")
ent3.grid(row = 1, column = 5)
ent3.insert(0, Xmax)

lba4 = Label(root, text = "Ymin:", width = 10,
             fg = "blue", font = "Ubutu, 12")
lba4.grid(row = 2, column = 2, sticky='e')
ent4 = Entry(root, width = 5, font = "Ubuntu, 12")
ent4.grid(row = 2, column = 3)
ent4.insert(0, Ymin)

lba5 = Label(root, text = "Ymax:", width = 10,
             fg = "blue", font = "Ubutu, 12")
lba5.grid(row = 2, column = 4, sticky='e')
ent5 = Entry(root, width = 5, font = "Ubuntu, 12")
ent5.grid(row = 2, column = 5)
ent5.insert(0, Ymax)

lba6 = Label(root, text = "Шаг меток:", width = 10,
             fg = "blue", font = "Ubutu, 12")
lba6.grid(row = 1, column = 6, sticky='e')
ent6 = Entry(root, width = 5, font = "Ubuntu, 12")
ent6.grid(row = 1, column = 7)
ent6.insert(0, dX)

lba7 = Label(root, text = "Смещение:", width = 10,
```

```
                fg = "blue", font = "Ubutu, 12")
lba7.grid(row = 2, column = 6, sticky='e')
ent7 = Entry(root, width = 5, font = "Ubuntu, 12")
ent7.grid(row = 2, column = 7)
ent7.insert(0, dY)

btn1 = Button(root, width = 20, bg = "#ccc",
              text = "Рисовать")
btn1.grid(row = 1, column = 8)
btn1.bind("<Button-1>", Draw)

btn2 = Button(root, width = 20, bg = "#ccc",
              text = "Выход")
btn2.grid(row = 2, column = 8)
btn2.bind("<Button-1>", Final)

root.mainloop()  # Цикл ожидания событий
```

Результат работы скрипта

На рис.8.5 представлена реализованная форма с графиками.

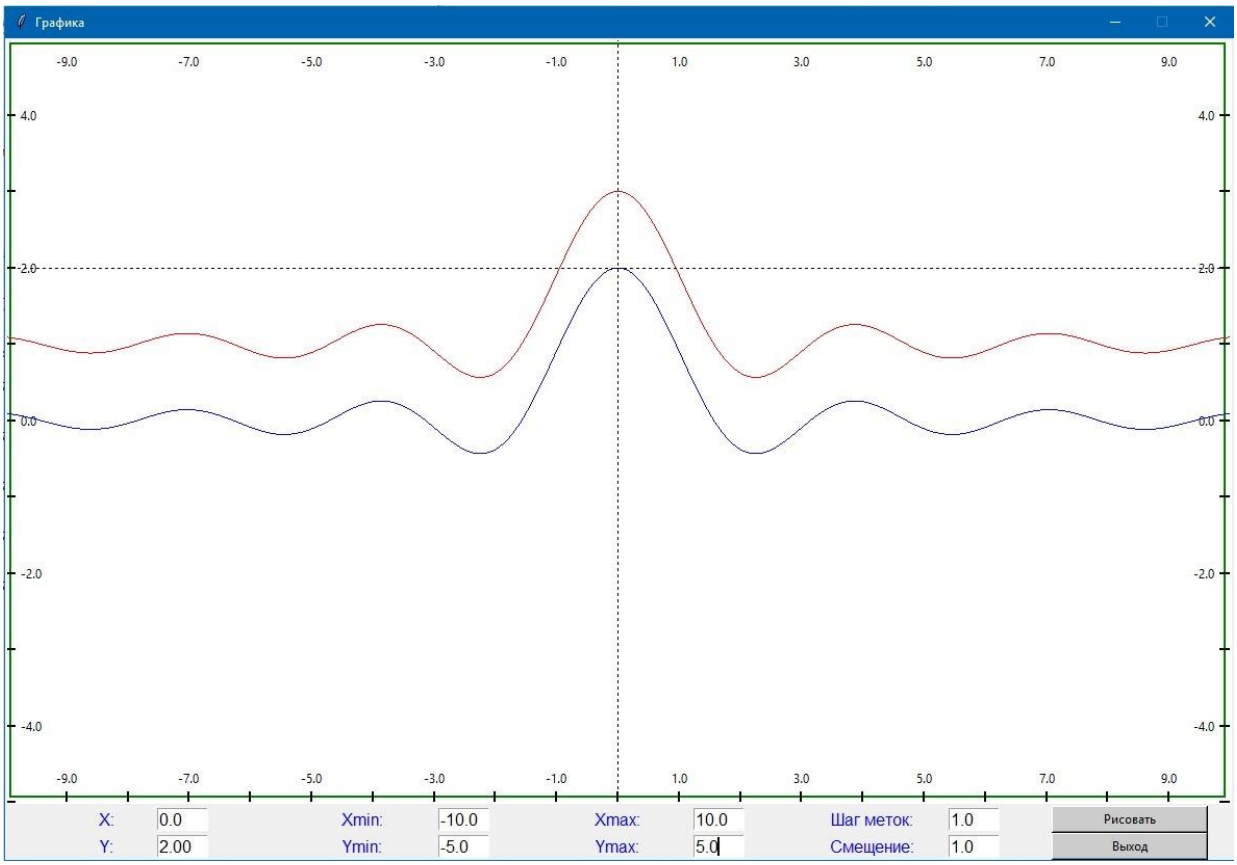

Рис.8.5. – Результат работы скрипта

Задание к лабораторной работе №8
«GUI, модуль Tkinter»

Изображение должно занимать большую часть экрана, сопровождаться заголовком, содержать наименования и градации осей и масштабироваться в зависимости от исходных данных. При любых допустимых значениях исходных данных изображение должно полностью помещаться на экране. Программа не должна опираться на конкретные значения разрешения экрана.

Вывести на экран в графическом режиме графики двух функций на интервале от `Хнач` до `Хкон` с шагом `dx`. Первая функция задана с помощью ряда Тейлора, ее вычисление должно выполняться с точностью ε. Значение параметра `b` для второй функции вводится с клавиатуры. Графики должны быть плавными и различаться цветами.

В вариантах с 12 по 20 требуется построить графическое изображение в виде секторных или столбцовых диаграмм. Для решения этих задач необходимо познакомиться с виджетами модуля Tkinter полнее, чем это изложено выше.

Вариант 1

$$y(x) = 2 \cdot \sum_{n=0}^{\infty} \frac{1}{(2 \cdot n + 1) \cdot x^{2 \cdot n + 1}} = 2 \cdot \left(\frac{1}{x} + \frac{1}{3 \cdot x^3} + \frac{1}{5 \cdot x^5} + \ldots \right), \qquad |x| > 1;$$

$$z(x) = \ln \frac{x+1}{x-1} + b.$$

Вариант 2

$$y(x) = \sum_{n=0}^{\infty} \frac{(-1)^n \cdot x^n}{n!} = \left(1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - \ldots \right), \qquad |x| < \infty;$$

$$z(x) = e^{-x} + b;$$

Вариант 3

$$y(x) = \sum_{n=0}^{\infty} \frac{x^n}{n!} = \left(1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots \right), \qquad |x| < \infty$$

$$z(x) = e^x + b.$$

Вариант 4

$$y(x) = \frac{\pi}{2} + \sum_{n=0}^{\infty} \frac{(-1)^{n+1}}{(2 \cdot n + 1) \cdot x^{2 \cdot n + 1}} = \frac{\pi}{2} - \frac{1}{x} + \frac{1}{3 \cdot x^3} - \frac{1}{5 \cdot x^5} + \ldots, \qquad x > 1;$$

$$z(x) = arctgx + b.$$

Вариант 5

$$y(x) = \sum_{n=0}^{\infty} \frac{(-1)^n \cdot x^{2 \cdot n + 1}}{(2 \cdot n + 1)} = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \ldots, \qquad |x| <= 1$$

$$z(x) = arctgx + b.$$

Вариант 6

$$y(x) = 2 \cdot \sum_{n=0}^{\infty} \frac{1}{(2 \cdot n + 1) \cdot x^{2 \cdot n + 1}} = 2 \cdot \left(\frac{1}{x} + \frac{1}{3 \cdot x^3} + \frac{1}{5 \cdot x^5} + \ldots \right), \qquad |x| > 1;$$

$$z(x) = Arthx + b.$$

Вариант 7

$$y(x) = -\frac{\pi}{2} + \sum_{n=0}^{\infty} \frac{(-1)^{n+1}}{(2 \cdot n + 1) \cdot x^{2 \cdot n + 1}} = -\frac{\pi}{2} - \frac{1}{x} + \frac{1}{3 \cdot x^3} - \frac{1}{5 \cdot x^5} + \ldots, \qquad x < -1$$

$$z(x) = arctgx + b.$$

Вариант 8

$$y(x) = \sum_{n=0}^{\infty} \frac{(-1)^n \cdot x^{2 \cdot n}}{n!} = 1 - x^2 + \frac{x^4}{2!} - \frac{x^6}{3!} + \frac{x^8}{4!} - \ldots, \qquad |x| < \infty$$

$$z(x) = e^{-x^2} + b.$$

Вариант 9

$$y(x) = \sum_{n=0}^{\infty} \frac{(-1)^n \cdot x^n}{(2 \cdot n)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6} + \ldots, \qquad |x| < \infty$$

$$z(x) = \cos x + b.$$

Вариант 10

$$y(x) = \sum_{n=0}^{\infty} \frac{(-1)^n \cdot x^{2 \cdot n}}{(2 \cdot n + 1)!} = 1 - \frac{x^2}{3!} + \frac{x^4}{5!} - \frac{x^6}{7!} + \ldots, \qquad |x| < \infty$$

$$z(x) = \frac{\sin x}{x} + b.$$

Вариант 11

$$y(x) = 2 \cdot \sum_{n=0}^{\infty} \frac{(x-1)^{2 \cdot n + 1}}{(2 \cdot n + 1) \cdot (x+1)^{2 \cdot n + 1}} = 2 \cdot \left(\frac{x-1}{x+1} + \frac{(x-1)^3}{3 \cdot (x+1)^3} + \frac{(x-1)^5}{5 \cdot (x+1)^5} + \ldots \right), \qquad x > 0;$$

$$z(x) = \ln x + b.$$

Вариант 12

Написать программу, которая выводит на экран секторную диаграмму. Диаграмму снабдить заголовком и наименованием для каждого сектора. Исходные данные сформировать в текстовом файле. Количество секторов задавать в программе в виде именованной константы.

Построение секторной диаграммы оформить в виде процедуры. Параметры процедуры: координаты центра диаграммы; радиус; количество секторов; массив процентов; массив наименований. Пример исходных данных см. Таблица 1.

Вариант 13

Написать программу, которая выводит на экран две секторные диаграммы, расположив их рядом. Диаграмму снабдить заголовком и наименованием для каждого сектора. Исходные данные сформировать в текстовом файле. Количество секторов задавать в программе в виде именованной константы.

Построение секторной диаграммы оформить в виде процедуры. Параметры процедуры: координаты центра диаграммы; радиус; количество секторов; массив процентов; массив наименований. Пример исходных данных см. Таблица 1.

Вариант 14

Написать программу, которая выводит на экран две столбиковые диаграммы. На экране диаграммы расположить рядом, каждую в своих координатных осях. Каждую диаграмму снабдить заголовком и наименованием единиц измерений по осям X и Y. Исходные данные сформировать в текстовом файле. Количество столбцов задавать в программе в виде именованной константы. Построение диаграммы оформить в виде процедуры. Пример исходных данных см. Таблица 1.

Вариант 15

Написать программу, которая выводит на экран две столбиковые диаграммы в одной координатной плоскости. Диаграмму снабдить градацией осей и заголовком. Исходные данные сформировать в текстовом файле. Количество столбцов задавать в программе в виде именованной константы. Построение диаграммы оформить в виде процедуры. Пример исходных данных см. Таблица 1.

Вариант 16

Написать программу, которая выводит на экран трехмерную столбиковую диаграмму. Диаграмму снабдить градацией осей и заголовком. Исходные данные сформировать в текстовом файле. Количество столбцов задавать в программе в виде именованной константы. Построение диаграммы оформить в виде процедуры. Пример исходных данных см. Таблица 1.

Вариант 17

Написать программу, которая выводит на экран столбиковую диаграмму, представляющую оптовые и розничные цены на различные наименования кофе. Исходные данные сформировать в текстовом файле.

Построение диаграммы оформить в виде процедуры. Параметры процедуры: количество наименований; массив значений оптовых цен; массив значений розничных цен; массив наименований. Наименования товаров разместить вертикально под осью абсцисс.

Вариант 18

Написать программу, которая выводит на экран столбиковую диаграмму, представляющую максимальную и среднюю норму прибыли при реализации различных сортов шоколада. Исходные данные сформировать в текстовом файле самостоятельно.

Построение диаграммы оформить в виде процедуры. Параметры процедуры: количество наименований; массив значений оптовых цен; массив значений розничных цен; массив наименований. Наименования товаров разместить вертикально под осью абсцисс.

Вариант 19

Написать программу, которая выводит на экран графики динамики изменения максимального, минимального и среднего курса доллара за заданное количество дней. Исходные данные сформировать в текстовом файле самостоятельно.

Построение графика оформить в виде процедуры. Параметры процедуры: массив дат; количество дней; массивы максимальных, минимальных и средних значений.

Вариант 20

Написать программу, которая выводит на экран трехмерную столбиковую диаграмму курса немецкой марки по отношению к рублю за заданное количество дней. Исходные данные сформировать в текстовом файле самостоятельно.

Построение диаграммы оформить в виде процедуры. Параметры процедуры: массив дат; количество дней; массив значений по оси Y; код заполнителя.

Пример исходных данных для вариантов 12 – 16

Таблица 1. – Лидеры мирового рынка ПК

Рейтинг 1996 г.	Поставщик	Объем Продаж 1996 г. тыс. шт.	Доля Рынка 1996 г., %	Объем Продаж 1995 г. тыс. шт.	Доля Рынка 1995 г., %	Рост 95/96, %
1	Compaq	7 036	10,3	5 757	9,8	22
2	IBM	6 081	8,9	4 785	8,1	27
3	Packard	4 247	6,2	4 392	7,5	-3
	Bell, NEC	3 587	5,2	4 627	7,9	22
4	Apple	2 995	4,4	2 023	3,4	48
5	HP	44 459	65,0	37 221	63,3	19
6	Другие					

Примечание: Попробуйте обновить информацию, получив необходимые данные из сети Интернет.