Indian Institute of Technology, Guwahati

Department Of Mechanical Engineering

SOFT COMPUTING
(ME674)

Coding Assignment 1

Submitted To:
Prof. Sukhomay Pal

Submitted By:
Mritunjay
Roll No. 214103004
M.Tech -AP

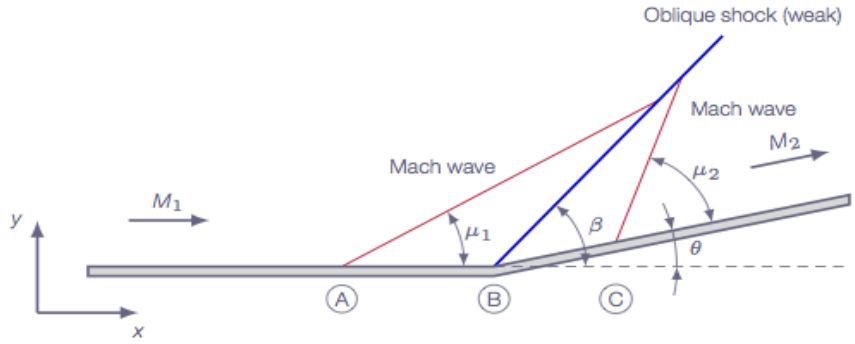
Table of Content

1. Introduction

In gas dynamics during a flow through the nozzle, it is very important to know the nature of flow around the corners especially when we are working with compressible supersonic flow.

However there are so many types of shock formation have been observed near the sharp corner or smooth corner like normal shock, oblique shock, Mach shock, oblique shock but for this report we are being little conservative to oblique shock waves.

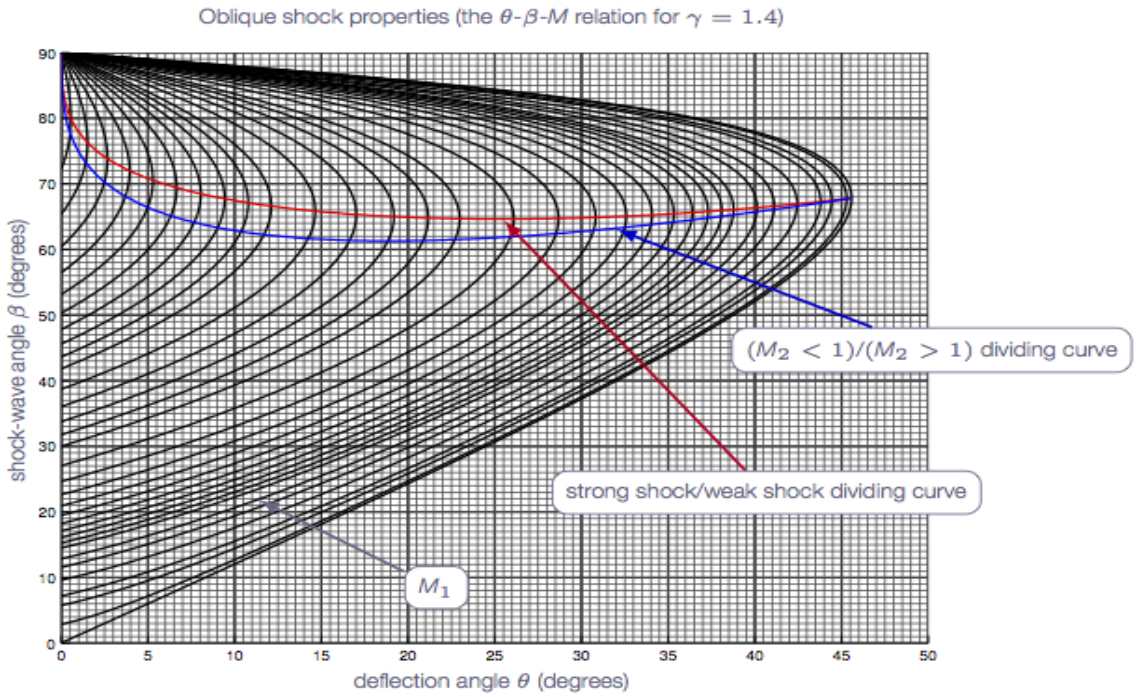
Oblique shock waves are those shock waves that forms near the sharp compression corner during a compressible fluid flow.



Analytical relation for calculating $\theta - \beta - M$ is:

$$\tan\theta = 2\cot\beta \frac{M^2 \sin^2\beta - 1}{M^2(\gamma + \cos 2\beta) + 2}$$

Using the above relation one can find the different value of $\theta - \beta - M$ according to our need on either side of the oblique shock wave. When we plot the above relation we find the below plot. Which is very helpful tool for calculation.



2. Data

Data for the training and testing pattern for the Artificial Neural Network (ANN) is done using the below sample data. Complete data file is being attached within the assignment folder as an excel file.

Below data is for the training of the neural network:

Theta (θ)	Initial Mach no.(M1)	$Beta(\beta)$	Initial Mach no.(M2)
5	47.89	1.5	1.33
5	34.3	2	1.82
5	27.42	2.5	2.29
5	23.13	3	2.75
5	20.18	3.5	3.2
5	18.02	4	3.64
5	16.37	4.5	4.07
10	56.68	1.5	1.11
10	39.31	2	1.64
10	31.85	2.5	2.09
10	27.38	3	2.51
10	24.38	3.5	2.9
10	22.23	4	3.29
10	20.62	4.5	3.65
10	19.38	5	4

From the above data set we are taking input as theta and upstream Mach number as the two inputs to the neural network and taking beta and downstream Mach number as the output of the neural network.

For the Testing of the neural network we used different set of pattern/data, The Testing data is shown in below table:

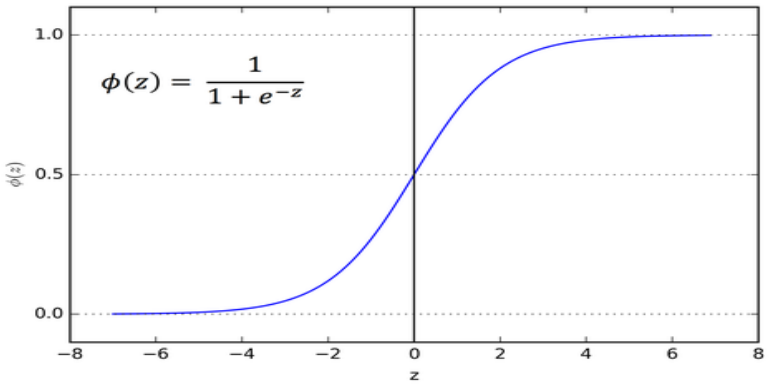
Theta (θ)	Initial Mach no.(M1)	$Beta(\beta)$	Initial Mach no.(M2)
40	18.5	52.55	1.76
40	19	52.53	1.76
40	19.5	52.5	1.76
40	20	52.48	1.77
40	20.5	52.46	1.77
45	14.5	66.67	1.04
45	15	66.22	1.06
45	15.5	65.91	1.07
45	16	65.67	1.08
45	16.5	65.48	1.09
45	17	65.31	1.1
45	17.5	65.17	1.11
45	18	65.05	1.11
45	18.5	64.94	1.12
45	19	64.84	1.12

3. Methodology

For the above parameters, a fully connected feed forward back propagating 3 layer ANN model was chosen to predict the two outputs based on the eight inputs. The key features of the model are:

- 308 patterns were taken as the training data while the remaining 260 patterns were used to verify the accuracy of the trained model.

- Transfer functions in both the hidden layer and output layer were taken as log-sigmoid with constant a1, a2 as 1.



- Transfer functions were taken as log-sigmoid as both inputs and outputs are greater than 0 and after normalization can be easily accommodated by the output range provided by log-sigmoid.

- The input data was normalized between the limits 0.1 to 0.9 in order to accommodate the output within range.

- The initial weight matrices were taken as zero for both input-hidden (Vij) and hidden-output (Wjk) for constant starting point to facilitate ease in comparison during the trial and error analysis method.

- The end condition of training was set to be mean square error less than 0.001 or maximum number of iterations less than 1,000 as the computation time for the model become very large.

- The final number of neurons in the hidden layer was taken as 2.

- Final values of learning rate and momentum coefficient were taken as 0.8 and 0.7 respectively from experimental trial and error analysis method.

4. Results & Observations

After using certain combination of learning rate (eta) and momentum co-efficient (∝) we come to an optimal condition of learning rate (eta) and momentum co-efficient (∝).

For hidden layer neuron we use trial and error technique and choose no. of hidden neurons as 8.

Sr. No.	Parameters	Optimal values
1.	Learning rate(eta)	0.8
2.	Momentum Co-efficient(∝)	0.7
3.	No. of hidden Neurons	8
4.	No. of Training patterns	248
5.	No. of Testing patterns	60

As the computation efficiency of the system is not sufficient for very high no. of iterations Therefore instead of choosing Mean square Error (MSE) as the convergence criteria we used only 1000-No. of iterations for the computation purpose and observed that after every iteration MSE is decreasing.

Sr. No.	Parameters (For 1000 Iterations only)	Results
1.	MSE of Training patterns	0.05332558270098947
2.	MSE of Testing patterns	0.07001509388203335

5. Conclusions

From the results MSE achieved after training the model the error are within limit for the given amount of iterations and the model has the ability to give better results with a powerful computing system.

6. Coding Section

PYTHON CODE for training of ANN-model

```
import numpy as np

# L=int(input("Enter no.of Input Neurons"))
# M=int(input("Enter no.of Hidden  Neurons"))
# N=int(input("Enter no.of Output Neurons"))
# P=int(input("Enter no.of Training Patterns"))
L=int(2)
M=int(8)
N=int(2)
P=int(247)
eta=0.8
alpha=0.7
MSE=1
iter=0
W_old=np.zeros((M,N))
V_old=np.zeros((L,M))

# Loading of input and output matrices
ip1,ip2=np.loadtxt('input.txt',skiprows=1,unpack=True)
ipmatrix=np.stack((ip1,ip2))

op1,op2=np.loadtxt('output.txt',skiprows=1,unpack=True)
opmatrix=np.stack((op1,op2))


# Random initialization of Weight matrices2

V=np.random.rand((L*M))
V=np.array(V).reshape(L,M)

W=np.random.rand((M*N))
W=np.array(W).reshape(M,N)


# Normalizing
for i in range(L):
    min_ip=min(ipmatrix[i])
    max_ip=max(ipmatrix[i])
    ipmatrix[i]=(ipmatrix[i]-min_ip)/(max_ip-min_ip)
    ipmatrix[i]=ipmatrix[i]*0.8+0.1
ipmatrix=np.transpose(ipmatrix)

for i in range(N):
    min_op=min(opmatrix[i])
    max_op=max(opmatrix[i])
    opmatrix[i]=(opmatrix[i]-min_op)/(max_op-min_op)
```

```
    opmatrix[i] = opmatrix[i] * 0.8 + 0.1
opmatrix=np.transpose(opmatrix)

# Main loop for calculating the connection weights
while(MSE>0.001 and iter<1000):
    iter=iter+1

    # Input to hidden layer
    ih=np.dot((ipmatrix),V)

    # Transfer function
    def transfunc(ip):
        return 1 / (1 + np.exp(-ip))

    # output from hidden layer
    oh = transfunc(ih)

    # Input to Output layer
    io=np.transpose(np.dot(oh,W))

    # Output from output layer
    op=np.transpose(transfunc(io))

    # Error and MSE calculations
    err= 0.5*((opmatrix-op)**2)
    temp=np.array(err).reshape(1,(2*P))
    MSE=(np.sum(temp))/P
    print(MSE)

    # calculation for Weight updatation
    delW = np.zeros((M,N))

    for p in range(P):
        for j in range(M):
            for k in range(N):
                delW[j][k] = delW[j][k] + eta * (opmatrix[p][k] - op[p][k]) * (1 - op[p][k] ** 2) *
oh[p][j]

    for j in range(M):
        for k in range(N):
            delW[j][k] = delW[j][k] / 2


    delV = np.zeros((L,M))
    for p in range(P):
        for i in range(L):
            for j in range(M):
                for k in range(N):
                    delV[i][j] = delV[i][j] + eta * (opmatrix[p][k] - op[p][k]) * (1 - op[p][k] ** 2) *
W[j][k] * oh[p][
                        j] * (1 - oh[p][j]) * ipmatrix[p][i]
```

```
    for i in range(L):
        for j in range(M):
            delV[i][j] = delV[i][j] / 4



    for j in range(M):
        for k in range(N):
            W[j][k] = W[j][k] + delW[j][k] + alpha * W_old[j][k]
    W_old = delW
    for i in range(L):
        for j in range(M):
            V[i][j] = V[i][j] + delV[i][j] + alpha * V_old[i][j]
    V_old = delV

# printing MSE after 1000 no. of iterations for training patterns
print("MSE of Training pattern::",MSE)


# Testing of patterns

# Data for testing pattern
Lt=int(2)
Mt=int(8)
Nt=int(2)
Pt=60
MSEt=1
itert=0
# Loading of input and output matrices
ip1,ip2=np.loadtxt('inputtest.txt',skiprows=1,unpack=True)
ipmatrix=np.stack((ip1,ip2))

op1,op2=np.loadtxt('outputtest.txt',skiprows=1,unpack=True)
opmatrix=np.stack((op1,op2))

# Normalizing
for i in range(Lt):
    min_ip=min(ipmatrix[i])
    max_ip=max(ipmatrix[i])
    ipmatrix[i]=(ipmatrix[i]-min_ip)/(max_ip-min_ip)
    ipmatrix[i]=ipmatrix[i]*0.8+0.1
ipmatrix=np.transpose(ipmatrix)

for i in range(Nt):
    min_op=min(opmatrix[i])
    max_op=max(opmatrix[i])
    opmatrix[i]=(opmatrix[i]-min_op)/(max_op-min_op)
    opmatrix[i] = opmatrix[i] * 0.8 + 0.1
opmatrix=np.transpose(opmatrix)

# Input to hidden layer
ih = np.dot((ipmatrix), V)
```

```
# Transfer function
def transfunc(ip):
    return 1 / (1 + np.exp(-ip))

# Output from hidden layer
oh = transfunc(ih)

# Input to output layer
io = np.transpose(np.dot(oh, W))

# Output frm output layer
op = np.transpose(transfunc(io))

# Error and MSE calculation
err = 0.5 * ((opmatrix - op) ** 2)
temp = np.array(err).reshape(1, (2 * Pt))
MSEt = (np.sum(temp)) / Pt

# printing of MSE for testing pattern
print("MSE of Testing pattern:", MSEt)
```

References

1. https://fdocuments.in/document/modern-compressible-flow-by-jdanderson.html?page=1
2. https://www.google.com/search?q=theta+beta+M+chart&oq=theta+beta+M+chart&aqs=chrome.0.69i59l2j69i60.11568j0j7&sourceid=chrome&ie=UTF-8
3. https://www.google.com/search?q=theta+beta+m+relation&tbm=isch&ved=2ahUKEwjU1Kah2tf2AhU8yaACHcu5Dl0Q2-cCegQIABAA&oq=the&gs_lcp=CgNpbWcQARgAMgcIIxDvAxAnMgcIIxDvAxAnMgQIABBDMgQIABBDMgQIABBDMgQIABBDMgQIABBDMgQIABBDMgQIABBDOgUIABCABDoGCAAQBxAeOgYIABAFEB46BggAEAgQHjoICAAQgAQQsQM6CAgAELEDEIMBOgsIABCABBCxAxCDAToHCAAQsQMQQzoECAAQHjoGCAAQChAYUK0NWK0dYMw2aANwAHgAgAG6AYgB4AmSAQMwLjeYAQCgAQGqAQtnd3Mtd2l6LWltZ8ABAQ&sclient=img&ei=qbI4YpTcGbySg8UPy_O66AU&bih=657&biw=1366#imgrc=0tUk2cG0MmEE2M
4. https://www.google.com/search?q=log+sigmoid+function&sxsrf=APq-WBvbhQYimsB5irz14NmKrC-Dr93REA:1647883110389&source=lnms&tbm=isch&sa=X&ved=2ahUKEwj29rP72tf2AhUISmwGHdBeBWQQ_AUoAXoECAIQAw&biw=1366&bih=657&dpr=1#imgrc=WrLX1T8sz3l8MM