Визуализация данных средствами языка Python, библиотек Tkinter и Matplotlib

САМОСТОЯТЕЛЬНАЯ РАБОТА

http://www.kimrt.ru

Онлайн-курс

Суперкомпьютерные технологии в задачах моделирования

Рассмотрим следующий код, который строит график функции $y = ax + b$.

Листинг 1.

```
# импорт библиотек
import tkinter as tk
from matplotlib.backends.backend_tkagg import (FigureCanvasTkAgg)
from matplotlib.figure import Figure
import numpy as np
from mpl_toolkits.axisartist.axislines import Subplot

# функция обработки нажатия кнопки button_graph
def graph(event):
    # присвоение значений переменным из текстовых полей
    a = int(entry_a.get())
    b = int(entry_b.get())
    ax.clear()  # отчистка графика
    ax.plot(x, x ** a + b)  # построение графика
    canvas.draw()  # прорисовка графика


# создание окна tkinter
window = tk.Tk()
# создание графика
fig = Figure(figsize=(5, 4), dpi=100)
ax = Subplot(fig, 111)
fig.add_subplot(ax)
x = np.arange(-10, 10, 1)  # генерация массива
ax.plot()

# объявление меток
label_y = tk.Label(text='x^a+b')
label_a = tk.Label(text='a')
label_b = tk.Label(text='b')
# объявление полей для ввода
entry_a = tk.Entry()
entry_b = tk.Entry()
```

```
# объявление кнопок
button_graph = tk.Button(text='draw')
# связывание кнопки с функцией
button_graph.bind("<Button-1>", graph)
button_quit = tk.Button(text="Quit", command=window.destroy)

# геометрия объектов в окне
label_y.pack()
label_a.pack()
entry_a.pack()
label_b.pack()
entry_b.pack()
button_graph.pack()
button_quit.pack()
canvas = FigureCanvasTkAgg(fig, master=window)
canvas.draw()
canvas.get_tk_widget().pack()
# цикл запуска событий tkinter
window.mainloop()
```

Сначала импортируем необходимые библиотеки и модули: *tkinter*, *FigureCanvasTkAgg* из серверной части *matplotlib*, *Figure* из *matplotlib*, *numpy*, *Subplot* из *matplotlib*. *Tkinter* необходим для построения графического интерфейса, *FigureCanvasTkAgg* необходим для взаимодействия между *tkinter* и *matplotlib*, *Figure* и *subplot* необходимы для работы с графиками, *numpy* необходим для работы с массивами [1]. Создаем окно (*window = tk.Tk()*), затем создаем объект класса *Figure* [2], который содержит все элементы графика *fig = Figure(figsize=(5, 4), dpi=100)*), где атрибут *figsize* задает размер, а атрибут *dpi* задает разрешение. Добавим объект класса Subplot к текущему объекту класса *Figure*, где *fig* – это объект. *111* – это атрибуты размещения *Subplot* в *fig*, 1 строка, 1 столбец, 1 график [3] (*ax = Subplot(fig, 111)*).
Далее размещаем Subplot [4] с помощью специального метода, *fig.add_subplot(ax).*

Создаем пустой график, *ax.plot()* и генерируем массив, который понадобится для построения графика, *x = np.arange(-10, 10, 1)*. Здесь -10 – начальное значение, 10 – конечное значение, 1 – шаг. Область в которой строится график зависит от начального и конечного значения в этом массиве, обратите на это внимание.

Объявим ярлыки и поля для ввода с помощью *Label* и *Entry* для *a* и *b*, а также кнопки с помощью команды Button. Кнопка выхода создается следующим образом.
button_quit = tk.Button(text="Quit", command=window.destroy)

Теперь сделаем кнопку для построения графика *ax+b* по введенным в текстовые поля *a* и *b*.

button_graph = tk.Button(text='draw')

Напишем функцию обработки для этой кнопки. С помощью *get()* получаем значение из текстового поля. Метод *clear()* очищает график.

```
def graph(event):
    # присвоение значений переменным из текстовых полей
    a = int(entry_a.get())
    b = int(entry_b.get())
    ax.clear()  # отчистка графика
    ax.plot(x, x ** a + b)  # построение графика
    canvas.draw()  # прорисовка графика
```

Привяжем выполнение этой функции к событию [5].

button_graph.bind("<Button-1>", graph)

Параметр *Button-1* указывает, что функция вызывается при нажатии на виджет левой кнопкой мыши. Далее разместим все с помощью менеджера геометрии *pack*. Для размещения графика необходимо написать следующее.

```
canvas = FigureCanvasTkAgg(fig, master=window)
canvas.draw()
canvas.get_tk_widget().pack()
```

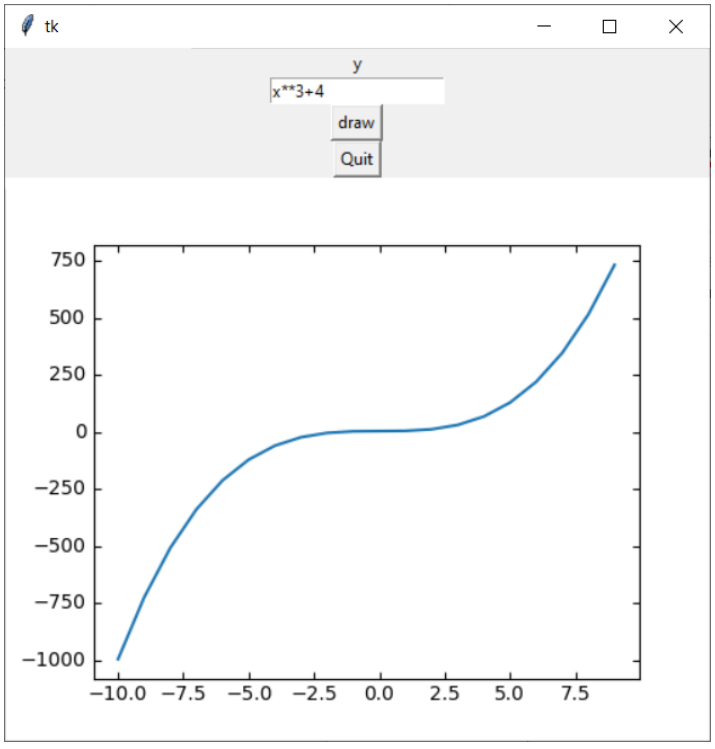
Ну и не забываем про *mainloop()*.

Следующий код строит график функции введенной в текстовое поле. Есть ряд ограничений по вводу функции: функция должна быть с одной независимой переменной, эта переменная должна быть определена как *x*, должны использоваться операторы python (например степень записывается так: **).

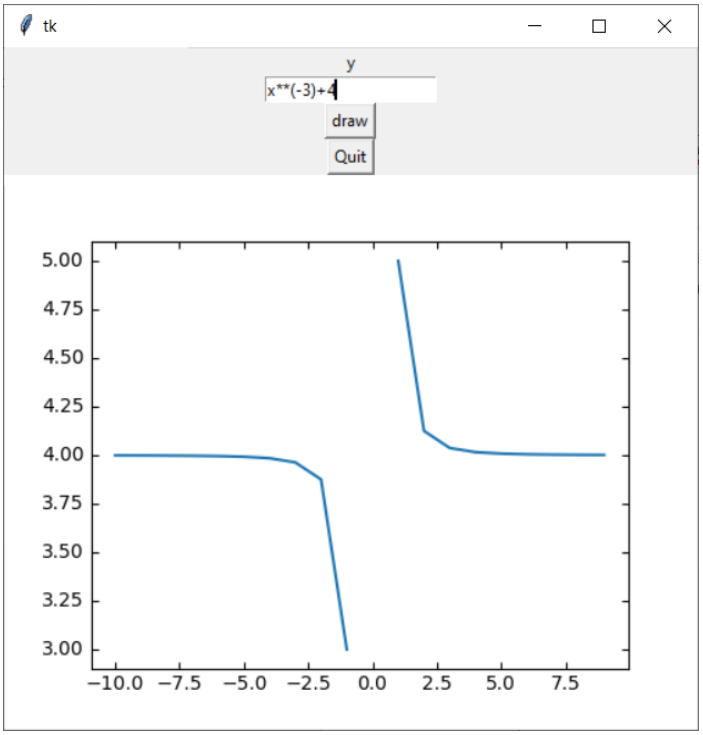
```
# импорт библиотек
import tkinter as tk
from matplotlib.backends.backend_tkagg import (FigureCanvasTkAgg)
from matplotlib.figure import Figure
import numpy as np
from mpl_toolkits.axisartist.axislines import Subplot

# функция обработки нажатия кнопки button_graph
def graph(event):
    y = str(entry_y.get())  # получение функции из поля
    ax.clear()  # отчистка графика
    ax.plot(x, eval(y))  # построение графика
    canvas.draw()  # прорисовка графика

# создание окна tkinter
window = tk.Tk()
# создание графика
```

```
fig = Figure(figsize=(5, 4), dpi=100)
ax = Subplot(fig, 111)
fig.add_subplot(ax)
np.seterr(divide='ignore')  # обработка ошибок
x = np.arange(-10, 10, 1, dtype=float) # генерация массива
ax.plot()

# объявление ярлыка
label_y = tk.Label(text='y')
# объявление поля для ввода
entry_y = tk.Entry()
# объявление кнопок
button_graph = tk.Button(text='draw')
button_graph.bind("<Button-1>", graph)
button_quit = tk.Button(text="Quit", command=window.destroy)
# геометрия объектов в окне
label_y.pack()
entry_y.pack()
button_graph.pack()
button_quit.pack()
canvas = FigureCanvasTkAgg(fig, master=window)
canvas.draw()
canvas.get_tk_widget().pack()
# цикл запуска событий tkinter
window.mainloop()
```

Запишем функцию x^3+4.



Теперь $x^{-3}+4$.



Рассмотрим основные моменты в коде.

В строке *«x = np.arange(-10, 10, 1, dtype=float)»* параметр *dtype=float* позволяет работать с числами типа *float*. Это позволило нам построить график функции $x^{-3} + 4$.

Также необходимо обработать ошибку при делении на ноль. Для этого можно использовать следующую команду.

np.seterr(divide='ignore')

Метод *seterr* устанавливает способ обработки ошибок с числами типа *float*. Параметр *divide='ignore'* устанавливает игнорирование ошибок вызванных делением на 0. Построение графика происходит следующим образом: получаем строку из текстового поля в переменную *y*, строим график с помощью *ax.plot()*.

```
y = str(entry_y.get())
ax.plot(x, eval(y))
```

Необходимо обратить внимание на функцию *eval()*. Эта функция выполняет выражение переданное ей. Поэтому необходимо использовать операторы python и *x* в качестве независимой переменной, так как другие переменные, кроме *x* не объявлены в программе и другие операторы будут распознаны неправильно или не распознаны вообще.

http://www.kimrt.ru/index/course_stm/0-24

Проект реализуется победителем Конкурса на предоставление грантов преподавателям магистратуры 2020/2021 благотворительной программы «Стипендиальная программа Владимира Потанина» Благотворительного фонда Владимира Потанина.

Источники

1. Пример кода встраивания matplotlib в tkinter. URL: https://matplotlib.org/stable/gallery/user_interfaces/embedding_in_tk_sgskip.html
2. Figure. URL: https://matplotlib.org/stable/api/figure_api.html#module-matplotlib.figure
3. Subplot. URL: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplot.html
4. Add_subplot. URL: https://matplotlib.org/stable/api/figure_api.html#matplotlib.figure.Figure.add_subplot
5. Метод bind. URL: https://younglinux.info/tkinter/bind