final_streamlines

December 8, 2021

1 Python code for plotting 2D potential flow, and combination of multiple flows.

1.1 Project Github

We will first import all the necessary modules

[ ]:
```
from tkinter import * #used for building a simple gui
from matplotlib.figure import Figure #plotting library
from matplotlib.backends.backend_tkagg import (FigureCanvasTkAgg,
NavigationToolbar2Tk)
from numpy.lib.utils import source
import numpy as np # Used for vectorial algorithms
import matplotlib.pyplot as plt
```

We keep al lthe different potential flows in a global list so that we can add new flows without losing the old ones

[ ]:
```
global flows
flows = []
```

We have different classes for each of the types of fundamental flows. These classes will have the all the required parameters of the flow.

1.2 Uniform flow

To define this fully, we only need - The velocity in the x direction - The velocity in the y direction

1.3 Source Flow

To define this fully, we will need: - The position of the Source - The strength of the source Based on these, we find the velocity induced by the flow at every point.

[ ]:
```
class Uniform:
    flow_type = "uniform"
    def __init__(self, x_direction, y_direction):
        self.x_velocity = np.full((100,100),x_direction)
```

```
        self.y_velocity = np.full((100,100),y_direction)

class Source:
    flow_type = "source"
    def __init__(self, x_pos, y_pos, strength):
        w = 9
        Y, X = np.mgrid[-w:w:100j, -w:w:100j]
        self.x_pos = np.full((100,100),x_pos)
        self.y_pos = np.full((100,100),y_pos)
        self.strength = int(strength)
        V =self.strength*(Y - self.y_pos)/(2*np.pi*((Y - self.y_pos)**2 + (X -␣
 ↪self.x_pos)**2))
        U =self.strength*(X - self.x_pos)/(2*np.pi*((Y - self.y_pos)**2 + (X -␣
 ↪self.x_pos)**2))
        self.y_velocity = np.where(V !=np.inf, V, 0)
        self.x_velocity = np.where(U !=np.inf, U, 0)
class Vortex:
    flow_type = "vortex"
    def __init__(self, x_pos, y_pos, strength):
        w = 9
        Y, X = np.mgrid[-w:w:100j, -w:w:100j]
        self.x_pos = int(x_pos)
        self.y_pos = int(y_pos)
        self.strength = int(strength)
        V = self.strength*(X - self.x_pos)/(2*np.pi*((Y - self.y_pos)**2 + (X -␣
 ↪self.x_pos)**2))
        U = -self.strength*(Y - self.y_pos)/(2*np.pi*((Y - self.y_pos)**2 + (X␣
 ↪- self.x_pos)**2))
        self.y_velocity = np.where(V !=np.inf, V, 0)
        self.x_velocity = np.where(U !=np.inf, U, 0)
```

2 plotting the vectorfield

To plot the vector field, we will iterate through the global flows list, and add the velocity induced by all the defined potential flows, at every point of the domain.

In this code, we have 100×100 points in the domain. To be able to use this in interactively, we need to be able to add the domains od all the flows efficiently, to do this, we use vectorized algorithms.

[ ]:
```
def refresh():
    try:
        canvas.clf()
    except NameError:
        canvas =None
    w = 9
    Y, X = np.mgrid[-w:w:100j, -w:w:100j]
```

```
    U = 0
    V = 0
    for i in flows:
        U = U +i.x_velocity
        V = V +i.y_velocity
    plot1.streamplot(X,Y,U,V, density = 0.5)
```

Now we define how to get input from the users by designing the GUI

[ ]:
```
def add_uniform():
    popup = Tk()
    popup.title("uniform flow definition")
    L1 = Label(popup, text="x direcion of the uniform flow")
    L1.grid(row = 0,column = 0)
    x_uniform = Entry(popup, bd =5)
    x_uniform.grid(row = 0,column = 1)
    L2 = Label(popup, text="y direcion of the uniform flow")
    L2.grid(row = 1,column = 0)
    y_uniform = Entry(popup, bd =5)
    y_uniform.grid(row = 1,column = 1)
    def close_window ():
        plt.clf()
        plt.cla()
        plot1.clear()
        global flows
        a = Uniform(int(x_uniform.get()),int(y_uniform.get()))
        flows.append(a)
        print(x_uniform.get(), y_uniform.get())
        refresh()
        popup.destroy()
    button = Button(popup, text = "Okay", command = close_window)
    button.grid(row = 2, column = 1)

def add_source():
    popup = Tk()
    popup.title("source flow definition")
    L1 = Label(popup, text="x position of the uniform flow")
    L1.grid(row = 0,column = 0)
    x_pos = Entry(popup, bd =5)
    x_pos.grid(row = 0,column = 1)
    L2 = Label(popup, text="y position of the uniform flow")
    L2.grid(row = 1,column = 0)
    y_pos = Entry(popup, bd =5)
    y_pos.grid(row = 1,column = 1)
    L3 = Label(popup, text = "strength of the source: ")
    L3.grid(row = 2, column = 0)
    strength = Entry(popup,bd =5)
```

```
    strength.grid(row = 2, column = 1)
    def close_window ():
        plt.clf()
        plot1.clear()
        plt.cla()
        global flows
        a = Source(int(x_pos.get()),int(y_pos.get()),int(strength.get()))
        flows.append(a)
        refresh()
        popup.destroy()
    button = Button(popup, text = "Okay", command = close_window)
    button.grid(row = 3, column = 1)

def add_vortex():
    popup = Tk()
    popup.title("Vortex flow definition")
    L1 = Label(popup, text="x position of the vortex")
    L1.grid(row = 0,column = 0)
    x_pos = Entry(popup, bd =5)
    x_pos.grid(row = 0,column = 1)
    L2 = Label(popup, text="y position of the vortex")
    L2.grid(row = 1,column = 0)
    y_pos = Entry(popup, bd =5)
    y_pos.grid(row = 1,column = 1)
    L3 = Label(popup, text = "strength of the vortex: ")
    L3.grid(row = 2, column = 0)
    strength = Entry(popup,bd =5)
    strength.grid(row = 2, column = 1)
    def close_window():
        plt.clf()
        plt.cla()
        plot1.clear()
        global flows
        a = Vortex(int(x_pos.get()),int(y_pos.get()),int(strength.get()))
        flows.append(a)
        refresh()
        popup.destroy()
    button = Button(popup, text = "Okay", command = close_window)
    button.grid(row = 3, column = 1)
def add_doublet():
    popup = Tk()
    popup.title("Doublet flow definition")
    L1 = Label(popup, text = "x positon of the doublet")
    L1.grid(row=0,column=0)
    x_pos = Entry(popup, bd =5)
    x_pos.grid(row =0, column =1)
    L2 = Label(popup, text = "y position of the doublet")
```

```
    L2.grid(row =1, column =0)
    y_pos =Entry(popup, bd = 5)
    y_pos.grid(row =1,column=1)
    L3 = Label(popup, text="Strength of the doublet")
    L3.grid (row= 2, column = 0)
    strength = Entry(popup,bd=5)
    strength.grid(row =2,column =1)
    def close_window():
        plt.clf()
        plt.cla()
        plot1.clear()
        global flows
        a = Source(int(x_pos.get()),int(y_pos.get()),int(strength.get()))
        b = Source(int(x_pos.get())+0.01,int(y_pos.get()), -int(strength.get()))
        flows.append(a)
        flows.append(b)
        refresh()
        popup.destroy()
    button = Button(popup, text = "Okay", command = close_window)
    button.grid(row = 3, column = 1)
window =Tk()
window.title("Potential flows")

window.geometry("500x500")

uniform_button = Button(master = window,
                        command = add_uniform,
                        height = 2,
                        width = 10,
                        text = "Uniform")
refresh_button = Button(master = window,
                         command = refresh,
                         height = 2,
                         width = 10,
                         text = "refresh")
source_button = Button(master = window,
                        command = add_source,
                        height = 2,
                        width = 10,
                        text = "Source")
vortex_button = Button(master = window,
                        command = add_vortex,
                        height = 2,
                        width = 10,
                        text = "Vortex")
doublet_button = Button(master = window,
                        command = add_doublet,
```

```
                        height = 2,
                        width = 10,
                        text = "Doublet")

uniform_button.pack(side = TOP)
source_button.pack(side = TOP)
vortex_button.pack(side = TOP)
doublet_button.pack(side =TOP)
refresh_button.pack(side = TOP)

fig = Figure(figsize = (5,5),dpi = 100)
plot1 = fig.add_subplot(111)
canvas = FigureCanvasTkAgg(fig,master = window)
fig.canvas.draw()

canvas.get_tk_widget().pack(side = BOTTOM)
toolbar = NavigationToolbar2Tk(canvas, window)
toolbar.update()

window.mainloop()
```

3 Known Issues

- The plot updates after the window has been maximised(Tested and replicated on Ubuntu 18.04/Python3.6.9). Bug could not be replicated on windows 10/Python3.8.1.

4 Possible upgrades

- Draggable potential flows in real time
  - Work on this is started in the ”draggable” branch of Project Github
- Refactor the code to not rely on global variable.
- Include options for more flow features(potential lines, pressure field, etc)