SSRG International Journal of Mechanical Engineering
ISSN: 2348 – 8360 / https://doi.org/10.14445/23488360/IJME-V10I2P101

Volume 10 Issue 2, 1-8, February 2023

Original Article

Modelling of Hardy Cross Method for Pipe Networks

Pankaj Dumka[1], Nitin Samaiya[2], Sumit Gandhi[3], Dhananjay R. Mishra[4]

[1,4]*Department of Mechanical Engineering, Jaypee University of Engineering and Technology, Guna, Madhya Pradesh.*
[2,3]*Department of Civil Engineering, Jaypee University of Engineering and Technology, Guna, Madhya Pradesh.*

Received: 16 December 2022 Revised: 18 January 2023 Accepted: 04 February 2023 Published: 15 February 2023

Abstract - *The transport of fluids through pipes is very common in core engineering practice. The main issue that came up when the pipe flow networks were its analysis part. The Hardy Cross approach is very accurate and reliable for solving these issues, but because it is iterative, the likelihood of errors increases as the number of circuit loops grows. Therefore, in this research, a piece of effort has been made to automate the Hardy Cross technique using Python programming (as it is user-friendly and has a large library backup) to remove the errors that come with using hand calculations. The built program has been applied to four different pipe flow network problems, and the outcomes are the same as those presented in the literature.*

Keywords - *Pipe flow, Pipe Network, Python Programming, Hardy Cross method*.

1. Introduction

One of the most difficult tasks faced by the practitioners and students of fluid dynamics is understanding, analysing, and solving the pipe network problem [1]. A pipe network is a very complex arrangement of different pipes (in series and parallel) with varying resistances of flow and flow rates [2], as seen in Fig. 1.

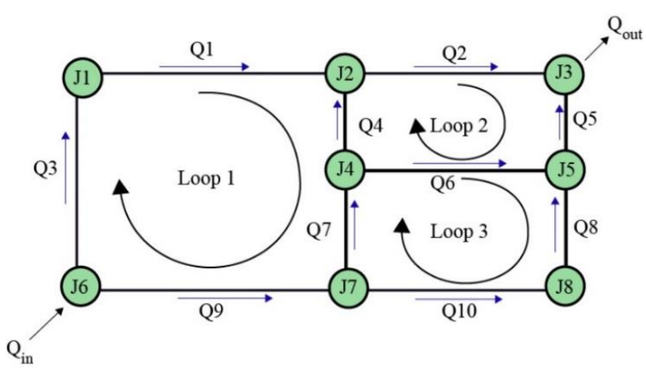


Fig. 1 A typical pipeline network [3]

The task is to find the flow rate in each pip so that the continuity equation and pressure equations satisfy simultaneously [4]. The meaning of continuity is that the sum of discharge at each node has to be zero, whereas the pressure means that in each hydraulic loop, the sum of the product of flow resistance and square of discharge should be zero. Mathematically these can be written as [4], [5]:

$$\sum Q = 0 \tag{1}$$

$$\sum R|Q|Q = 0 \tag{2}$$

The pipe network analysis cannot be done explicitly as at each node of the network, Eq. 1 has to satisfy, and Eq. 2 has to be satisfied in each loop. So the network can only be solved iteratively. The best-known method so far is called the Hardy-Cross method for pipe flow networks. Nevertheless, this problem is the complex nature of the iterations involved, which grows as the network complexity. In some cases, it becomes very difficult, trying and error-prone to apply the method to the pipe network using hand calculations. Therefore it becomes essential to apply the method with the help of computer programming [27].

Therefore python catches the eyes of researchers as it is an easy programming language with a light and user-friendly syntax[7]–[14]. Furthermore, its modules NumPy and SymPy are quite useful for numerical computations [12], [15]–[20]. A multi-dimensional array, matrix data structure, and small storage are features of NumPy. Moreover, NumPy is quick when it comes to loops. Matplotlib. Pylab [21]–[23] is a very useful library for plotting data. 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 pipe network problems. The Hardy Cross method has been modelled as functions in python. Moreover, four complex pipe networks are solved with the help of developed Python functions and code.

2. Hardy Cross Method

The solution of Eq. 1 and 2 is done by the Hardy cross method (which is an iterative process) by setting up an initial guess (Q) in the whole pipe network [24]–[26]. Let Q_0 is the correct flow rate, and Q is the guess. Then the error in the flow rate can be written as:

$$dQ = Q - Q_0 \tag{3}$$

It is assumed that the error is constant for all hydraulic paths in a loop (dQ = const). Let head drop in a particular element of the loop be h h' based on Q and Q_0 as shown below:

$$\boldsymbol{h} = R|Q|Q \tag{4}$$

$$\boldsymbol{h'} = R|Q_0|Q_0 \tag{5}$$

Then according to Eq 1. in a loop, the above equations become:

$$\sum h = error \tag{6}$$

$$\sum h' = 0 \tag{7}$$

Eq. 6 gives the error due to the assumed values of the flow rate. On combining Eq. 6 and 7, one can get:

$$\sum(h - h') = \sum(dh) = error \tag{8}$$

Where dh is the error in the pressure equation for a path. On differentiating Eq. 4, the value of dhcan be obtained in terms of dQas:

$$dh = 2R|Q|dQ \tag{9}$$

now substituting dh in Eq. 8, one can get:

$$error = \sum 2R|Q|dQ \tag{10}$$

As it is assumed that dQ is a constant so it can be taken out from the summation as:

$$dQ = \frac{error}{\sum 2R|Q|} \tag{11}$$

Therefore, from Eq. 7, 8, and 4, one can get the error in flow rate as follows:

$$dQ = \frac{\sum R|Q|Q}{\sum 2R|Q|} \tag{12}$$

The Hardy cross methodology is explained below:

1. The flow rate in each pipe of the network is assumed so that the continuity equation, i.e. Eq. 1, is satisfied. To reduce the number of iterations, one can set lower values of flow rates for the pipes with high resistance and vice-versa.
2. With the assumed flow rates, evaluate the error in discharge (for each loop) using Eq. 12.
3. Check whether the error is less than a certain threshold value or not. If yes, then the final answer is arrived and break the loop; else, follow the steps below.
4. Then update the discharge using Eq. 3 for each loop.
5. Repeat steps from 2 to 4.

3. Implementation of Hardy Cross Method in Python

First, the function is developed, which will accept the array of Resistances and assumed guesses (as arguments) in each loop. In the article, the function is called " Hardy_Cross". This function evaluates dQ based on Eq. 12 and returns the updated discharge equation in each loop. The function is as follows:

```
def Hardy_Cross(R,Qg):
    """
    Function to evaluate corrected 
discharge in a loop
    Input: Array of R and assumed 
discharge (Q)
    Output: Updated discharge
    """
    numerator=sum(R*abs(Qg)*Qg)
    denominator=sum(2*R*abs(Qg))
    dQ=numerator/denominator
    Q=Qg-dQ
  return Q
```

This function will be called in the main program, where arrays of resistances and respective assumed flow rates are defined. Then for a particular loop above function will return the new updated value of discharge for a particular iteration.

Let say R1 and Q1 are the arrays for loop 1, and R2 and Q2 are arrayed for loop 2. Also, let us assume these have P_c as common pipe shown in Fig. 2.

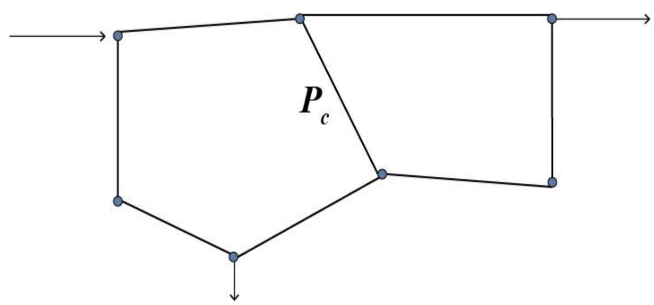


Fig. 2 Common pipe in a pipe network

Once the updated discharge values are obtained for loop 1 then, before solving for loop 2, the value of discharge corresponding to the pipe P_c for loop 2 has to be updated from the new values of loop 1. Also, after the updated value of discharge from loop 2 is obtained, the guess for pipe P_c for loop1 again has to be updated. This is done by using the index of pipe P_c from the discharge array. This is done as follows:

```
#---- LOOP 1 -----#
# Hardy Cross function
Q1= Hardy_Cross(R1,Q1g)

# Error for loop 1
e1=abs(Q1-Q1g)

# Updating Q1g guess
Q1g=Q1.copy()

#---- LOOP 2 -----#
# Common Pipe
Q2g[1]=Q1g[1]

# Hardy Cross function
Q2= Hardy_Cross(R2,Q2g)

# Error for loop 1
e2=abs(Q2-Q2g)

# Updating Q2g guess
Q2g=Q2.copy()

# Common Pipe
Q1g[1]=Q2g[1]
```

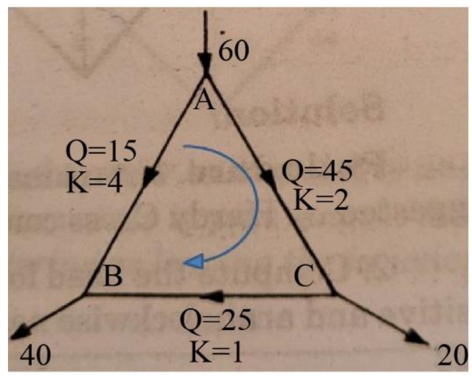
Also, the loop error has to be evaluated simultaneously for each loop and checked for the given convergence criterion in the while loop. The while loop will be as follows:

```
# Initial error value to enter in
the loop
error=1
# Iteration counter
count=1

while error>1.E-8:

#--------- LOOP 1 ----------#
# Hardy Cross function call
    Q1= Hardy_Cross(R1,Q1g)

    # Error for loop 1
    e1=abs(Q1-Q1g)

    # Updating Q1g guess
    Q1g=Q1.copy()

#--------- LOOP 2 ----------#
    # Common Pipe
    Q2g[1]=Q1g[1]

# Hardy Cross function call
    Q2= Hardy_Cross(R2,Q2g)

    # Error for loop 1
    e2=abs(Q2-Q2g)

    # Updating Q2g guess
    Q2g=Q2.copy()

    # Common Pipe
    Q1g[1]=Q2g[1]

    # Error Evaluation
    error=sum(e1**2+e2**2)

    # Loop counter increment
  count+=1
```

4. Implementation of Python Functions to Solve Problems

In this section, some numerical problems will be taken up to show the accuracy and ease with which complex pipe flow problems can be done.

Question 1: Solve below mentioned network :



In this problem, only one loop is there, so the first task is to guess the values of flow rates in each pipe of the loop. The guess values are written on each pipe along with the respective resistances (K). Assume a clockwise direction of the loop the array of resistances and initial discharge will be; $\boldsymbol{R} = [\mathbf{2}, \mathbf{1}, \mathbf{4}]$ and $\boldsymbol{Q_g} = [\mathbf{45}, \mathbf{25}, -\mathbf{15}]$. The negative sign of discharge in pipe of resistance 4 is taken; as in a clockwise direction, the assumed motion of fluid will be opposite to the loop direction. The detailed program will be as follows:

Python Modules and Function

```
from numpy import *
def Hardy_Cross(R,Qg):
    """
    Function to evaluate corrected
discharge in a loop
    Input: Array of R and assumed
discharge (Q)
    Output: Updated discharge
    """
    numerator=sum(R*abs(Qg)*Qg)
```

```
    denominator=sum(2*R*abs(Qg))
    dQ=numerator/denominator
    Q=Qg-dQ
    return Q
```

Main Program

```
R=array([2,1,4])
Qg=array([45,25,-15])

error=1
count=1

while error>1.E-10:
#---- LOOP 1 -----#
    Q= Hardy_Cross(R,Qg)

    e=abs(Q-Qg)

    # Updating Q1g guess
    Qg=Q.copy()

    error=sqrt(sum(e**2))

    count+=1
print(Q)
```

Data display

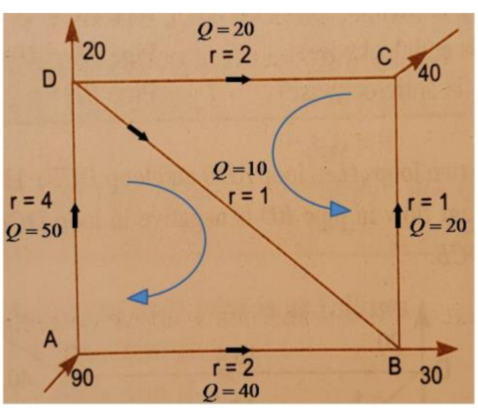
```
from pandas import *
pipe=array(["AC","CB","BA"])
data={'Pipe':pipe,'Q':Q}
df=DataFrame(data)
df
```

The output of the following program will be:

	Pipe	Q
0	AC	34.52763
1	CB	14.52763
2	BA	-25.47237

The negative value tells that the discharge is opposite to the assumed loop's direction.

Question 2: Solve below mentioned network :



There are two loops, i.e. an extra loop compared to the p revious question. Here also, the first thing which we will be doing is selecting guess values for each pipe considering Eq. 1. The initial guess is written along each pipeline.

	Pipes	R	Q_g
Loop1	AD DB BA	[4,1,2]	[50,10, −40]
Loop2	CD DB BC	[2,1,1]	[−20,10,20]

The point to focus on is that the common guess pipe mu st be updated after applying Hardy cross to each loop. The p rogram will be as follows (from here onwards, the main prog ram and the display portion will be shown as functions, and modules will remain the same):

Main Program

```
# Loop 1
R1=array([4,1,2])
Q1g=array([50,10,-40])

# Loop 2
R2=array([2,1,1])
Q2g=array([-20,10,20])

error=1
count=1

while error>1.E-8:
#---- LOOP 1 -----#
    Q1= Hardy_Cross(R1,Q1g)

    e1=abs(Q1-Q1g)

    # Updating Q1g guess
    Q1g=Q1.copy()

#---- LOOP 2 -----#
    Q2g[1]=Q1g[1]

    Q2= Hardy_Cross(R2,Q2g)

    e2=abs(Q2-Q2g)

    # Updating Q2g guess
    Q2g=Q2.copy()

    Q1g[1]=Q2g[1]

    error=sqrt(sum(e1**2)+sum(e2**2))

    count+=1
```

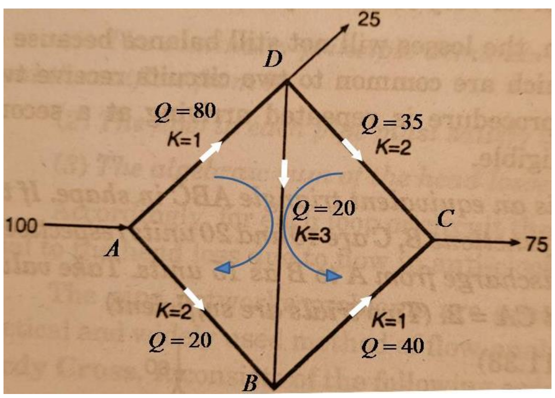
Data display

```
from pandas import *
Lp1=array(["AD","DB","BA"])
Lp2=array(["CD","DB","BC"])
data={'Lp1':Lp1,'Q1':Q1,'Lp2':Lp2,'Q2'
:Q2}
df=DataFrame(data)
df
```

The program output will be as follows:

	Lp1	Q1	Lp2	Q2
	AD	37.274316	CD	-16.590611
	DB	1.580257	DB	1.580257
	BA	-52.725684	BC	23.409389

Question 3: Solve below mentioned network :



In this problem, again, there are two loops. The initial guess discharge along each pipe is shown in the figure.

	Pipes	R	Q_g
Loop1	AD DB BA	[1,3,2]	$[80,20,-20]$
Loop2	CD DB BC	[2,3,1]	$[-35,20,40]$

The program will be as follows :

Main Program

```
# Loop 1
R1=array([4,1,2])
Q1g=array([50,10,-40])

# Loop 2
R2=array([2,1,1])
Q2g=array([-20,10,20])

error=1
count=1

while error>1.E-8:
#---- LOOP 1 -----#
    Q1= Hardy_Cross(R1,Q1g)

    e1=abs(Q1-Q1g)

    # Updating Q1g guess
    Q1g=Q1.copy()

#---- LOOP 2 -----#
    Q2g[1]=Q1g[1]

    Q2= Hardy_Cross(R2,Q2g)

    e2=abs(Q2-Q2g)

    # Updating Q2g guess
    Q2g=Q2.copy()

    Q1g[1]=Q2g[1]

    error=sqrt(sum(e1**2)+sum(e2**2))

    count+=1
```

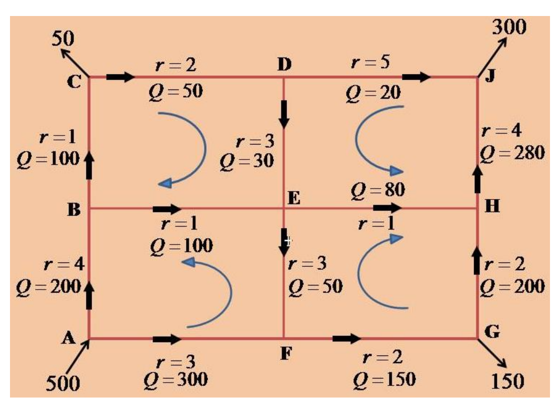
Data display

As nomenclatures are the same as in the previous problem so this portion will be the same as in Question 2

The program output will be as follows:

	Lp1	Q1	Lp2	Q2
	AD	58.523825	CD	-31.139097
	DB	2.273617	DB	2.273617
	BA	-41.476175	BC	43.860903

Question 4: Solve below mentioned network :



In this problem, there are four loops. The initial guess discharge along each pipe is shown in the figure. The loops are as follows:

	Pipes	R	Q_g
Loop 1	CD DE EB BC	[2,3,1,1]	[50,30,-100,100]
Loop2	JD DE EH HJ	[5,3,1,4]	[-20,30,80,280]
Loop3	GF FE EH HG	[2,3,1,2]	[-150,-50,80,-200]
Loop4	AF FE EB BA	[3,3,1,4]	[300,-50,-100,-200]

The program will be as follows :

Main Program

```
# Loop 1
R1=array([2,3,1,1])
Q1g=array([50,30,-100,100])

# Loop 2
R2=array([5,3,1,4])
Q2g=array([-20,30,80,280])

# Loop 3
R3=array([2,3,1,2])
Q3g=array([-150,-50,80,-200])

# Loop 4
R4=array([3,3,1,4])
Q4g=array([300,-50,-100,-200])

error=1
count=1

while error>1.E-8:

#-------- LOOP 1 ---------#
    Q1= Hardy_Cross(R1,Q1g)

    e1=abs(Q1-Q1g)

    # Updating Q1g guess
    Q1g=Q1.copy()


#-------- LOOP 2 ---------#
# common b/w Loop 1 and 2
    Q2g[1]=Q1g[1]

    Q2= Hardy_Cross(R2,Q2g)

    e2=abs(Q2-Q2g)

    # Updating Q2g guess
    Q2g=Q2.copy()

# common b/w Loop 1 and 2
    Q1g[1]=Q2g[1]

#-------- LOOP 3 ---------#
# common b/w Loop 2 and 3
    Q3g[2]=Q2g[2]

    Q3= Hardy_Cross(R3,Q3g)

    e3=abs(Q3-Q3g)

    # Updating Q2g guess
    Q3g=Q3.copy()
# common b/w Loop 2 and 3
    Q2g[2]=Q3g[2]

#-------- LOOP 4 ---------#
# common b/w Loop 3 and 4
    Q4g[1]=Q3g[1]

    Q4= Hardy_Cross(R4,Q4g)

    e4=abs(Q4-Q4g)

    # Updating Q2g guess
    Q4g=Q4.copy()

# common b/w Loop 1 and 2
    Q3g[1]=Q4g[1]

# common b/w Loop 4 and 1
    Q1g[2]=Q4g[2]

    error=sqrt(sum(e1**2)+sum(e2**2)+s
    um(e3**2)+sum(e4**2))

    count+=1
```

Data display

```
Lp1=array(["CD","DE","EB","BC"])
Lp2=array(["JD","DE","EH","HJ"])
Lp3=array(["GF","FE","EH","HG"])
Lp4=array(["AF","FE","EB","BA"])

data={'Lp1':Lp1,'Q1':Q1,'Lp2':Lp2,'Q2'
:Q2,'Lp3':Lp3,'Q3':Q3,'Lp4':Lp4,'Q4':Q
4}
df=DataFrame(data)
df
```

As in the problem, the last loop is also sharing the pipe with the first one, so the common pipe discharge has been updated before another while loop iteration will start. The program output will be as follows:

Lp1	Q1	Lp2	Q2	Lp3	Q3	Lp4	Q4
CD	77.603857	JD	-139.922426	GF	-26.222991	AF	271.047829
DE	-62.446775	DE	-62.446775	FE	44.363169	FE	44.363169
EB	-128.952171	EH	84.207524	EH	84.207524	EB	-128.952171
BC	127.603857	HJ	160.077574	HG	-76.222991	BA	-228.952171

As the main code has grown quite a bit, what can be done is one can make two more functions, one for loop and one for data transfer between common pipes. This will reduce the size of the main program as follows:

New Functions

```
def Loops(R1,Q1g):

    Q1=Hardy_Cross(R1,Q1g)

    e1=abs(Q1-Q1g)

    # Updating Q1g guess
    Q1g=Q1.copy()

    return Q1g,e1

def common(Q1g,Q2g,comm_index):
    # Common pipe
    Q2g[comm_index]=Q1g[comm_index]
    return Q2g
```

Main Program

```
# Loop 1
R1=array([2,3,1,1])
Q1g=array([50,30,-100,100])

# Loop 2
R2=array([5,3,1,4])
Q2g=array([-20,30,80,280])

# Loop 3
R3=array([2,3,1,2])
Q3g=array([-150,-50,80,-200])

# Loop 4
R4=array([3,3,1,4])
Q4g=array([300,-50,-100,-200])

#common=array([1,])
error=1
count=1

while error>1.E-8:
    #---- LOOP 1 -----#
    Q1g,e1=Loops(R1,Q1g)
    # common 1 ---> 2
    Q2g=common(Q1g,Q2g,1)

    #---- LOOP 2 -----#
    Q2g,e2=Loops(R2,Q2g)
    # common 2 ---> 1
    Q1g=common(Q2g,Q1g,1)
    # common 2 ---> 3
    Q3g=common(Q2g,Q3g,2)

    #---- LOOP 3 -----#
    Q3g,e3=Loops(R3,Q3g)
    # common 3 ---> 2
    Q2g=common(Q3g,Q2g,2)
    # common 3 ---> 4
    Q4g=common(Q3g,Q4g,1)

    #---- LOOP 3 -----#
    Q4g,e4=Loops(R4,Q4g)
    # common 4 ---> 3
    Q3g=common(Q4g,Q3g,1)
    # common 4 ---> 1
    Q1g=common(Q4g,Q1g,2)

     error=sqrt(sum(e1**2)+sum(e2**2)+s
     um(e3**2)+sum(e4**2))

    count+=1
```

By doing the above moderation, the size of the main program has been reduced as a lot of the repetitive stuff has already been gone into the functions (Loop and common). However, for smaller networks with one or two loops, one can follow Questions 1 to 3.

5. Conclusion

In this manuscript, a pipe flow network has been modelled with the help of the Hardy Cross method in python. First, a detailed explanation of the Hardy Cross algorithm has been done then an algorithm has been modelled in python. Four problems have been taken to check the function and program, and in all the cases, the Python code has given results in accordance with the literature. Practicing engineers and researchers will be able to address pipe flow issues accurately with the help of the methods described in this article. Computer programmes that have been created are extremely reliable and can be modified to address any kind of pipe flow network.

Nomenclature

h	Pressure head
Q	Discharge, i.e. volume flow rate
R	Flow resistance

Funding Statement

The authors have not received any funding for this research

References

[1] Bruno Eckhardt et al., "Turbulence Transition in Pipe Flow," *Annual Review of Fluid Mechanics*, vol. 39, pp. 447–468, 2007. [CrossRef] [Google Scholar] [Publisher Link]

[2] Hong-yue Sun et al., "Experimental Studies of Groundwater Pipe Flow Network Characteristics in Gravelly Soil Slopes," *Landslides*, vol. 9, no. 4, pp. 475-483, 2012. [CrossRef] [Google Scholar] [Publisher Link]

[3] Bruce Larock, Roland W. Jeppson, and Gary Watters, *Pipe Network Analysis*, pp. 2–5, 1999. [CrossRef]

[4] Hilary Ockendon, "Viscous Fluid Flow," *The European Journal of Mechanics - B/Fluids*, vol. 20, no. 1, pp. 157-158, 2001. [CrossRef] [Google Scholar] [Publisher Link]

[5] G. Biswas, and S. K. Som, *Introduction to Fluid Mechanics and Fluid Machines*, Tata Mcgraw-Hill Education, 2003. [Google Scholar]

[6] Satyabrata Podder, Paulam Deep Paul, and Arunabha Chanda, "The Effect of the Magnetic Field of High Intensities on Velocity Profiles of Slip Driven Non-Newtonian Fluid Flow Through the Circular, Straight Microchannel," *International Journal of Engineering Trends and Technology*, vol. 70, no. 4, pp. 383-388, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[7] Pankaj Dumka et al., "Finite Volume Modelling of an Axisymmetric Cylindrical Fin Using Python," *Research and Applications of Thermal Engineering*, vol. 4, no. 3, pp. 1–11, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[8] Yoong Cheah Huei, "Benefits and Introduction to Python Programming for Freshmore Students Using Inexpensive Robots," *Proceedings of IEEE International Conference on Teaching, Assessment and Learning for Engineering: Learning for the Future Now, TALE*, pp. 12–17, 2014. [CrossRef] [Google Scholar] [Publisher Link]

[9] Giovanni Moruzzi, "*Python Basics and the Interactive Mode,*" *Essential Python for the Physicist*, Cham: Springer International Publishing, pp. 1–39, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[10] Pankaj Dumka et al., "Kinematics of Fluid : A Python Approach," *International Journal of Research and Analytical Reviews*, vol. 9, no. 2, pp. 131–135, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[11] Pankaj Dumka et al., "Implementation of Buckingham ' S Pi Theorem Using Python," *Advances in Engineering Software*, vol. 173, p. 103232, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[12] Parth Singh Pawar, Dhananjay R. Mishra, and Pankaj Dumka, "Obtaining Exact Solutions of Visco- Incompress ible Parallel Flows Using Python," *International Journal of Engineering Applied Sciences and Technology*, vol. 6, no. 11, pp. 213–217, 2022. [Google Scholar] [Publisher Link]

[13] Dhananjay R. Mishra, and Pankaj Dumka, "Understanding the TDMA / Thomas Algorithm and Its Implementation in Python," *International Journal of All Research Education & Scientific Methods*, vol. 10, no. 10, pp. 998–1002, 2022. [Google Scholar] [Publisher Link]

[14] Krishna Gajula et al., "First Law of Thermodynamics for Closed System : A Python Approach," *Research and Applications of Thermal Engineering*, vol. 5, no. 3, pp. 1–10, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[15] Moisés Cywiak, and David Cywiak, "Sympy," *Multi-Platform Graphics Programming with Kivy: Basic Analytical Programming for 2D, 3D, and Stereoscopic Design*, Berkeley, CA: Apress, pp. 173–190, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[16] Aaron Meure et al., "Sympy: Symbolic Computing in Python," *Peerj Computer Science*, vol. 2017, no. 1, pp. 1–27, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[17] Robert Johansson, *Numerical Python: Scientific Computing and Data Science Applications with Numpy, Scipy and Matplotlib, Second Edition*, Apress, Berkeley, CA, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[18] Pankaj Dumka et al., "Application of He's Homotopy and Perturbation Method to Solve Heat Transfer Equations: A Python Approach," *Advances in Engineering Software*, vol. 170, p. 103160, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[19] Pankaj Dumka et al., "Load and Load Duration Curves Using Python," *International Journal of All Research Education and Scientific Methods (IJARESM)*, vol. 10, no. 8, pp. 2127–2134, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[20] Yashasvini Raghuvanshi et al., "Modelling Logic Gates in Python," *International Journal for Multidisciplinary Research*, vol. 4, no. 5, pp. 1–8, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[21] Ekaba Bisong, "Matplotlib and Seaborn," *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, Berkeley, CA: Apress, pp. 151–165, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[22] Pankaj Dumka et al., "Modelling Air Standard Thermodynamic Cycles Using Python," *Advances in Engineering Software*, vol. 172, p. 103186, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[23] Pankaj Dumka et al., "Modelling Pipe Flow Using Python," *International Education & Research Journal*, vol. 8, no. 10, pp. 8–11, 2022. [Publisher Link]

[24] G. F. Round, "Analysis of Flow in Pipe Networks ," *Canadian Journal of Civil Engineering*, vol. 10, no. 1, 1983. [CrossRef] [Publisher Link]

[25] A. M. G. Lopes, "Implementation of the Hardy-Cross Method for the Solution of Piping Networks," *Computer Applications in Engineering Education*, vol. 12, no. 2, pp. 117–125, 2004. [CrossRef] [Google Scholar] [Publisher Link]

[26] Dejan Brki´c, and Pavel Praks, "Short Overview of Early Developments of the Hardy Cross Type Methods for Computation of Flow Distribution in Pipe Networks," *Applied Science*, vol. 9, no. 10, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[27] E. John Finnemore, and Joseph B. Franzini, *Fluid Mechanics With Engineering Applications*, Mcgraw-Hill Education, 1977. [Google Scholar] [Publisher Link]