Research Paper | Engineering | E-ISSN No : 2454-9916 | Volume : 8 | Issue : 10 | Oct 2022

MODELLING PIPE FLOW USING PYTHON

*Pankaj Dumka[1], Krishna Gajula[1], Vikrant Sharma, Dhananjay R. Mishra[1]

[1]Department of Mechanical Engineering, Jaypee University of Engineering and Technology, Guna, Madhya Pradesh, India.

ABSTRACT

In every wake of life, the flow of fluids through pipes is encountered. The major problem encountered while analysing pipe flow problems is obtaining friction factor. Though Moody's diagram helps evaluate the friction factor, the obtained solution is error-prone due to errors in reading the graph. So, to remove the mistakes using hand calculations and improper use of diagram, an attempt has been made in this research article to automate the process of pipe flow modelling. The Colebrook-White equation has been iteratively solved to obtain the friction factor. The modelling is done using Python as it is easy to use and has a vast library backup. The robustness of the developed program has been demonstrated by plotting Moody's diagram using the code. Three different pipe flow problems through a single pipeline are solved using the developed program, and the obtained results are precisely equal to those shown in the literature.

KEYWORDS: Pipe flow; Friction factor; Python Programming; Moody's diagram; Reynolds number.

Nomenclature:

A	Cross-sectional area of pipe (m^2)
c_f	Fanning's friction factor
d	Pipe diameter (m)
D_h	Hydraulic diameter (m)
L	Pipe length (m)
Q	Volume flow rate (m^3/s)
R_e	Reynolds number
v	Average flow velocity
f	Darcy's Friction factor
ρ	Fluid density (kg/m^3)
Δp	Pressure drop (Pa)
ϵ	Wall roughness (m)

1. INTRODUCTION:

Understanding the fluid flow regime is of utmost importance from the engineering point of view when a fluid flows. Reynold's number (Re) [1], which is the ratio of inertia to viscous force, talks about the flow regime, i.e., whether it is laminar or turbulent [2]. For Pipe flow, if the Re is less than 2000, it is called laminar, whereas when it is more than 4000, it is called turbulent flow [3]. Laminar flow is where laminas of fluid glide smoothly one over the other. In contrast, the turbulent flow is the one intermingling of fluid laminas. In other words, turbulent flow can be said to be where the flow becomes an irregular function of time [4]. And as the flow velocities in turbulent flows become periodic with respect to the time, we cannot have a steady flow. On the other hand, these fluctuations are random, so a statistical description is possible. The variation may be brought down by the law of probability so that an average velocity is defined [5].

Usually, all flows in nature are turbulent, so laminar flow occurs with minimal velocities, which are not usually encountered in practical applications. So the most vital thin in turbulent flow is that the fluid friction viz. layer to layer and between fluid and sold friction gets enhanced [6]. Which means it offers more resistance for the given type of fluid. This means for a given flow rate through a duct, the pressure drop across a length of a duct is much more than that of a laminar flow. For a fluid flow in a pipe (Figure 1), the wall shear stress (τ_w) can be evaluated as [7]:

$$\tau_w = \frac{1}{4}\frac{d}{L}\Delta p \quad \text{..... (1)}$$

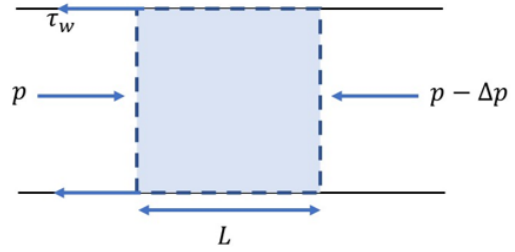

Figure 1: Force balance of a fluid element in a pipe flow

This means that stress is responsible for the pressure drop in the pipe. Now, if the above expression is replaced in the skin friction coefficient (c_f), then it is also called the Fannings friction factor [7], as shown in Eq (2).

$$c_f = \frac{1}{4}\frac{d}{L}\frac{\Delta p}{\left(\frac{1}{2}\rho v^2\right)} \quad \text{..... (2)}$$

Where v is the average flow velocity ($v=Q/A$), Darcy defined his friction factor as four times that of Fanning's viz [8].

$$f = 4 \times c_f \quad \text{..... (3)}$$

Therefore, the pressure drops (Δp) across a pipe length L in a fluid of density ρ and velocity v is given by:

$$\Delta p = f\frac{L}{d}\frac{1}{2}\rho v^2 \quad \text{..... (4)}$$

And the power to be supplied to the fluid (of flow rate Q) to overcome is pressure drop is given by:

$$Power = \Delta p \times Q \quad \text{..... (5)}$$

If the flow is laminar, one can quickly get the relation between friction factor f and Re, as shown in Eq. (6).

$$f = \frac{64}{Re} \quad \text{.....(6)}$$

But when the flow becomes turbulent, friction between the fluid and solid wall and within the fluid enhances due to mixing in the turbulent zone. So due to a rise in flow resistance, the pressure drop gets enhanced. But there is not a single equation by which one can fit the pressure drop and flow rate relation or f-Re relations through the entire range of turbulent flow regimes. So, to handle such situations, experiments come in. People like Stanton [9], Nikuradse [9,10], etc., have worked on developing such expressions, and finally, Moody documented them in the form of a diagram which is famously known as Moody's diagram [11]. They have observed that in turbulent flow, the f is not the only function of Re but wall roughness as well. But reading this diagram requires a great deal of interpolation, which can result in an error. So, to help the researchers working in the field of pipe flow, some sort of computations is required, which will save time and make the error of the computation free.

Python is an easy programming language with a light and user-friendly syntax [12–15]. Moreover, when it comes to numerical computations, its modules viz. NumPy and SymPy are of great help [16–20]. NumPy has a multi-dimensional array and matrix data structure and has compact storage. Moreover, when it comes to loops, the NumPy is swift. Moreover, the slicing aspect of the array is compelling when it comes to numerical computations. Matplotlib.pylab [21] is a convenient library for data plotting. Pylab also uses NumPy, so while using Pylab, calling NumPy is not required.

In this research article, a python-based approach has been used to solve the friction factor and the development of Moody's diagram. Also, three fundamental pipe flow problems are solved using the functions developed in Python.

2. COLEBROOK-WHITE EQUATION AND THE EVALUATION OF FRICTION FACTOR:

For fully developed turbulent flow in a pipe, the Darcy friction factor (f) can be approximated with the help of the Colebrook-White Equation [22], which is as follows:

$$\frac{1}{\sqrt{f}} = -2\log\left(\frac{\epsilon}{2.7\,D_h} + \frac{2.51}{Re\sqrt{f}}\right) \quad \text{.....(7)}$$

Where D_h is the hydraulic diameter which gets reduced to the pipe diameter d for the circular pipe. Eq. (7) is an implicit equation which cannot be directly solved for the f. So, an iterative procedure can be adapted to solve for f. The steps to solve the Eq. (7) implicitly are as follows:

i. Rearrange the Colebrook equation into the following form:

$$f = \left(\frac{1}{-2\log\left(\frac{\epsilon}{3.7d} + \frac{2.51}{Re\sqrt{f}}\right)}\right)^2 \quad \text{.....(8)}$$

ii. Choose the initial guess for f (say f_g). For a better guess, use a value between 0.1 and 0.01, as this is the range of f.

iii. Solve Eq. (8) for new f (say f_n) by using f_g as:

$$f_n = \left(\frac{1}{-2\log\left(\frac{\epsilon}{3.7d} + \frac{2.51}{Re\sqrt{f_g}}\right)}\right)^2 \quad \text{.....(9)}$$

iv. Obtain the error ($|f_g - f_n|$) and check for convergence.

v. Update the f_g by obtained value f_n as a new guess for the next iteration.

vi Repeat steps iii to v till the convergence is achieved.

3. IMPLEMENTATION OF COLEBROOK EQUATION VIA PYTHON:

The algorithm discussed in the above section has been implemented in Python via following function:

```
def f_f(ϵ,d,Re,f_g):
    if Re<2000:
        f_n=64/Re
    else:
        error=1
        count=0
        ϵ_d=ϵ/d
        while error>1.E-5:
            count+=1

            # Evaluation of new value of f
            a=ϵ_d/(3.7)+2.51/(Re*sqrt(f_g))
            f_n=(1/(-2*log10(a)))**2

            # Error calculation
            error=abs(f_n-f_g)

            # Guess updation
            f_g=f_n

    return f_n
```

First, Re is checked; if it is less than 2000 then Eq. (6) is used to evaluate the f, whereas the Colebrook equation is solved for other values. Moreover, one can also observe that in the equation, relative roughness is used, which is the ratio of wall roughness to that of pipe diameter (ϵ/d). Therefore, to use this function, one has to supply the wall roughness, pipe diameter, Re, and guess value for friction factor as shown in the below-mentioned code snippet.

```
# Wall roughness
ϵ=0.15*10**(-3)
# Pipe diameter
d=150*10**(-3)
# Flow Reynolds number
Re=5000
# Initial guess supplied
f_g=0.1
# Function Call
fric_fact=f_f(ϵ,d,Re,f_g)
# Output display
print(f'f={round(fric_fact,6)}')
```

The output of the above code is $f = 0.038496$, which is the exact value of the friction factor for the given arguments.

One can also go one step further and plot the value of f for different Re and ϵ/d. Below code will produce the result, which is nothing, but Moody's diagram and Figure 2 shows the diagram obtained from the code. One can observe from the plot that for every roughness in the turbulent region, there exists a Re beyond which the dependence of f on Re ends, i.e., the line becomes horizontal.

```
from pylab import *

# Range of wall roughness
ϵ=linspace(1.E-8,0.004,10)
d=0.05
#Range of Reynolds number
R=arange(700,1.E7,100)

figure(2,dpi=300)
for eps in ϵ:
    fric_fact=empty(len(R))
    f_g=1
    for i,Re in enumerate(R):
        if Re==2000:
            fric_fact[i]=100 # Some arbitrary value (very high)
        else:
            fric_fact[i]=f_f(eps,d,Re,f_g)

    semilogx(R,fric_fact,'-',label=f'ϵ/d={eps}')
    if eps/d<0.1:
        text(R[-1],fric_fact[-1],f'  {round(eps/d,7)}')
text(R[-1],ϵ[-1]/d+0.015,'   ϵ/d')

xlabel('Re',fontsize=14)
ylabel('f',fontsize=14)
ylim(0.008,0.1)
#grid('--')
for pos in ['right', 'top']:
    gca().spines[pos].set_visible(False)

savefig("Moody's.jpg")
show()
```

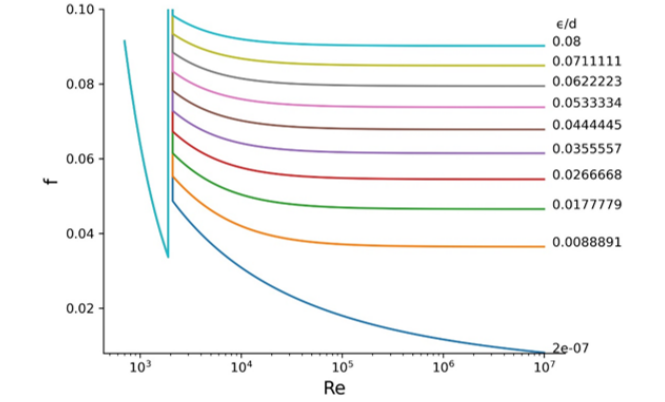

Figure 2: Friction factor vs Re for different values of relative roughness.

4. CLASSES OF PROBLEMS TYPICALLY ENCOUNTERED IN A FLUID FLOW IN A SINGLE PIPELINE:

When flow through a single pipeline happens then, three distinct types of problems usually are arisen, which are listed below:

I. The pipe diameter, pipe length, and flow rates are given, and one has to find

the pressure drop and corresponding pumping power required.

ii. Head loss over a pipe length for a pipe of known diameter is given; the target is to find the flow rate and power required.

iii. The flow rate and the head loss over a given pipe length are known the pipe diameter has to be evaluated.

The first class of problem is explicit, and one can quickly solve the pressure drop and power using Eq. (4) and (5). However, the second and third types of problems cannot be solved explicitly as velocity and friction factors are unknown in the second, and diameter, and friction factors are unknown in the third class. So, these problems can only be solved by iteration procedure. To solve these problems with the help of Python, following apart for the function to evaluate friction factor, the following more functions are created to ease up the problem solving:

This function evaluates $\Delta h, \Delta p$ and accept $f, L, d\ \rho, v, g$ as input arguments.	`def press_drop(f,L,d,ρ,v,g):`
	`  dp=f*(L/d)*(ρ*v**2)/2`
	`  dh=dp/(ρ*g)`
	`  return dp,dh`

This function evaluates power with Δp and Q as input	`def Power(a,b):` `  return a*b`
This function returns flow velocity when discharge and diameter are given	`def vel_Q(Q,d):` `  return Q/((pi/4)*d**2)`
This function returns Re for given $\rho, v, d,\ \mu$	`def Rel_num(ρ,v,d,μ):` `  return ρ*v*d/μ`
This function returns Re for given $v, d,\ \nu$	`def Rel_num_ν(v,d,ν):` `  return v*d/ν`
This function returns velocity when Δh is given f are known	`def v_Δh(Δh,d,L,g,f_g):` `  return sqrt(Δh*(d/L)*2*g/f_g)`

This function returns diameter when Q and Δh are known	`def d_Q_Δh(Q,Δh,L,g,f_g):`
	`  return`
	`((f_g/Δh)*(L/(2*g))*(4/pi)**2*Q**2)**(1/5)`

Now let us demonstrate how these problems can be solved using the above python functions.

i. Determine the frictional head loss and pumping power required when water flows through a 300 m long steel pipe of 150 mm diameter at a flow rate of 0.05 m^3/s. The kinematic viscosity of water and the wall roughness of the pipe are $\nu = 1.14\times10^{-6}$ mm and $\epsilon = 0.15$ mm [23].

Python Solution:

```
from pylab import *
# Case I
#(Given: Q, d, L)
g=9.81
L=300.0
d=150*10**(-3)
Q=0.05
ν=1.14*10**(-6)
ϵ=0.15*10**(-3)
ρ=1.0*10**3
μ=ρ*ν

v=vel_Q(Q,d)

Re=Rel_num(ρ,v,d,μ)

friction_factor=f_f(ϵ,d,Re,0.1)
#print('f = ',f)
Δp,Δh=press_drop(friction_factor,L,d,ρ,v,g)

print('Δh = ',round(Δh,3), 'm')

p=Power(Δp,Q)
print('Power = ',round(p,3), 'W')
```

The output of the program is: $\Delta h = 16.67$ m and Power = 8176.72 W

ii. Oil of kinematic viscosity of $10^{-5}\ m^2/s$ flows at a steady state through a pipe of diameter 100 mm and of surface roughness of 0.25 mm. For a pipe length of 120 m, the friction head loss is 5 m. What is the flow rate of oil through the pipe [23].

Python solution:

```
from pylab import *
# Case II
# (Given: Δh, L, d)
ν=10**(-5)
d=100*10**(-3)
ϵ=0.25*10**(-3)
L=120.0
Δh=5

# initial guess of friction factor
f_g=0.1

error=1
count=1
while error > 1.E-5:
    v=sqrt(Δh*(d/L)*2*g/f_g)
    Re=Rel_num_ν(v,d,ν)
    f_new=f_f(ϵ,d,Re,f_g)

    error=abs(f_new-f_g)
    f_g=f_new

    count=count+1

v=sqrt(Δh*(d/L)*2*g/f_new)
Q=(pi/4)*d**2*v
print('iteration = ',count)
print('Discharge = ',Q)
```

The program output is: Q = 0.0126 m^3/s.

iii. Determine the size of a pipe (ϵ=0.15 mm) needed to transmit water (ν=1.14×10^{-6} m^2/s) to a distance of 180 m at 0.85 m^3/s with a loss of head of 9 m [23].

Python solution:

```
Case III
# (Given: Δh, Q, L)
L=180.0
Q=0.085
Δh=9.0
ν=1.14*10**(-6)
ϵ=0.15*10**(-3)
g=9.81

f_g=0.1

error=1
count=0
while error>1.E-5:
    d=d_Q_Δh(Q,Δh,L,g,f_g)
    v=vel_Q(Q,d)
    Re=Rel_num_ν(v,d,ν)
    f_new=f_f(ϵ,d,Re,f_g)
    error=abs(f_g-f_new)

    f_g=f_new
    count=count+1
    print(count,error)

print('final diameter =',d)
```

The output of the program is: d=0.1872987 m.

It has been observed that the results obtained by the computer programs are the same as those obtained in the literature [23].

5. CONCLUSION:

In this manuscript, pipe flow modelling has been done using Python. A detailed explanation of the pipe friction factor has been done. A precise algorithm has been explained to obtain the friction factor in the turbulence zone using the Colebrook-White Equation. Moreover, three classes of fluid flow problems in a single pipeline were presented and solved using the functions developed in Python. The capacity of function to obtain friction factor has also been used to plot Moody's diagram. The methodology provided in this article will help practice engineers and researchers solve pipe flow problems without errors. The developed computer programs are very robust and can be extended to solve any flow problem.

REFERENCES:

I. N. Rott, Note on the History of the Reynolds Number, Annu. Rev. Fluid Mech. 22 (1990) 1–12. doi:10.1146/annurev.fl.22.010190.000245.

II. M.T. Schobeiri, Basic Physics of Laminar-Turbulent Transition, in: Turbomach. Flow Phys. Dyn. Perform., Springer Berlin Heidelberg, Berlin, Heidelberg, 2012: pp. 515–544. doi:10.1007/978-3-642-24675-3_19.

III. G. Biswas, V. Eswaran, Turbulent flows: fundamentals, experiments and modeling, CRC Press, 2002.

IV. A. Cioncolini, L. Santini, An experimental investigation regarding the laminar to turbulent flow transition in helically coiled pipes, Exp. Therm. Fluid Sci. 30 (2006) 367–380. doi:https://doi.org/10.1016/j.expthermflusci.2005.08.005.

V. Y. Zhou, Turbulence theories and statistical closure approaches, Phys. Rep. 935 (2021) 1–117. doi:https://doi.org/10.1016/j.physrep.2021.07.001.

VI. E.A. Al-Khdheeawi, D.S. Mahdi, Apparent viscosity prediction of water-based muds using empirical correlation and an artificial neural network, Energies. 12 (2019) 1–10. doi:10.3390/en12163067.

VII. B. Guerrero, M.F. Lambert, R.C. Chin, Extreme wall shear stress events in turbulent pipe flows: spatial characteristics of coherent motions, J. Fluid Mech. 904 (2020) A18. doi:10.1017/jfm.2020.689.

VIII. H.A. Milukow, A.D. Binns, J. Adamowski, H. Bonakdari, B. Gharabaghi, Estimation of the Darcy–Weisbach friction factor for ungauged streams using Gene Expression Programming and Extreme Learning Machines, J. Hydrol. 568 (2019) 311–321. doi:https://doi.org/10.1016/j.jhydrol.2018.10.073.

IX. J. Nikuradse, Regularity of turbulent flow in smooth pipes, 1949.

X. J. Nikuradse, others, Laws of flow in rough pipes, (1950).

XI. M. LaViolette, On the History, Science, and Technology Included in the Moody Diagram, J. Fluids Eng. 139 (2017). doi:10.1115/1.4035116.

XII. P. Dumka, S. Sharma, H. Gautam, D.R. Mishra, Finite Volume Modelling of an Axisymmetric Cylindrical Fin using Python, Res. Appl. Therm. Eng. 4 (2021) 1–11.

XIII. Y.C. Huei, Benefits and introduction to python programming for freshmore students using inexpensive robots, in: Proc. IEEE Int. Conf. Teaching, Assess. Learn. Eng. Learn. Futur. Now, TALE 2014, 2015: pp. 12–17. doi:10.1109/TALE.2014.7062611.

XIV. G. Moruzzi, Python Basics and the Interactive Mode, in: Essent. Python Phys., Springer International Publishing, Cham, 2020: pp. 1–39. doi:10.1007/978-3-030-45027-4_1.

XV. P. Dumka, A. Singh, G.P. Singh, D.R. Mishra, Kinematics of Fluid : A Python Approach, Int. J. Res. Anal. Rev. 9 (2022) 131–135.

XVI. M. Cywiak, D. Cywiak, SymPy, in: Multi-Platform Graph. Program. with Kivy Basic Anal. Program. 2D, 3D, Stereosc. Des., Apress, Berkeley, CA, 2021: pp. 173–190. doi:10.1007/978-1-4842-7113-1_11.

XVII. A. Meurer, C.P. Smith, M. Paprocki, O. Čertík, S.B. Kirpichev, M. Rocklin, Am.T. Kumar, S. Ivanov, J.K. Moore, S. Singh, T. Rathnayake, S. Vig, B.E. Granger, R.P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M.J. Curry, A.R. Terrel, Š. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman, A. Scopatz, SymPy: Symbolic computing in python, PeerJ Comput. Sci. 2017 (2017) 1–27. doi:10.7717/peerj-cs.103.

XVIII. R. Johansson, Numerical python: Scientific computing and data science applications with numpy, SciPy and matplotlib, Second edition, Apress, Berkeley, CA, 2018. doi:10.1007/978-1-4842-4246-9.

XIX. P.S. Pawar, D.R. Mishra, P. Dumka, M. Pradesh, Obtaining Exact Solutions of Visco- Incompressible Parallel Flows Using Python, Int. J. Eng. Appl. Sci. Technol. 6 (2022) 213–217.

XX. P. Dumka, P.S. Pawar, A. Sauda, G. Shukla, D.R. Mishra, Application of He's homotopy and perturbation method to solve heat transfer equations: A python approach, Adv. Eng. Softw. 170 (2022) 103160. doi:10.1016/j.advengsoft.2022.103160.

XXI. E. Bisong, Matplotlib and Seaborn, in: Build. Mach. Learn. Deep Learn. Model. Google Cloud Platf., Apress, Berkeley, CA, 2019: pp. 151–165. doi:10.1007/978-1-4842-4470-8_12.

XXII. D.I.H. Barr, C. White, A.A. Smith, T.E. Stanton, Application of Similitude Theory to Correlation of Uniform Flow Data., Proc. Inst. Civ. Eng. 37 (1967) 487–509.

XXIII. G. Biswas, S.K. Som, Introduction to Fluid Mechanics and Fluid Machines, Tata McGraw-Hill Education, 2003.