**Федеральное государственное образовательное бюджетное**

**учреждение высшего образования**

**«Санкт-Петербургский государственный университет**

**телекоммуникаций им. проф. М. А. Бонч-Бруевича»**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Факультет

инфокоммуникационных сетей и систем

Кафедра

Защищенных сетей и систем

**КУРСОВАЯ РАБОТА**

учебная дисциплина «Технологии и методы программирования»

Тема: **«**Программа вывода графиков со сканированием точек**»**

Выполнил(а) студент(ка) II курса группы

ИКБ-63 Громов П.В.

Проверил Штеренберг С.И.

\_\_\_\_\_\_\_\_\_\_(подпись)

**Оглавление**

[**Введение** 3](#_Toc515842844)

[**1** **Реализация Алгоритма.** 4](#_Toc515842845)

[**Заключение** 5](#_Toc515842846)

[**Список использованных интернет-ресурсов** 6](#_Toc515842847)

[**Приложение** 7](#_Toc515842848)

# **Введение**

В этой курсовой работе, будет рассмотрен пример реализации программы вывода графиков со сканированием точек

**Задание по курсовой работе:** выполнить реализацию программу вывода графиков со сканированием точек на языке программирования “Python”.

# **Реализация Алгоритма.**

С помощью библиотеки matplotlib была реализованна графическая часть программы, библиотека numpy используется для работы с матрицами.

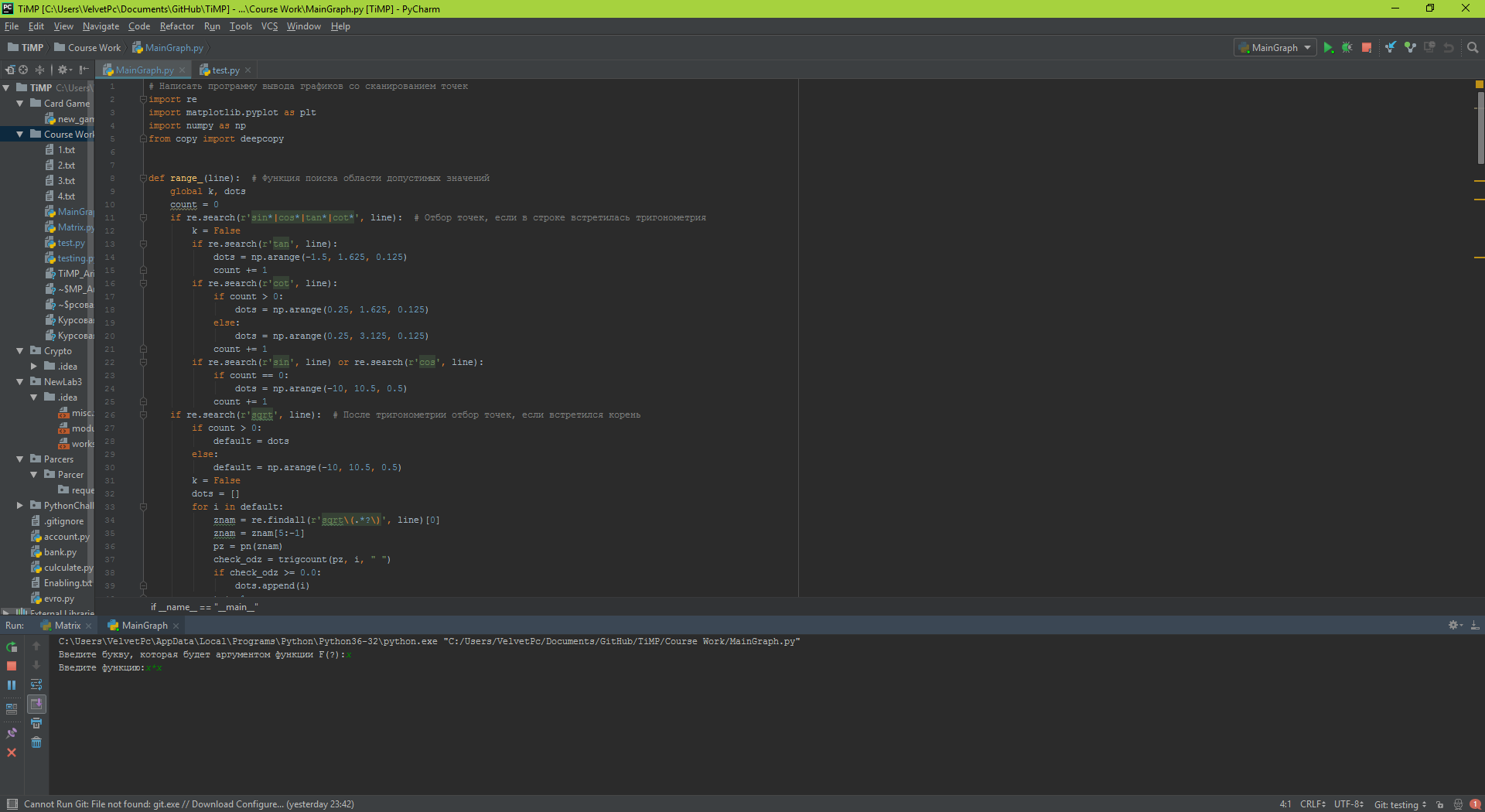


Рисунок 1. Интерфейс.

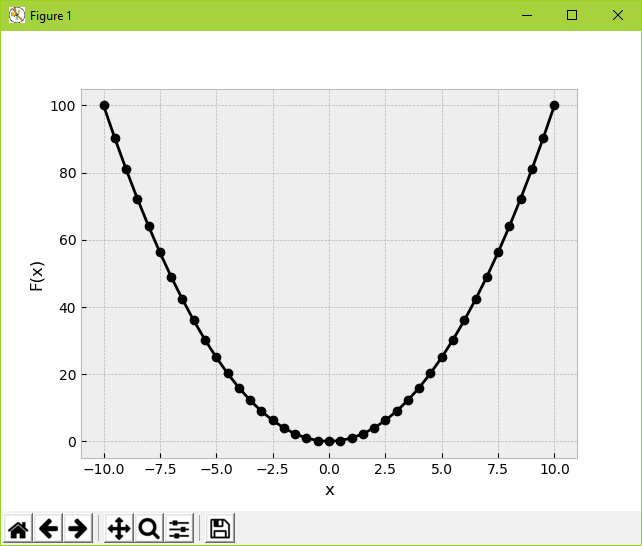


Рисунок 2. График этой функции

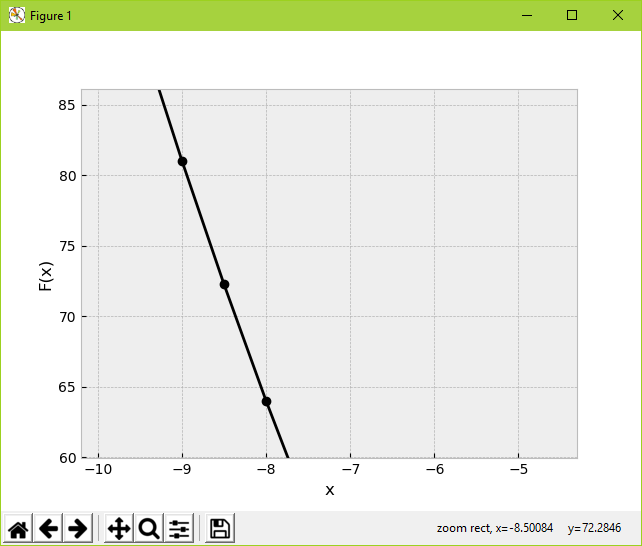


Рисунок 3. Пример приближения части графика и сканирования координат

# **Заключение**

В данной курсовой работе была выполнена программа выполняющая различные действия с матрицами на языке программирования “Python”.

# **Список использованных интернет-ресурсов**

https://matplotlib.org/ - Документация Matplotlib

<http://www.numpy.org/> - Документация NumPy

<https://ru.stackoverflow.com/> - Разбор типовых ошибок

# **Приложение**

import re  
import matplotlib.pyplot as plt  
import numpy as np  
from copy import deepcopy  
  
  
def range\_(line): # Функция поиска области допустимых значений  
 global k, dots  
 count = 0  
 if re.search(r'sin\*|cos\*|tan\*|cot\*', line): # Отбор точек, если в строке встретилась тригонометрия  
 k = False  
 if re.search(r'tan', line):  
 dots = np.arange(-1.5, 1.625, 0.125)  
 count += 1  
 if re.search(r'cot', line):  
 if count > 0:  
 dots = np.arange(0.25, 1.625, 0.125)  
 else:  
 dots = np.arange(0.25, 3.125, 0.125)  
 count += 1  
 if re.search(r'sin', line) or re.search(r'cos', line):  
 if count == 0:  
 dots = np.arange(-10, 10.5, 0.5)  
 count += 1  
 if re.search(r'sqrt', line): # После тригонометрии отбор точек, если встретился корень  
 if count > 0:  
 default = dots  
 else:  
 default = np.arange(-10, 10.5, 0.5)  
 k = False  
 dots = []  
 for i in default:  
 znam = re.findall(r'sqrt\(.\*?\)', line)[0]  
 znam = znam[5:-1]  
 pz = pn(znam)  
 check\_odz = trigcount(pz, i, " ")  
 if check\_odz >= 0.0:  
 dots.append(i)  
 count += 1  
 if re.search(r'/', line): # После тригонометрии и корня, если встретилось деление  
 if count > 0:  
 default = dots  
 else:  
 default = np.arange(-10, 10.5, 0.5)  
 k = False  
 if re.search(r'sin\*|cos\*|tan\*|cot\*|sqrt\*', line):  
 new\_dots = []  
 znam = re.findall(r'\/\(.\*?\)+', line)[0]  
 znam = znam[2:-1]  
 check\_odz = odz\_ext(znam)  
 for i in dots:  
 if i != check\_odz:  
 new\_dots.append(i)  
 dots = new\_dots  
 else:  
 dots = []  
 for i in default:  
 znam = re.findall(r'\/\(.\*?\)', line)[0]  
 znam = znam[2:-1]  
 pz = pn(znam)  
 check\_odz = trigcount(pz, i, " ")  
 if check\_odz != 0.0:  
 dots.append(i)  
 count += 1  
 if count != 0:  
 return trig\_logic(line)  
 else: # Строится обычный график, если одз не требутеся  
 k = True  
 dots = np.arange(-10, 10.5, 0.5)  
 return pn(line)  
  
  
def odz\_ext(trig\_line): # Дополнение к основному одз, выполняющее расчеты  
 new\_trig\_line = trig\_line  
 items\_trigs = re.findall(r'sin\(.\*?\)+|cos\(.\*?\)+|tan\(.\*?\)+|cot\(.\*?\)+|sqrt\(.\*?\)+', new\_trig\_line)  
 for i in dots:  
 for j in items\_trigs:  
 text\_in\_brackets = re.findall(r'\(.\*\)+', j)[0]  
 pz = pn(text\_in\_brackets[1:-1])  
 result\_of\_one\_item = round(trigcount(pz, i, j), 3)  
 if result\_of\_one\_item == 0.0:  
 return i  
  
  
def pn(initial): # Запись выражения в обратную польскую запись  
 priority = {"+": 1, "-": 1, "\*": 2, "/": 2, "(": 0}  
 out = []  
 operators = []  
 first\_ch = re.findall(r'\-?\w+\.?\w\*|[(+\*)/\-]', initial)  
 for item in first\_ch:  
 if item == ")":  
 for item2 in range(len(operators)):  
 if operators[-1] != "(":  
 out.append(operators[-1])  
 operators.remove(operators[-1])  
 else:  
 operators.remove("(")  
 elif not item.isdigit() and not item.isalpha() and len(item) < 2:  
 if "(" in operators and len(operators) > 1 and priority[item] <= priority[operators[-1]]:  
 for item3 in range(len(operators)):  
 if operators[-1] != "(":  
 out.append(operators[-1])  
 operators.remove(operators[-1])  
 else:  
 operators.append(item)  
 break  
 elif "(" == item and len(operators) != 0:  
 operators.append(item)  
 elif "(" not in operators and len(operators) != 0 and priority[item] <= priority[operators[-1]]:  
 for item4 in range(len(operators)):  
 out.append(operators[-1])  
 operators.remove(operators[-1])  
 else:  
 operators.append(item)  
 else:  
 operators.append(item)  
 else:  
 out.append(item)  
 else:  
 for item5 in range(len(operators)):  
 out.append(operators[-1])  
 operators.remove(operators[-1])  
 if k:  
 return count(out)  
 else:  
 return out  
  
  
def trig\_logic(trig\_line): # Логика по расчету тригонометрии и корня  
 new\_trig\_line = deepcopy(trig\_line)  
 graph\_y = []  
 items\_trigs = re.findall(r'sin\(.\*?\)|cos\(.\*?\)|tan\(.\*?\)|cot\(.\*?\)|sqrt\(.\*?\)', new\_trig\_line)  
 for i in dots:  
 for j in items\_trigs:  
 text\_in\_brackets = re.findall(r'\(.\*\)+', j)[0]  
 pz = pn(text\_in\_brackets[1:-1])  
 result\_of\_one\_item = round(trigcount(pz, i, j), 3)  
 new\_trig\_line = new\_trig\_line.replace(j, str(result\_of\_one\_item))  
 else:  
 pz = pn(new\_trig\_line)  
 final\_result = trigcount(pz, i, new\_trig\_line)  
 graph\_y.append(round(final\_result, 4))  
 new\_trig\_line = deepcopy(trig\_line)  
 return dots, graph\_y  
  
  
def trigcount(y, tt, trig\_item): # Непосредственный расчет тригонометрии и корня  
 for j in range(len(y)):  
 if y[j] == value:  
 y[j] = tt  
 y = list(map(lambda x: x if x == "+" or x == "-" or x == "\*" or x == "/" else float(x), y))  
 lenght = len(y)  
 i = 0  
 while i < lenght:  
 if len(y) != 1:  
 if y[i] in ["+", "-", "\*", "/"]:  
 if y[i] == "\*":  
 y[i] = y[i - 2] \* y[i - 1]  
 y.pop(i - 1)  
 y.pop(i - 2)  
 i = 0  
 lenght -= 2  
 elif y[i] == "+":  
 y[i] = y[i - 2] + y[i - 1]  
 y.pop(i - 1)  
 y.pop(i - 2)  
 i = 0  
 lenght -= 2  
 elif y[i] == "-":  
 y[i] = y[i - 2] - y[i - 1]  
 y.pop(i - 1)  
 y.pop(i - 2)  
 i = 0  
 lenght -= 2  
 elif y[i] == "/":  
 y[i] = y[i - 2] / y[i - 1]  
 y.pop(i - 1)  
 y.pop(i - 2)  
 i = 0  
 lenght -= 2  
 else:  
 i += 1  
 else:  
 break  
 if re.search(r'sqrt', trig\_item):  
 return np.sqrt(y[0])  
 elif re.search(r'sin', trig\_item):  
 return np.sin(y[0])  
 elif re.search(r'cos', trig\_item):  
 return np.cos(y[0])  
 elif re.search(r'tan', trig\_item):  
 return np.tan(y[0])  
 elif re.search(r'cot', trig\_item):  
 return 1 / np.tan(y[0])  
 else:  
 return y[0]  
  
  
def count(y): # Стандартный расчет функции  
 y\_copy = deepcopy(y)  
 graph\_y = []  
 for i in dots:  
 for j in range(len(y)):  
 if y[j] == value:  
 y[j] = i  
 else:  
 y = list(map(lambda x: x if x == "+" or x == "-" or x == "\*" or x == "/" else float(x), y))  
 lenght = len(y)  
 i = 0  
 while i < lenght:  
 if len(y) != 1:  
 if y[i] in ["+", "-", "\*", "/"]:  
 if y[i] == "\*":  
 y[i] = y[i - 2] \* y[i - 1]  
 y.pop(i - 1)  
 y.pop(i - 2)  
 i = 0  
 lenght -= 2  
 elif y[i] == "+":  
 y[i] = y[i - 2] + y[i - 1]  
 y.pop(i - 1)  
 y.pop(i - 2)  
 i = 0  
 lenght -= 2  
 elif y[i] == "-":  
 y[i] = y[i - 2] - y[i - 1]  
 y.pop(i - 1)  
 y.pop(i - 2)  
 i = 0  
 lenght -= 2  
 elif y[i] == "/":  
 y[i] = y[i - 2] / y[i - 1]  
 y.pop(i - 1)  
 y.pop(i - 2)  
 i = 0  
 lenght -= 2  
 else:  
 i += 1  
 else:  
 graph\_y.append(y[0])  
 y = deepcopy(y\_copy)  
 break  
 return dots, graph\_y  
  
  
def main():  
 global value  
 value = input("Введите букву, которая будет аргументом функции F(?):")  
  
 while True: # Построение графика по заранее расчитанным точкам  
 statement = input("Введите функцию:")  
 if statement == "exit":  
 break  
 graph = range\_(statement)  
  
 with plt.style.context("bmh"):  
 plt.plot(graph[0], graph[1], 'k-o')  
 plt.xlabel(value)  
 plt.ylabel("F(" + value + ")")  
 plt.show()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()