International Journal of Engineering Applied Sciences and Technology, 2022
Vol. 6, Issue 11, ISSN No. 2455-2143, Pages 213-217
Published Online March 2022 in IJEAST (http://www.ijeast.com)

OBTAINING EXACT SOLUTIONS OF VISCO-INCOMPRESSIBLE PARALLEL FLOWS USING PYTHON

Parth Singh Pawar, Dhananjay R. Mishra, Pankaj Dumka
Department of Mechanical Engineering,
Jaypee University of Engineering and Technology,
Guna-473226, Madhya Pradesh, India

***Abstract*— This paper applies the solution of ODE's encounter in fluid mechanics by augmenting it with symbolic Python. The implementation procedure has been illustrated for two types of parallel flows, i.e., Couette and Hagen Poiseuille flow. The suggestive results thus obtained are plotted and presented using Matplotlib. The manuscript will help beginners of fluid mechanics solve the fluid flow problems computationally and produce the results in a better way.**

***Keywords*— Couette flow; Hagen Poiseuille flow; Parallel flow; Python; SymPy; Matplotlib**

I. INTRODUCTION

Navier stokes equations, often called NS equations, combine mass and momentum equations. The is no single closed-form solution to the NS equation to date, as the momentum equation is a nonlinear partial differential equation (PDE) [1]. Some scientists and mathematicians have tried to solve these equations for a particular class of flows called parallel flow [2]. Parallel flows are often encountered while studying viscous flow theory. Beginners of fluid mechanics find it challenging to solve PDE transformed ODE in the case of viscous fluid flows. Not only this, but the interpretation of results is also a must for better understanding a fluid flow problem. Though closed-form solutions to these problems exist, and the methodology is well established, the freshmen always struggle with ODE.

Here comes the role of programming to solve these ODE. Python is a straightforward and robust language that can quickly solve any ODE symbolically without any problem. Even implementing these algorithms is very easy and straightforward in Python due to its rich and adaptive libraries (modules) [3]. SymPy and Numpy modules are very effective for symbolic and numerical computations, whereas for visualization, matplotlib. pylab is a convenient module [4].

In this paper, the viscous parallel flow problems are solved using Python. The results are plotted and interpreted with the help of matplotlib. Finally, the programs developed to solve the problems are given so that the readers can benefit and apply these programs to solve further problems encountered in fluid flows.

SYMPY, NUMPY, AND MATPLOTLIB

Python has a powerful library for either numeric or symbolic computations. The symbolic calculations are done with the help of the SymPy [5] module, whereas for numerical computations, NumPy [6] is used. Matplotlib [7] is a handy tool for data plotting and visualization. These modules are open source and can be used freely for scientific computations. Both NumPy and Sympy are very powerful in algebra, discrete mathematics, calculus etc. [8], [9]. One of the attractive features of SymPy is its capability to format and present the results in LaTeX format.

In this research article, the governing equations are solved by using SymPy, and the results are presented using Matplotlib. Pylab.

II. PARALLEL FLOWS AND THEIR SOLUTION USING PYTHON

As it has been explained, in the case of parallel flows, the only velocity in one direction exists, and the other direction does not exist. Couette, Hagen Poiseuille, and Plane Poiseuille flow are the one that comes under the parallel flow category [10]. In this section, Couette and Hagen Poiseuille flows are modelled using Python. Assumptions used in developing the velocity profiles are:

- Viscous fluid flow
- The fluid flow is incompressible
- Parallel flow
- Fully developed flow
- Neglecting body force

A. Couette Flow

This is the flow of fluid between two plates in which the bottom plate is stationary, and the top plate moves with a constant velocity (u_0). Fig. 1 shows the Couette flow schematically.

International Journal of Engineering Applied Sciences and Technology, 2022
Vol. 6, Issue 11, ISSN No. 2455-2143, Pages 213-217
Published Online March 2022 in IJEAST (http://www.ijeast.com)

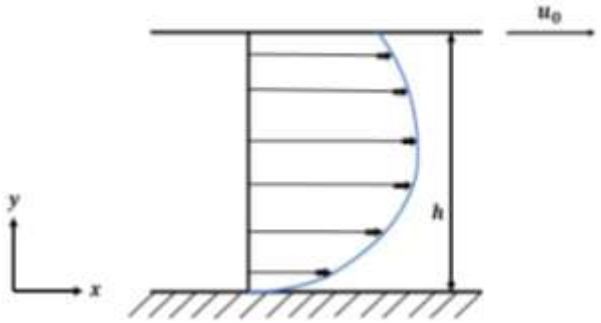

Fig. 1. Schematic of Couette flow

Let the Eulerian flow field velocity is given by:

$$\vec{V} = \hat{\imath}u + \hat{\jmath}v + \hat{k}w \quad (1)$$

As this is a parallel flow, it is assumed that $v = w = 0$ and $u \neq 0$. From the continuity equation, u velocity variation is given by:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0 \quad (2)$$

$$\frac{\partial u}{\partial x} = 0 \rightarrow u \neq u(x) \quad (3)$$

Moreover, if the plates are assumed to be infinitely large in the z-direction, then the variation of any parameter with respect to z will be negligible ($\frac{\partial (any\ variable)}{\partial z} = 0$); the above relation just boils down to:

$$u = u(y) \quad (4)$$

Therefore, the Navier-Stokes equations (NS) in the y and z directions will result in:

$$\frac{\partial p}{\partial y} = \frac{\partial p}{\partial z} = 0 \rightarrow p = p(x) \quad (5)$$

The NS equation in the x-direction will result in:

$$0 = -\frac{\partial p}{\partial x} + \mu \frac{\partial^2 u}{\partial y^2} \quad (6)$$

But as p is a function of x and u is a function of y, this partial differential equation (PDE) will get converted into an ordinary differential equation (ODE). Also, due to this type of functional relation, the terms become a constant.

$$\frac{dp}{dx} = \mu \frac{d^2 u}{dy^2} = constant \quad (7)$$

Eq. 7 will be solved using SymPy and plotted using NumPy with the boundary condition that $u(y = 0) = 0$ and $u(y = h) = u_0$. The detailed code is shown in Table 1, along with its output.

Table -1 Python Code for Couette Flow

Python code	**Program Output**

# Importing package 	
from sympy import *
from sympy.abc import *
 	
# Creating symbols for pressure gradient and top plate velocity 	
p_x,u_0=symbols('p_x, u_0')
 	
# Setting-up function 	
u=Function('u')(y)
 	
# Setting boundary condition 	
bc={u.subs(y,**0**):**0**,u.subs(y,h):u_0}
 	
# Forming differential equation 	
de=Eq(mu*u.diff(y,y),p_x)
 	
# Solving differential equation 	
sol=dsolve(de,ics=bc)
#simplify(expand(sol.rhs/u_0).coeff(p_x))
 	
# Obtaining final result by dividing it by normalizing it with u_0 	
u_u0=expand(simplify(sol.rhs.subs(p_x,-alpha***2***mu*u_0/h****2**))/u_0)
#pprint(u_u0)
u_u0
	Out[99]: $\frac{\alpha y}{h} - \frac{\alpha y^2}{h^2} + \frac{y}{h}$

Matplotlib.pylab has been used to plot the output, which is implemented via the following code:

International Journal of Engineering Applied Sciences and Technology, 2022
Vol. 6, Issue 11, ISSN No. 2455-2143, Pages 213-217
Published Online March 2022 in IJEAST (http://www.ijeast.com)

```
from pylab import *

def u_u0(α,h,y):
    return α*y/h-α*y**2/h**2+y/h

h=0.1
y=linspace(0,h,10)
figure(1,dpi=300)
for α in [-3,-2,-1,0,1,2,3]:
    plot(u_u0(α,h,y),y,'-o',label=f'α={α}')
xlabel('u/$u_0$')
ylabel('y/h')
legend()
show()
```

Fig. 2 shows the variation of non-dimensional velocity along the plate for different vales of α (Nondimensional pressure gradient = $-\frac{h^2}{2\mu u_0}\left(\frac{dp}{dx}\right)$) ranging from -3 to 3. The positive value of α means a favourable pressure gradient, and hence the flow velocity is in the forward direction. In contrast, the flow reversal takes place for negative values, and the onset of flow reversal happens at $\alpha = -1$, where it just starts.

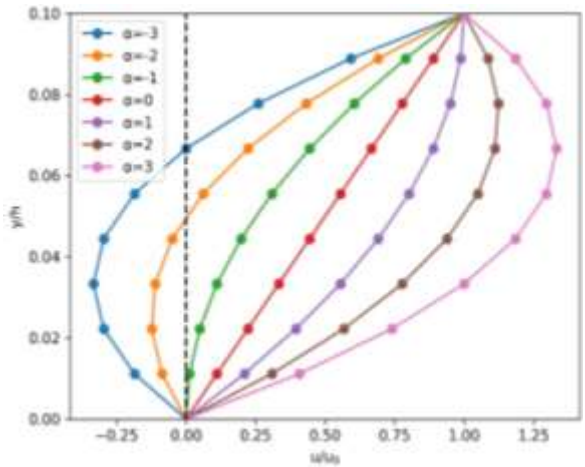

Fig. 2. Flow velocity distribution for Couette flow

B. Hagen Poiseuille Flow

This is a type of parallel flow that takes place in a pipe. The flow is shown schematically in Fig. 3.

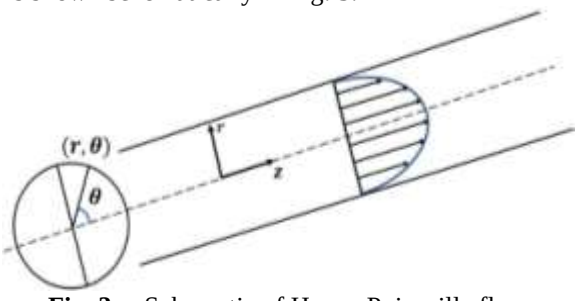

Fig. 3. Schematic of Hagen Poiseuille flow

To model the problem, the best coordinate system will be cylindrical. The velocity profile can be written as:

$$\vec{V} = \hat{e}_r v_r + \hat{e}_\theta v_\theta + \hat{e}_z v_z \quad (8)$$

As the flow is parallel so it can be assumed that $v_r = v_\theta = 0$ and $v_z \neq 0$. After applying the above assumption to the continuity equation, one can get:

$$v_z \neq v_z(z) \quad (9)$$

For axisymmetric flow, $\left(\frac{\partial(any\ variable)}{\partial\theta} = 0\right)$ the axial velocity is a function of r only. Hence $v_z = v_z(r)$.

Now NS equation for r and θ direction results in:

$$\frac{\partial p}{\partial r} = \frac{\partial p}{\partial \theta} = 0 \rightarrow p = p(z) \quad (10)$$

Therefore, the NS equation in the z-direction will result in:

$$0 = -\frac{\partial p}{\partial z} + \mu\left(\frac{\partial^2 v_z}{\partial r^2} + \frac{1}{r}\frac{\partial v_z}{\partial r}\right) \quad (11)$$

But as p is a function of z and v_z is a function of r, this partial differential equation (PDE) will get converted into (ODE). Also, due to this type of functional relation, the terms become a constant.

$$\frac{1}{\mu}\frac{dp}{dz} = \frac{d^2 v_z}{dr^2} + \frac{1}{r}\frac{dv_z}{dr} = constant \quad (12)$$

Eq. 12 will be solved using SymPy and plotted using NumPy with the boundary condition that $v_z(r = R) = 0$ and $\frac{dv_z(r=0)}{dr} = 0$. The detailed code and output are shown in Table 2.

Python code	**Program Output**

# Importing package 	
from sympy import *
from sympy.abc import *
 	
# Creating symbols for pressure gradient and flow velocity 	
v_z,p_z,v_max=symbols('v_z,p_z,v_max')
 	
# Setting-up function 	
v_z=Function('v_z')(r)
 	
# Setting boundary condition 	
bc={v_z.subs(r,R):**0**,v_z.diff(r).subs(r,**0**):**0**}
 	
# Forming differential equation 	
	Out[6]: $1 - \frac{r^2}{R^2}$

International Journal of Engineering Applied Sciences and Technology, 2022
Vol. 6, Issue 11, ISSN No. 2455-2143, Pages 213-217
Published Online March 2022 in IJEAST (http://www.ijeast.com)

```
eqn=Eq(v_z.diff(r,r)+(1/r)*v_z.diff(r),(1/mu)*p_z)

# Solving differential equation
v_v0=simplify(dsolve(eqn,ics=bc).rhs)

# Obtaining maximum velocity
v_max=v_v0.subs(r,0)

# Normalizing the velocity by maximum velocity
v_v0=v_v0/v_max

simplify(v_v0)
u_u0
```

The flow velocity variation is shown in Fig. 4. One can notice that the velocity variation is parabolic, with a maximum at the centre for a fully developed case.

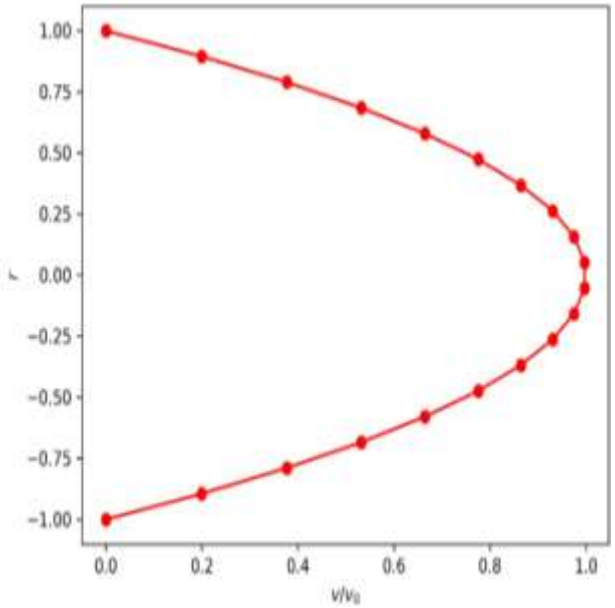

Fig. 4. Velocity variation in Hagen poiseuille flow

The code with which the above velocity variation has been developed is shown below:

```
from pylab import *

R=1.0
def v_v0(r,R):
    return 1-r**2/R**2

figure(1,dpi=300)

r=linspace(-R,R,20)
plot(v_v0(r,R),r,'r-o')
xlabel('$v/v_0$')
ylabel('$r$')
savefig('Hagen_poise.jpg')
show()
```

III. CONCLUSION

In this research article, a class of parallel flows viz. Couette and Hagen Poiseuille flow have been solved using Python. SymPy and NumPy modules were used to solve the differential equations and plot the results. The research article will help beginners of fluid mechanics to solve differential equations using Python, and they will be able to interpret the results by following the procedure mentioned.

IV. REFERENCE

[1] A. Majda, A. Bertozzi, and A. Ogawa, "Vorticity and Incompressible Flow. Cambridge Texts in Applied Mathematics," Appl. Mech. Rev., vol. 55, no. 4, pp. B77–B78, 2002.

[2] F. M. White, Viscous flow theory. McGraw-Hill, New York, 1974.

[3] A. Bäcker, "Computational physics education with python," Comput. Sci. Eng., vol. 9, no. 3, pp. 30–33, 2007.

[4] M. Cywiak and D. Cywiak, "SymPy," in Multi-Platform Graphics Programming with Kivy: Basic Analytical Programming for 2D, 3D, and Stereoscopic Design, Berkeley, CA: Apress, 2021, pp. 173–190.

[5] A. Meurer et al., "SymPy: Symbolic computing in python," PeerJ Comput. Sci., vol. 2017, no. 1, pp. 1–27, 2017.

[6] S. Van Der Walt, S. C. Colbert, and G. Varoquaux, "The NumPy array: A structure for efficient numerical computation," Comput. Sci. Eng., vol. 13, no. 2, pp. 22–30, 2011.

[7] E. Bisong, "Matplotlib and Seaborn," in Building Machine Learning and Deep Learning Models on Google Cloud Platform, Berkeley, CA: Apress, 2019, pp. 151–165.

[8] M. Rocklin and A. R. Terrel, "Symbolic statistics with SymPy," Comput. Sci. Eng., vol. 14, no. 3, pp. 88–93, 2012.

[9] P. S. Pawar, D. R. Mishra, and P. Dumka, "Solving First Order Ordinary Differential Equations using Least

Square Method : A comparative study," Int. J. Innov. Sci. Res. Technol., vol. 7, no. 3, pp. 857–864, 2022.

[10] G. Biswas and S. K. Som, Introduction to Fluid Mechanics and Fluid Machines, no. March. Tata McGraw-Hill Education, 2003.