Лабораторная работа №7: «Программирование графики». Модуль turtle

Цель работы:

Познакомить студентом на практике с написанием программ для формирования графических изображений с использованием модулей Python.

Постановка задачи

Используя программный код из лабораторной работы №2 для Задания 1 и Задания 2, а так же один из графических модулей Python, написать скрипты, которые иллюстрируют работу кода в графическом виде:

– Задание 1 – Решить обратную задачу – построить график функции, используя программный код, написанный к этому заданию;

– Задание 2 – построить графическое изображение, которое получается в результате генерации точек со случайными координатами. Использовать до 10000 испытаний. Оценить площадь заштрихованной части фигуры методом Монте-Карло и сравнить с реальной площадью. Оценку вывести в процентах по отношению к реальной площади.

Теоретическое введение

В Python разработано несколько модулей, обеспечивающих работу с графикой. Два графических модуля являются частью стандартной библиотеки Python:

– `turtle` (черепашка) – простой графический пакет, который так же может быть использован для создания несложного пользовательского графического интерфейса – Graphical User Interface (GUI);

– `tkinter` – разработан непосредственно для создания графического интерфейса пользователя GUI. Так, интерфейс IDLE построен с использованием `tkinter`. В этом модуле имеется виджет Canvas, позволяющий рисовать графические изображения.

Другой путь написания графических приложений – это использование кроссплатформенных графических библиотек. Одной из кроссплатформенных графических библиотек является Qt. Эта библиотека используется с такими языками, как C++, Java, Ruby, Delphi, Lazarus, и др. У нее имеется "привязка" и к Python – PyQt.

Компьютерная графика – это довольно обширная и сложная область знания. Она включает знания о технических средствах, позволяющих отображать изображение: растровые и векторные дисплеи, плоттеры и принтеры. Знание о способах формирования цветного изображения, которое воспринимается либо как свечение отдельных точек дисплея, либо как отражение, например, от листа бумаги: аддитивная цветовая модель RGB или субтрактивная - CMY. Обширная область математических и физических знаний помогает решать вопросы перемещения, поворота объекта,

формирования второго и первого планов изображения (сцены), учета рефлексов (вторичных отражений) и еще много чего.

В этом пособии рассмотрен функционал графических модулей, который позволяет рисовать графики функций.

При построении графиков нам потребуются знания о функциях и методах графической системы, которые позволяют:

- формировать окно, в котором будет строиться график (размер, система координат);
- строить графические примитивы (точка, линия, …);
- задавать ширину и цвет линий, цвет фона, выполнять заливку изображения или его части заданным цветом;
- управлять маркером (указателем), который используется для рисования;
- наносить текст.

Модуль turtle

Модуль `turtle` входит в стандартную поставку библиотеки Python. Исходно этот модуль позиционируется как средство для обучения компьютерной графике детей.

Модель этого модуля следующая. Имеется прямоугольная поверхность, по которой ползает черепашка. Черепашка может перемещаться на заданное расстояние прямо, назад, под углом или по заданным координатам. Черепашку можно клонировать, создавая группу черепашек. При этом каждая черепашка живет своей жизнью. Для рисования черепашка использует цветовое перо (карандаш), которое может быть поднято или опущено. Если перо опущено, то остается след. Можно изменять цвет и толщину линии.

Черепашка понимает команды, с помощью которых можно нарисовать окружность заданного радиуса и цвета, дугу с заданным углом, залить фигуру определенным цветом, получить текущее состояние настроек или изменить их. Форма черепашки может быть изменена пользователем и использована как штамп, после которого на холсте остается рисунок.

В модуле `turtle` реализованы и интерактивные способы взаимодействия с черепашкой. События, связанные с кликом кнопки мыши или нажатия \ отпускания клавиши клавиатуры, могут обрабатываться пользовательскими функциями, привязанными к этим событиям.

ВНИМАНИЕ:

1. Язык Python и его модули находятся в стадии постоянного развития и модификации. При использовании модулей следует контролировать их обновления и появление обновленной документации.

2. С электронной версией пособия распространяется файл Turtle_3-6-1.pdf, в котором приводится перечень функций и методов модуля `turtle` на русском языке.

Знакомство с функциями модуля `turtle` начнем с решения конкретной задачи: построим график функции, которая задавалась в графическом виде в лабораторной работе №2 Задание 1 (решим обратную задачу).

Решение первой задачи

На первом шаге оформим программу, написанную в лабораторной работе №2, Задание 1 в виде функции, а затем добавим графическую часть.

При построении графика функции мы будем перебирать значения аргумента функции в том диапазоне, который задан рисунком. Поскольку процесс получения аргумента программный, возможны ситуации, в общем случае, когда для заданного аргумента значение функции не может быть вычислено и работа программы завершиться с ошибкой. Ошибка может быть вызвана, например, делением на ноль или получением квадратного корня из отрицательного числа.

Для исключения подобных ситуаций необходимо исследовать функцию и те точки, в которых она не может быть вычислена, следует исключить условными операторами. Примем, что функция для таких точек возвращает значение `None`.

У нашей функции особых точек нет, но, тем не менее, в листинге функции, приведенном ниже, вставлен дополнительный код. Этот код приведен для примера и закомментирован, но в последующем вы сможете проверить его работу.

```
def MyFun(x):
#    if (x >= 5) and (x <= 7):
#        return(None)
    if x < -5:
        y = 1
    elif x >=-5 and x<0:
        y = -(3/5)*x-2
    elif x >= 0 and x<2:
        y = -sqrt(4-x**2)
    elif x >= 2 and x<4:
        y = x-2
    elif x >= 4 and x<8:
        y = 2+sqrt(4-(x-6)**2)
    else: y = 2
    return(y)
```

Замечание: Параметры, необходимые для решения задания следует получить из графика и определить в программе, например, радиус дуги.

Графическая часть

При построении графика нам необходимо выполнить ряд условий и провести подготовительную работу. Прежде, чем перейти к написанию

графической части программы следует разобраться с графической системой. Необходимо ответить на вопрос: "Как это работает?"

Замечание: Под монитором тут понимается внешнее устройство, которое подключается к видеокарте системного блока. Дисплей – устройство, с помощью которого формируется изображение. Дисплей является составной частью монитора. Изображение составляется из квадратных элементов – пикселов (pixel) дисплея, а каждый пиксел состоит из трех прямоугольных элементов, позволяющих формировать цвет пиксела в формате RGB. Одной из технических характеристик дисплея является разрешение: число точек по горизонтали и по высоте. Например, 1680x1050.

Графическое изображение может создаваться на дисплее монитора, принтере или плоттере. Каждое из этих устройств обладает определенными функциональными возможностями. С целью построения инвариантной графической системы (device-independent graphics) используют определенные положения. Рассмотрим некоторые из них для плоского рисунка.

1. В многозадачной системе каждое приложение может иметь свое графическое окно, которое может занимать всю поверхность дисплея или его часть. Такое окно назовем главным окном (main window). Размеры окна и его положение задаются в качестве аргументов функции `setup()`:

```
turtle.setup(Dw, Dh, Xc, Yc),
```

где `Dw` и `Dh` размеры окна по горизонтали и вертикали, соответственно, а `Xc` и `Yc` – координаты окна на дисплее. Если значения координат положительные, то они определяют положение верхнего левого угла главного окна, а если отрицательные – нижнего правого угла. Начало системы координат будет находиться в центре окна. Единицей измерения является пиксел. Размеры окна можно менять путем перетаскивания его границ. Направление осей системы координат зависит от режима, см. функцию `mode()`. Дополнительную информацию можно получить из документации на модуль `turtle`. Обратите внимание и на конфигурационный файл, в котором могут быть определены параметры главного окна и не только.

2. Для рисования графическая система Python предоставляет пользователю холст (полотно, канва – canvas). Размеры холста можно задать через функцию `screensize()`:

```
turtle.screensize(Cw, Ch, bg),
```

где `Cw`, `Ch` – размеры холста по горизонтали и вертикали, а `bg` – цвет фона.

Замечание: Если воспользоваться функцией `screensize()` для изменения только цвета фона;

```
turtle.screensize(bg="yellow"),
```

то размеры холста будут установлены как размеры по умолчанию.

Другой способ определить размеры холста, а заодно и положение мировой системы координат (world coordinate system) – это использовать функцию `setworldcoordinates()`:

```
turtle.setworldcoordinates(Xlc, Ylc, Xrc, Yrc),
```

где и `Xrc, Yrc` – координаты левого нижнего и правого верхнего углов холста, соответственно. При этом в графической системе устанавливается режим "world". Для задания цвета фона может использоваться функция `bgcolor()`. Если размеры холста больше размеров главного окна, то появляются полосы прокрутки – виджеты холста.

И главное окно, и холст инициализируются при импорте модуля `turtle`. Значения, для инициализации, берутся из конфигурационного файла, а при его отсутствии устанавливаются по умолчанию. Кроме этого можно, используя функционал модуля, изменять размеры и положение главного окна, а так же параметры холста по своему усмотрению.

Если размеры главного окна и его координаты задаются в пикселах, то размеры холста – пользовательские единицы. Например, вы рисуете на листе бумаги геометрические фигуры, а размеры задаете в миллиметрах. Если ваш лист формата А4, то размеры холста можно определить как 210x297, а координаты некоторой точки на этом холсте можно задать как (34.7; 12.0). Если единицей измерения являются сантиметры, то размеры холста можно задать как 21x29.7, а координаты этой же точки будут – (3.47; 1.2). И наконец, если вы рисуете Солнечную систему, то с таким же успехом можно определить размеры холста в астрономических единицах и указывать положения изображаемых тел в тех же единицах и их долях.

В каких отношениях находятся введенные понятия: главное окно, холст, мировая система координат? Рис.7.1 поясняет сказанное.

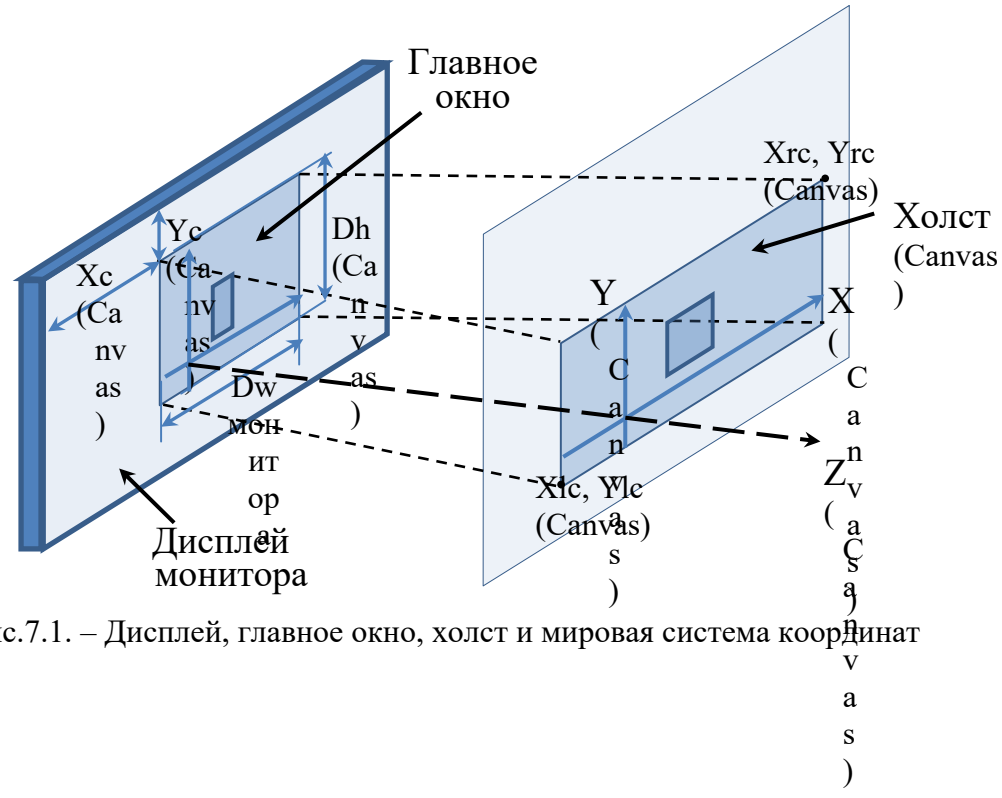

Рис.7.1. – Дисплей, главное окно, холст и мировая система координат

Холст размещается в плоскости проектирования (Z=0). Координаты Xlc и Ylc определяют положение левого нижнего угла холста, а Xrc, Yrc – правого. Размеры холста составляют: Cw = Xrc – Xlc и Ch = Yrc – Ylc. Графическая система проектирует холст на главное окно так, что угловые точки главного окна и холста совпадают, и устанавливается соответствие между единицами измерения в мировой системе координат и пикселами главного окна – масштаб. Масштабы, установленные по осям X и Y, могут быть разными.

Из рисунка очевиден следующий момент: если отношение ширины холста к его высоте и отношение ширины главного окна к его высоте (соотношение сторон – aspect ratio) неодинаковые, то возникает искажение. Искажаются углы и отношения между элементами изображения. Как показано на рис.7.1, квадрат преобразуется в прямоугольник, поскольку ширина холста больше ширины главного окна при их равной высоте. Этот эффект может быть использован, например, для изменения размеров изображения или преднамеренного их искажения. Кстати, аналогичный эффект можно наблюдать с графиками в Excel.

Замечание: На рисунке координаты левого нижнего угла холста отрицательные и начало системы координат находится на холсте. Если координаты угла определить как положительные числа, то начало координат будет находиться вне холста. Фигуры, размещаемые на холсте, могут находиться за размерами холста и, соответственно проектироваться вне главного окна. Для обнаружения таких "убежавших" фигур можно изменять размеры путем перетаскивания границ мышкой.

Прежде, чем строить график, надо принять решение о том, в каких диапазонах будут изменяться аргумент и значение функции. Примем следующие обозначения: Xmin, Xmax – диапазон изменения аргумента, а Ymin, Ymax – диапазон значений функции. Тогда параметры холста можно задать так:

```
turtle.setworldcoordinates(Xmin, Ymin, Xmax, Ymax)
```

Следует учитывать и то, что бордюр окна перекрывает часть изображения, поэтому параметры холста и соответствующие параметры главного окна можно задавать с небольшим запасом. Это небольшая проблема и она может быть решена в процессе отладки программы.

Если считать, что ширина и высота главного окна, в которое будет спроектирован холст, составляют Dw и Dh, соответственно, то для исключения искажений углов и элементов графика следует удовлетворить следующее условие:

```
k = Dw / Dh = (Xmax – Xmin) / (Ymax – Ymin)
```

$$k = \frac{Dw}{Dh} = \frac{Xmax - Xmin}{Ymax - Ymin}$$

или, например, Dh = Dw / k $$Dh = \frac{Dw}{k}$$

Теперь можно перейти к решению наших задач.

Первый шаг:

Определим границы области, в которой будем рисовать наш график: границы изменения аргумента (x) и границы изменения значений функции (y). В общем случае, когда нам неизвестно поведение функции на заданном интервале аргумента, можно предварительно выполнить вычисления функции и найти минимальное и максимальное значения. Эти значения помогут задать границы области построения. К тому же такие вычисления можно сделать один раз, поместив результаты вычислений либо в массив, либо в список, который затем можно использовать при построении графика.

Вновь обратимся к лабораторной работе №2, Задание 1, и примем обозначения переменных, которые мы будем использовать в нашей программе.

а) Импорт модулей. Нам потребуются математические функции и функции модуля `turtle`:

```
from math import sqrt
import turtle as tr
```

б) Размеры по горизонтали и вертикали составляют (-10, 10) и (-2, 4) условных единиц соответственно. Увеличим эти размеры на 20% (по 10% с каждой стороны) с тем, что бы график был в границах главного окна (выбор процентов – это субъективная вещь). Мы получим:

```
aX = [-12, 12]  # левая и правая
aY = [-3, 5]    # нижняя и верхняя
```

в) Определим размер главного окна по горизонтали (`Dx`) в 800 пикселов. Тогда, при заданных параметрах графика, высота главного окна, для исключения искажений, составит:

```
Dx = 800
Dy = Dx / ((aX[1] - aX[0]) / (aY[1] - aY[0]))
# создаем главное окно
tr.setup(Dx, Dy)
```

г) Установим число точек для получения качественного рисунка:

```
# число точек рисования
Nmax = 1000
```

```
д) Установим мировую систему координат
```

```
tr.setworldcoordinates(aX[0], aY[0], aX[1], aY[1])
```

Второй шаг

На этом шаге нарисуем оси, нанесем деления и надписи, нарисуем стрелки. Для этого используем такие функции модуля`turtle`, как `up()`, `down()`,`goto()`,`write()`,`begin_fill()`,`end_fill()`.

Рисование осей, меток, стрелок и надписей к осям, оформлено в виде функций. Алгоритм рисования оси прост:

- поднять перо;
- перейти к левой (нижней) границе;
- опустить перо;
- перейти к правой (верхней) границе.

Алгоритм нанесения делений и надписей:

- поднять перо;
- в цикле от левой (нижней) границы к правой (верхней) границе, с заданным шагом;
- перейти к точке на оси;
- опустить перо;
- перейти к точке ниже (правее) оси. Расстояние от оси (длина штриха) подбирается опытным путем;
- поднять перо;
- перейти к точке нанесения надписи (ниже / правее оси): определяем опытным путем;
- выводим число, предварительно преобразовав его в строку;
- если цикл не завершился, то перейти к следующей итерации.

Алгоритм рисования стрелки:

- подготавливаем два списка с координатами характерных точек рисунка `a` и `b`. Первую точку с координатами (0, 0) не указываем, см. рис.8.2;
- в цикле строим многоугольник и регистрируем его как новую форму черепашки;
- назначаем черепашке новую форму – наш многоугольник и вытягиваем стрелку;
- устанавливаем черепашку в то место, где должна быть стрелка, разворачиваем ее под нужным углом и создаем штамп:
- надписываем ось.

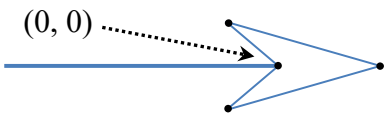

Рис.7.2.– Характерные точки стрелки, использованные в программе

Третий шаг

Переходим к рисованию функции. На этом шаге выберем другой цвет и толщину линии. По размеру области значений аргумента и по числу точек `Nmax` определяем приращение по оси `X`:

```
dx = (aX[1] - aX[0])/Nmax.
```

Алгоритм рисования следующий:

– получить начальную координату, вычислить значение функции;

– установить черепашку в начальную позицию;

– в цикле `while`, от начального значения аргумента до конечного значения;

– получить текущее значение аргумента, вычислить значение функции;

– если функция не вычислена (`None`), перо поднять, а иначе перейти к точке и опустить перо;

– продолжить цикл пока не получено значение, превышающее значение для правой точки аргумента.

Замечание: Следует обратить внимание на то, что в теле цикла проводится проверка на допустимость значений для `Y`. Так как мы приняли, что для особых точек функция возвращает значение `None`, то при обнаружении такой ситуации процесс рисования пропускается (перо поднимается).

Собственно все. Ниже приводится листинг программы и результат работы, см. рис.7.3.

PS: Удалите символы комментария в функции и посмотрите на результат.

Листинг программы

```
# -*- coding: cp1251 -*-
from math import sqrt
import turtle as tr
#
def Fun1(x):
    """
    Кривая  из лаб. 2 Задание 1
    """
    #if (x >= 5) and (x <= 7):
    #    return(None)
    if x < -5:
        y = 1
    elif x >= -5 and x < 0:
        y = -(3/5) * x - 2
    elif x >= 0 and x < 2:
        y = -sqrt(4 - x**2)
    elif x >= 2 and x < 4:
        y = x - 2
    elif x >= 4 and x < 8:
        y = 2 + sqrt(4 - (x - 6)**2)
    else: y = 2
    return(y)

def Axis(txy, ax = 'X'):
    """
```

```
    Рисование оси.
    txy - список: [Xmin, Xmax]
            или   [Ymin, Ymax]
    ax - 'X' или 'Y'
    """
    a = txy[0]
    b = txy[1]
    tr.up()
    if (ax == 'X'):
        pb = [a, 0]
        pe = [b, 0]
    else:
        pb = [0, a]
        pe = [0, b]
    tr.goto(pb)
    tr.down()
    tr.goto(pe)

def Mark(txy, ax = 'X'):
    """
    Маркировка оси.
    txy - список: [Xmin, Xmax]
            или   [Ymin, Ymax]
    ax - 'X' или 'Y'
    """
    a = txy[0]
    b = txy[1]
    tr.up()
    for t in range(a, b):
        if (ax =='X'):
            pb = [t, 0]
            pe = [t, 0.2]
            pw = [t, -0.5]
        else:
            pb = [0, t]
            pe = [0.2, t]
            pw = [0.2, t]
        tr.goto(pb)
        tr.down()
        tr.goto(pe)
        tr.up()
        tr.goto(pw)
        tr.write(str(t))

def Arrow(txy, ax = 'X'):
```

```
    """
    Рисование стрелки.
    txy - список: [Xmin, Xmax]
            или   [Ymin, Ymax]
    ax - 'X' или 'Y'
    """
# Параметры многоугольника
    a = [0.1, 0, -0.1]
    b = [-0.1, 0.3, -0.1]
    tr.up()
    tr.goto(0, 0)       # в начало
    tr.begin_poly()     # начинаем запись вершин
    for i in range(2):  # для всех вершин
        tr.goto(a[i], b[i]) # многоугольника
    tr.end_poly()       # останавливаем запись
    p = tr.get_poly()  # ссылка на многоугольник
# регистрируем новую форму черепашке
    tr.register_shape("myArrow",p)
    tr.resizemode("myArrow")
    tr.shapesize(1,2,1) # растягиваем (пример)
    if (ax == 'X'):           # для оси X
        tr.tiltangle(0)       # угол для формы
        tr.goto(txy[1]+0.2,0) # к месту стрелки
        pw = [int(txy[1]),-1.0] # надпись
    else:                     # для оси Y
        tr.tiltangle(90)      # угол для формы
        tr.goto(0,txy[1]+0.2) # к месту стрелки
        pw = [0.2, int(txy[1])] # надпись
    tr.stamp()       # оставить штамп - стрелка
# надпишем ось
    tr.goto(pw)              # к месту надписи
    tr.write(ax, font=("Arial",14,"bold"))
#
def main():
# Начальные параметры
# ************************************
# Границы графика: [Xmin, Xmax] и [Ymin, Ymax]
    aX = [-12, 12]  # левая и правая
    aY = [-3, 5]    # нижняя и верхняя
# Главное окно
    Dx = 800
    Dy = Dx / ((aX[1] - aX[0]) / (aY[1] - aY[0]))
    tr.setup(Dx, Dy)
    tr.reset()
# Число точек рисования
```

```
    Nmax = 1000
#
# Установка мировой системы координат
    tr.setworldcoordinates(aX[0],aY[0],
                           aX[1],aY[1])
#
    tr.title("Lab_8_2_1")     # заголовок
    tr.width(2)               # толщина линии
    tr.color("blue", "blue") # цвет и заливка
# ************************************
    tr.ht()                   # невидимая
    tr.tracer(0,0)            # нет задержек
#
# X - ось, метки, стрелка
    Axis(aX, 'X')
    Mark(aX, 'X')
    Arrow(aX, 'X')
# Y - ось, метки, стрелка
    Axis(aY, 'Y')
    Mark(aY, 'Y')
    Arrow(aY, 'Y')
#
# Функция
    tr.color("green")         # цвет линии
    tr.width(3)               # и толщина
    dx = (aX[1]-aX[0])/Nmax  # шаг
# в начало
    x = aX[0]
    y = Fun1(x)
    if (y is None):
        tr.up()
        tr.goto(x, 0)
    else:
        tr.goto(x, y)
        tr.down()
# рисуем
    while x <= aX[1]:
        x = x + dx
        y = Fun1(x)
        if (y is None):
            tr.up()
            continue
        else:
            tr.goto(x, y)
            tr.down()
```

```
#
if __name__ == "__main__":
    main()
#
# Комментировать при работе в IDLE
# tr.mainloop()
```

Результат работы программы

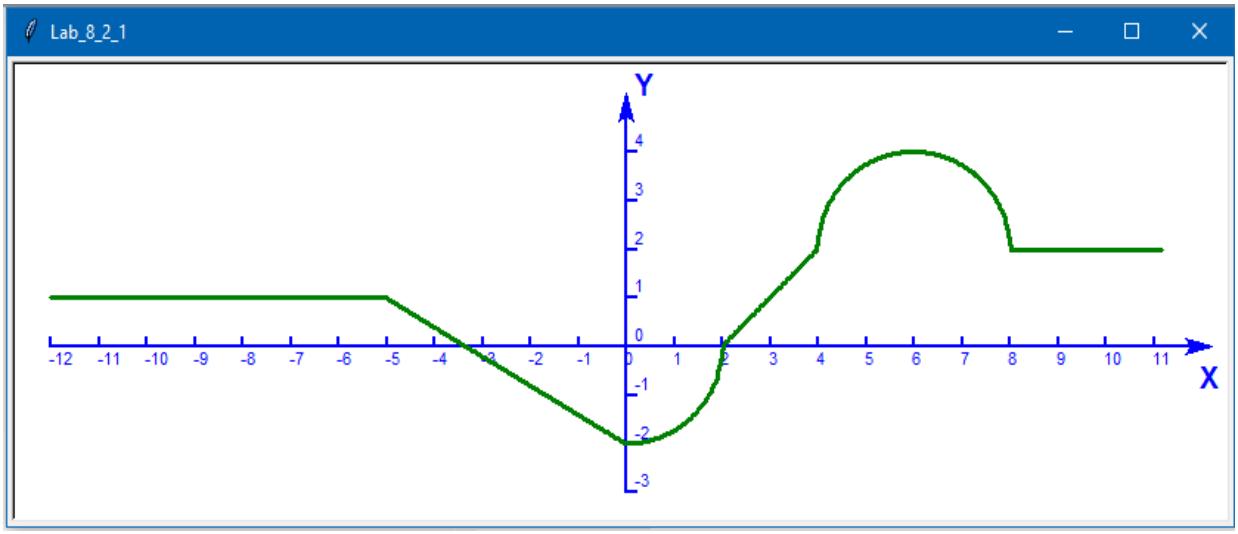

Рис.7.3.– График функции

Вторая задача

Для решения второй задачи (лабораторная работа №2, Задание 2), нам потребуется понимание того, что такое метод Монте-Карло и генератор случайных чисел.

Напомним, что в задании требовалось написать программу, которая определяет по введенным пользователем координатам точки, попадает ли эта точка в заштрихованную область.

Теоретическое введение

Метод Монте-Карло – численный метод решения математических задач при помощи моделирования случайных величин (метод статистических испытаний).

Одним из простейших механизмов генерации случайных величин является рулетка – основной атрибут игорных домов. Европейский город, знаменитый игорными домами – Монте-Карло, столица княжества Монако. От имени этого города и пошло название метода.

Одной из задач, решаемых методом Монте-Карло, является задача вычисления площади или объема сложной фигуры. Суть метода в следующем. Если фигура изображена на плоскости, то около нее можно описать другую фигуру, площадь которой мы можем вычислить точно. Например, круг или правильный многоугольник. Генерируя `N` случайных точек, координаты которых будут равномерно распределены по поверхности описанной фигуры, мы можем подсчитать число точек, которые попали в

фигуру с неизвестной площадью – N_f. Зная площадь описывающей фигуры S, можно вычислить площадь искомой фигуры с определенной точностью:

$$S_f = S \cdot \frac{N_f}{N}$$

Точность будет тем выше, чем больше точек будет сгенерировано и чем равномернее они будут разбросаны по всей описывающей фигуре. Точность пропорциональна $\sqrt{D/N}\sqrt{\frac{D}{N}}$, где D – некоторая постоянная, аN – число испытаний (число точек).

Обратите внимание на то, что для повышения точности на порядок (в 10 раз), потребуется увеличить число испытаний в 100 раз. Применение метода стало возможным с развитием вычислительной техники.

Больше информации о методе Монте-Карло можно найти в Интернете.

Решение второй задачи

Для нашей задачи необходим генератор, который формирует случайные числа с вероятностью, равномерно распределенной в заданном интервале. Для этой цели используем функцию `uniform()` модуля `random`:

```
from random import uniform
```

Используя листинг программы, написанной в лабораторной работе №2, Задание 2 и знания, полученные при решении предыдущей задачи, нам несложно написать программу, в которой координаты точек будут генерироваться случайным образом, и результат попадания будет отображаться графически.

Определение точной площади фигуры

В задачах, которые описаны в лабораторной работе 2 Задание 2, площадь большинства заштрихованных фигур находится просто.

Поскольку наша фигура образована пересечением кубической параболы и прямой, точное определение площади возможно с использованием интегрального исчисления.

В общем случае площадь фигуры, образованной двумя линиями, можно найти из выражения: $S_f = \int_a^b (f_2(x) - f_1(x)) \cdot dx$ $S_f = \int_a^b \left(f_2(x) - f_1(x)\right) \cdot dx$, где $f_2(x)$ и $f_1(x)$ – функции, которыми ограничена фигура, a и b – границы фигуры слева и справа, соответственно. При этом $f_2(x) >= f_1(x)$. Уравнения для наших линий следующие:

$f_1(x) = 2 \cdot x + 2$ и $f_2(x) = x^3 - 4x^2 + x + 6$ $f_2(x) = x^3 - 4 \cdot x^2 + x + 6$

Разобьем нашу фигуру на две области. Одна область будет в диапазоне аргумента [-1, 1], а вторая – [1, 4]. Тогда получим, что площадь равна сумме двух интегралов:

$$S_f = \int_{-1}^{1} \left(f_2(x) - f_1(x)\right) \cdot dx + \int_{1}^{4} \left(f_1(x) - f_2(x)\right) \cdot dx$$

Подробного решения не приводим. Вычисление дает: $S_f = 253/12 \approx 21.0833$

$$S_f = \frac{25\text{з}}{1\text{2}} \approx 21.083\text{з}$$

Для вычисления площади фигуры методом Монте-Карло опишем около нее прямоугольник с основанием 7 и высотой 13 условных единиц (диапазон $a_X = [–2, 5]$ и $a_Y = [–2, 11]$). Площадь прямоугольника составляет:

$$S = [5– (–2)] * [11– (–2)] = 91.$$

В программе нет каких-либо особенностей. Следует обратить внимание лишь на то, что рисование большого числа точек требует много времени.

Для ускорения вычислений можно не рисовать точки, которые не попадают в заштрихованную область, т.е. оставить только первую часть условного оператора, который находится в теле цикла генерации координат точек.

Листинг программы

```
# -*- coding: cp1251 -*-
import turtle as tr
from random import uniform
#
def fun2_2(x,y):
    if (x < -1) or (x > 4):
        flag = 0       #False
    if ((x>=-1) and (x<1) and (y>=2*x+2)
        and (y<=x**3-4*x**2+x+6) or (x>=1)
        and (x<=4) and (y>=x**3-4*x**2+x+6)
        and (y<=2*x+2)):
        flag = 1
    else:
        flag = 0
    return(flag)

# Инициализация turtle
# ************************************
# Границы графика
aX = [-2, 5]     # левая и правая
aY = [-2, 11]    # нижняя и верхняя
Dx = 300
Dy = Dx/((aX[1] - aX[0])/(aY[1] - aY[0]))
tr.setup(Dx, Dy, 200, 200)
tr.reset()
# число точек рисования
Nmax = 10000
#
```

```
# Установка мировой системы координат
tr.setworldcoordinates(aX[0], aY[0], aX[1], aY[1])
#
tr.title("Lab_8_2_2")    # заголовок
tr.width(2)              # толщина линии

tr.ht()         # невидимая
tr.tracer(0,0) # 0 - задержка между обновлениями
# ************************************
#
# рисование функции
tr.up()
mfun = 0          # точек попало в
                  # заштрихованную область
for n in range(Nmax):
# генерируем координаты точки
    x = uniform(aX[0],aX[1])
    y = uniform(aY[0],aY[1])
    tr.goto(x,y)
    if fun2_2(x,y) != 0:   # попала
        tr.dot(3,"green")
        mfun += 1
#    else:                  # не попала
#        tr.dot(3, "#ffccff")
#
tr.color("blue", "blue") # цвет и заливка
#
# Рисуем оси
# Ось X
tr.up()
tr.goto(aX[0], 0)
tr.down()
tr.goto(aX[1], 0)
# Ось Y
tr.up()
tr.goto(0, aY[1])
tr.down()
tr.goto(0, aY[0])
#
# координатные метки
# и надписи на оси X
tr.up()
for x in range(aX[0], aX[1]):
    tr.goto(x, 0.1)
    tr.down()
```

```
    tr.goto(x, 0)
    tr.up()
    tr.sety(-0.4)
    coords = str(x)
    tr.write(coords)
#
# на оси Y
for y in range(aY[0], aY[1]):
    tr.goto(0, y)
    tr.down()
    tr.goto(0.1, y)
    tr.up()
    tr.setx(0.2)
    coords = str(y)
    tr.write(coords)
#
# Рисуем стрелки
# на оси X
poli = [0, 0.1, 0, -0.1, 0]
Arrbeg = int(aX[1])
Xpoli = [Arrbeg, Arrbeg - 0.1, Arrbeg+0.3,
         Arrbeg - 0.1, Arrbeg]
tr.goto(Xpoli[0],poli[0])
tr.begin_fill()
tr.down()
for i in range(1,5):
    tr.goto(Xpoli[i], poli[i])
tr.end_fill()      # заливаем стрелку
# Надпишем ось X
tr.up()
tr.goto(Arrbeg, -0.7)
tr.write("X", font=("Arial",14,"bold"))
#
# на оси Y
Arrbeg = int(aY[1])
Ypoli = [Arrbeg, Arrbeg - 0.1, Arrbeg + 0.3,
         Arrbeg - 0.1, Arrbeg]
tr.up()
tr.goto(poli[0], Ypoli[0])
tr.begin_fill()
tr.down()
for i in range(1,5):
    tr.goto(poli[i],Ypoli[i])
tr.end_fill()
# Надпишем ось Y
```

```
tr.up()
tr.goto(0.2, Arrbeg)
tr.write("Y", font=("Arial",14,"bold"))
Sf = (aX[1] - aX[0]) * (aY[1] - aY[0]) * mfun/Nmax
tr.goto(1, 9)
fstr = "N = {0:8d}\nNf = {1:8d}\nSf = {2:8.2f}"
meseg = fstr.format(Nmax, mfun, Sf)
tr.write(meseg, font=("Arial",12,"bold"))
print(meseg)
# Комментировать при работе в IDLE
# tr.done()
```

Результат работы программы

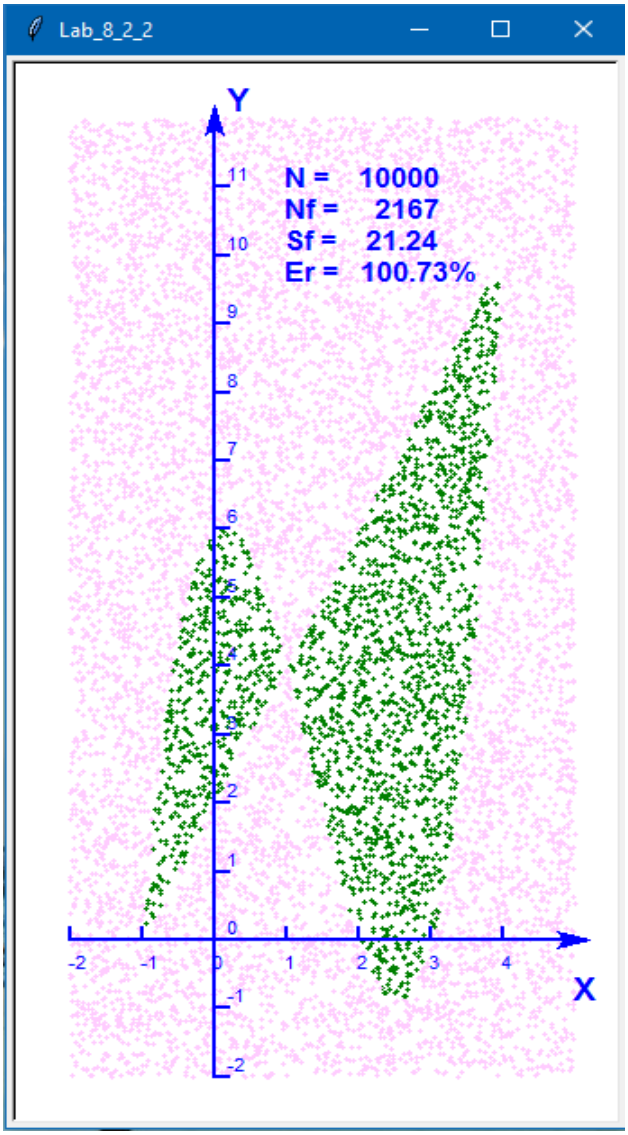

Рис.8.4. – Результат работы программы

Список рекомендованной литературы

1. Прохоренок Н.А., В.А. Дронов, Python 3и PyQt 5. Разработка приложений, БХВ-Петербург, 2017

2. Эйнджел Э., Интерактивная компьютерная графика. Вводный курс на базе OpenGL, 2 изд., и.д. "Вильямс", 2001
3. Хахаев И.А., Практикум по алгоритмизации и программированию на Python, Альт Линукс, 2011
4. Ермаков С.М., Метод Монте-Карло в вычислительной математике (вводный курс), СПб., 2009
5. Новожилов Б.В., Метод Монте-Карло. М.: Знание, 1966.

Задание к лабораторной работе №7
"Программирование графики". Модуль turtle

Выполнить в графической форме лабораторные работы №3 Задания 1, 2, 3.

При выполнении Задания 1 решить обратную задачу – нарисовать график функции.

В Задании 2, используя метод Монте-Карло определить площадь заштрихованной части фигуры. Получить точечное изображение фигуры, используя до 10000 испытаний. Выполнить оценку точности вычисления площади в процентах по отношению к реальной площади. Реальную площадь получить геометрическими методами, а при необходимости использовать интегральное исчисление.

В Задании 3 нарисовать график функции, значения которой вычисляются через ряд с точностью 10^{-3}.

Изображение графиков 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.