University of Guanajuato
Finite Volume Method (Project II)

García Sánchez Marco Antonio[1]
[1]División de Ingenierías, Campus Irapuato-Salamanca, Universidad de Guanajuato, Salamanca, Gto., Mexico

Instructor: Damián Ascencio César Eduardo, Ph.D.

August 2019

Contents

Abstract

This work presents the finite volume method using two schemes for discretizing the governing equation in the advection-diffusion phenomenon.

1 Introduction

Many physical problems can be modeled by analyzing the balance of two phenomena: advection and diffusion, which occur simultaneously.
The first is related to the transport of species due to the presence of velocity fields, and the second is defined as the dispersion of the species involved along the physical domain of the problem.
The advection-diffusion phenomenon is common in nature and in both industrial and engineering applications; it is generally also referred to as a transport problem.
The finite volume method consists of dividing the domain into small finite-volume cells and applying the laws of energy conservation, relating the central node field with the neighboring nodes. Moreover, this method must include the effect of advective terms, which are present in any convective phenomenon in nature.
In this project, the effects of advective terms are included, and two different schemes are proposed to solve this problem. We will use the forward discretization scheme and the central difference scheme to account for these effects.

2 Governing Equations

A schematic together with the problem data to be addressed is shown in Figure 0.

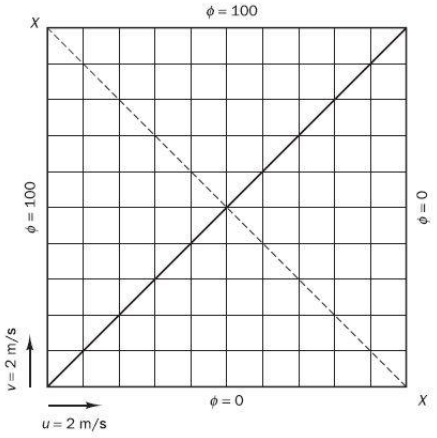


Figure 0. Problem schematic.

The advection-diffusion governing equation from [1] is given by:

$$\frac{\partial}{\partial x}(\rho u\varphi) + \frac{\partial}{\partial y}(\rho v\varphi) = \Gamma\frac{\partial^2}{\partial x^2}\varphi + \Gamma\frac{\partial^2}{\partial y^2}\varphi \tag{1}$$

3 Discretization

Discretizing the governing equation yields:

$$\begin{aligned} &A_n(\rho V_n)\varphi_n + A_e(\rho U_e)\varphi_e - A_w(\rho U_w)\varphi_w - A_s(\rho V_s)\varphi_s = \\ &= A_e\Gamma_e\frac{d\varphi}{dx}|_e - A_w\Gamma_w\frac{d\varphi}{dx}|_w + A_n\Gamma_n\frac{d\varphi}{dy}|_n - A_s\Gamma_s\frac{d\varphi}{dy}|_s \end{aligned} \tag{2}$$

Assuming:
-Homogeneous mesh $dx = dy$; $A_n = A_w = A_s = A_e$
-Also $F_i = (\varphi U)_i$ and $D_i = \frac{\Gamma_i}{dx}$
From this, a discretized governing equation can be obtained:

$$\begin{aligned} F_n\varphi_n + F_e\varphi_e - F_w\varphi_w - F_s\varphi_s = D_e(\varphi_e - \varphi_p) + \\ + D_n(\varphi_n - \varphi_p) - D_w(\varphi_p - \varphi_w) - D_s(\varphi_p - \varphi_s) \end{aligned} \tag{3}$$

3.1 Case 1: UDS

The forward difference scheme is presented as:

$$u > 0;\ \varphi_e = \varphi_p \text{ and } \varphi_w = \varphi_W$$

$$v > 0;\ \varphi_n = \varphi_p \text{ and } \varphi_s = \varphi_S$$

Substituting the above into equation 3 and factoring, we find the equation for central nodes:

$$\begin{aligned} \varphi_p(F_e + F_n + D_e + D_w) + \varphi_w(-F_w - D_w) + \\ + \varphi_s(-F_s - D_s) + \varphi_e(-D_e) + \varphi(-D_n) = 0 \end{aligned} \tag{4}$$

3.1.1 Discretization for corners

Node 1; using ghost nodes at W and S:

$$\begin{aligned} \varphi_p(F_e + F_n + F_s + D_e + 2D_w + 2D_s + D_n) + \varphi_e(-D_e) \\ + \varphi_n(-D_n) = 2C_w(F_w + D_w) + 2C_s(F_s + D_s) \end{aligned} \tag{5}$$

Node n; using ghost nodes at E and S:

$$\begin{aligned} \varphi_p(F_e + F_n + F_s + 2D_e + D_w + 2D_s + D_n) + \varphi_w(-F_w \\ - D_w) + \varphi_n(-D_n) = 2C_e(D_e) + 2C_s(F_s + D_s) \end{aligned} \tag{6}$$

Node γ ; using ghost nodes at W and N:

$$\begin{aligned} \varphi_p(F_e + F_n + F_w + D_e + 2D_w + D_s + 2D_n) + \varphi_e(-D_e) \\ + \varphi_s(-F_s - D_s) = 2C_w(F_w + D_w) + 2C_n(D_n) \end{aligned} \tag{7}$$

Node n^2 ; using ghost nodes at E and N:

$$\begin{aligned} \varphi_p(F_e + F_n + F_w + 2D_e + D_w + D_s + 2D_w) + \varphi_w(-D_w \\ -F_w) + \varphi_s(-F_s - D_s) = 2C_n(D_n) + 2C_e(D_e) \end{aligned} \tag{8}$$

3.1.2 Discretization for faces

South face: using ghost node at S

$$\begin{aligned} \varphi_p(F_e + F_n + F_s + D_e + D_w + 2D_s + D_n) + \varphi_w(-D_w \\ - F_w) + \varphi_n(-D_n) + \varphi_e(-D_e) = 2C_s(F_s + D_s) \end{aligned} \tag{9}$$

North face; using ghost node at N

$$\begin{aligned}&\varphi_p(F_e + F_n + F_s + D_e + D_w + D_s + 2D_n) + \varphi_w(-D_w \\ &- F_w) + \varphi_e(-D_e) + \varphi_s(-F_s - D_s) = 2C_n(D_n)\end{aligned} \tag{10}$$

West face; using ghost node at W

$$\begin{aligned}&\varphi_p(F_e + F_n + F_w + D_e + 2D_w + D_s + D_n) + \varphi_e(-D_e) \\ &+ \varphi_s(-F_s - D_s) + \varphi_n(-D_n) = 2C_w(F_w + D_w)\end{aligned} \tag{11}$$

3.2 Case 2: CDS

The central difference scheme is presented as:

$$\varphi_e = \frac{\varphi_p + \varphi_E}{2} \text{ and } \varphi_w = \frac{\varphi_p + \varphi_W}{2}$$

$$\varphi_n = \frac{\varphi_p + \varphi_N}{2} \text{ and } \varphi_s = \frac{\varphi_p + \varphi_S}{2}$$

Substituting the above into equation 3 and factoring, we obtain the equation for central nodes:

$$\begin{aligned}&\varphi_p(F_e + F_n + D_e + D_w) + \varphi_w(-F_w - D_w) \\ &+ \varphi_s(-F\_D_s) + \varphi_e(-D_e) + \varphi(-D_n) = 0\end{aligned} \tag{12}$$

3.2.1 Discretization for corners

Node 1; using ghost node at W and S:

$$\begin{aligned}&\varphi_p(\frac{1}{2}(F_e + F_n) + D_e + D_w + 2D_s + 2D_n) + \varphi_e(\frac{1}{2}F_e - D_e) \\ &+ \varphi_n(\frac{1}{2}F_n - D_n) = 2C_w(\frac{1}{2}F_w + D_w) + 2C_s(\frac{1}{2}F_s + D_s)\end{aligned} \tag{13}$$

Node n; using ghost node at E and S:

$$\begin{aligned}&\varphi_p(\frac{1}{2}(F_e + F_n - F_s - F_w) + 2D_e + D_w + 2D_s + D_n) + \varphi_w(-\frac{1}{2}F_w \\ &- D_w) + \varphi_n(\frac{1}{2}F_n - D_n) = 2C_e(-\frac{1}{2}F_e + D_e) + 2C_s(\frac{1}{2}F_s + D_s)\end{aligned} \tag{14}$$

Node γ; using ghost node at W and N:

$$\begin{aligned}&\varphi_p(\frac{1}{2}(F_e - F_s) + D_e + 2D_w + D_s + 2D_n) + \varphi_e(\frac{1}{2}F_e - D_e) \\ &+ \varphi_s(-\frac{1}{2}F_s - D_s) = 2C_w(\frac{1}{2}F_w + D_w) + 2C_n(-\frac{1}{2} + D_n)\end{aligned} \tag{15}$$

Node n^2; using ghost node at E and N:

$$\begin{aligned}&\varphi_p(\frac{1}{2}(F_w - F_s) + 2D_e + D_w + D_s + 2D_n) + \varphi_w(-D_w - \frac{1}{2}F_w) \\ &+ \varphi_s(-\frac{1}{2}F_s - D_s) = 2C_n(-\frac{1}{2}F_n - D_n) + 2C_e(-\frac{1}{2}F_e + D_e)\end{aligned} \tag{16}$$

3.2.2 Discretization for faces

South face: using ghost node at S:

$$\begin{aligned}&\varphi_p(\frac{1}{2}(F_e+F_n-F_w)+D_e+D_w+2D_s+D_n)+\varphi_w(-D_w-\frac{1}{2}F_w)\\&+\varphi_n(\frac{1}{2}F_n-D_n)+\varphi_e(\frac{1}{2}F_e-D_e)=2C_s(\frac{1}{2}F_s+D_s)\end{aligned} \tag{17}$$

East face: using ghost node at E:

$$\begin{aligned}&\varphi_p(\frac{1}{2}(F_e+F_n-F_s)+D_e+D_w+D_s+2D_n)+\varphi_w(-D_w-\frac{1}{2}F_w)\\&+\varphi_e(\frac{1}{2}F_e-D_e)+\varphi_s(-\frac{1}{2}F_s-D_s)=2C_n(\frac{1}{2}F_w-D_n)\end{aligned} \tag{18}$$

North face: using ghost node at N:

$$\begin{aligned}&\varphi_p(\frac{1}{2}(F_n-F_s-F_w)+D_e+D_w+D_s+2D_n)+\varphi_e(\frac{1}{2}F_e-D_e)\\&+\varphi_s(-\frac{1}{2}F_s-D_s)+\varphi_W(\frac{1}{2}F_w-D_w)=2C_w(-\frac{1}{2}F_w+D_w)\end{aligned} \tag{19}$$

4 Results and Mesh Independence

To solve the equations generated by the finite volume method, it is possible to use numerical methods such as Gauss-Seidel. For the proposed problem, a Python code was developed, shown in the Appendices, which solves the equations obtained. This also allows control over parameters such as the number of nodes, boundary conditions, and other constraints, making it possible to observe the variations along the fin according to the stipulated conditions.
The properties and characteristics of the problem for computational solution are shown in Table 1.

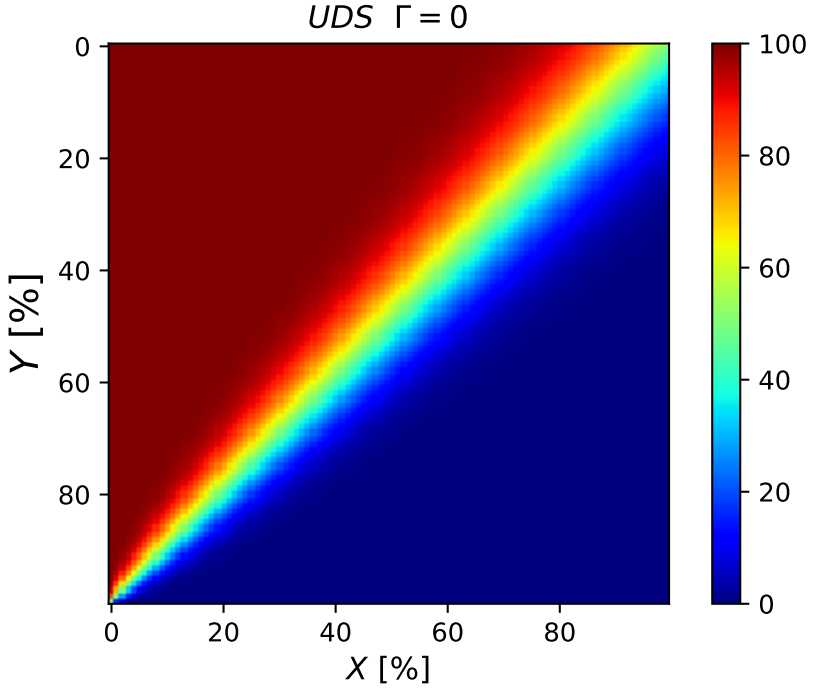
Table 1: Problem properties

Data	**Value**	Units
C_s	0	–
C_n	100	–
C_e	0	–
C_w	100	–
ρ	1	kgm^{-3}
Γ	1.0	–
L	1	m
u	2	ms^{-1}
v	2	ms^{-1}
$E_{proposed}$	0.001	–

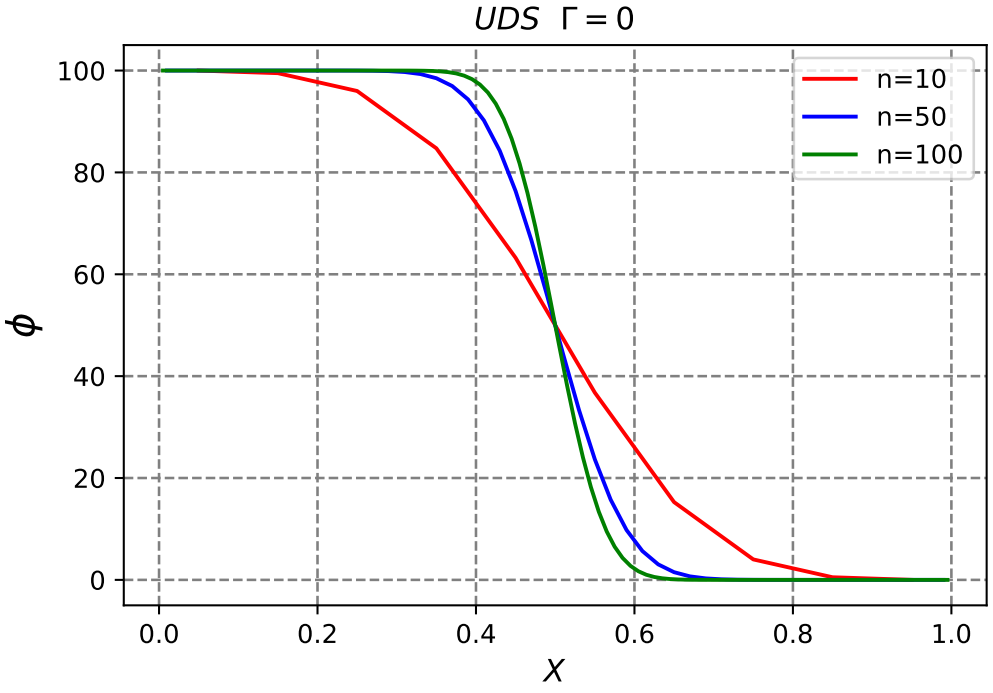
4.1 UDS Results

4.1.1 $\Gamma = 0$

By numerically solving the discretized equations, it was possible to construct a scheme where the distribution of the property φ can be observed.



(a) φ distribution



(b) Mesh independence (false diffusion)

Figure 1: UDS for $\Gamma = 0$

Figure 1 shows the distribution and a mesh independence analysis. In 1(a), the solution is as expected due to symmetry, while in 1(b) it is possible to observe that mesh independence is satisfactory from 50x50 elements,

with the solution trend stabilizing at 100x100 elements. For the solution of the problem, a mesh density of 50x50 elements was chosen.

4.1.2 $\Gamma = 1$

Using the same mesh density, it is possible to obtain the distribution of φ when $\Gamma = 1$. Figure 2 shows the independence and property distribution for this case. From 2(b), it can be seen that a 10x10 mesh yields correct results, as the non-constant diagonal property has been plotted, indicating a good approximation to the physical result. Similar behavior is expected due to symmetry.

4.2 CDS Results

4.2.1 $\Gamma = 0$

For this scheme, the discretized equations yield a singular system (one diagonal element is zero), making it impossible to solve. Therefore, there is no numerical solution using the central difference scheme when $\Gamma = 0$.

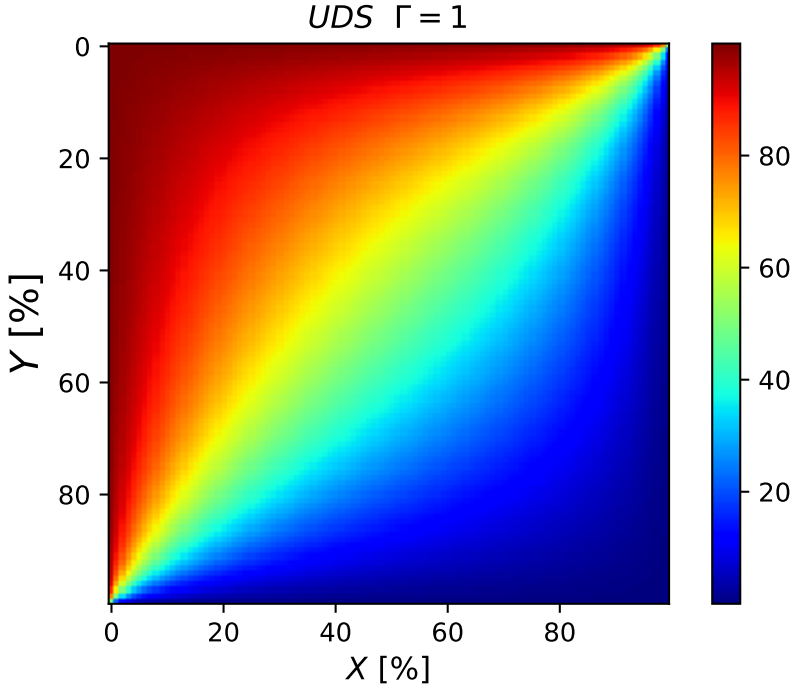
4.2.2 $\Gamma = 1$

Using the equations of this case, Figure 3 was obtained, showing mesh independence and the property distribution. As can be seen from Figures 2 and 3, a very similar behavior is observed for the meshes when $\Gamma = 1$, as expected, since both schemes should provide the same solution with an adequate mesh.

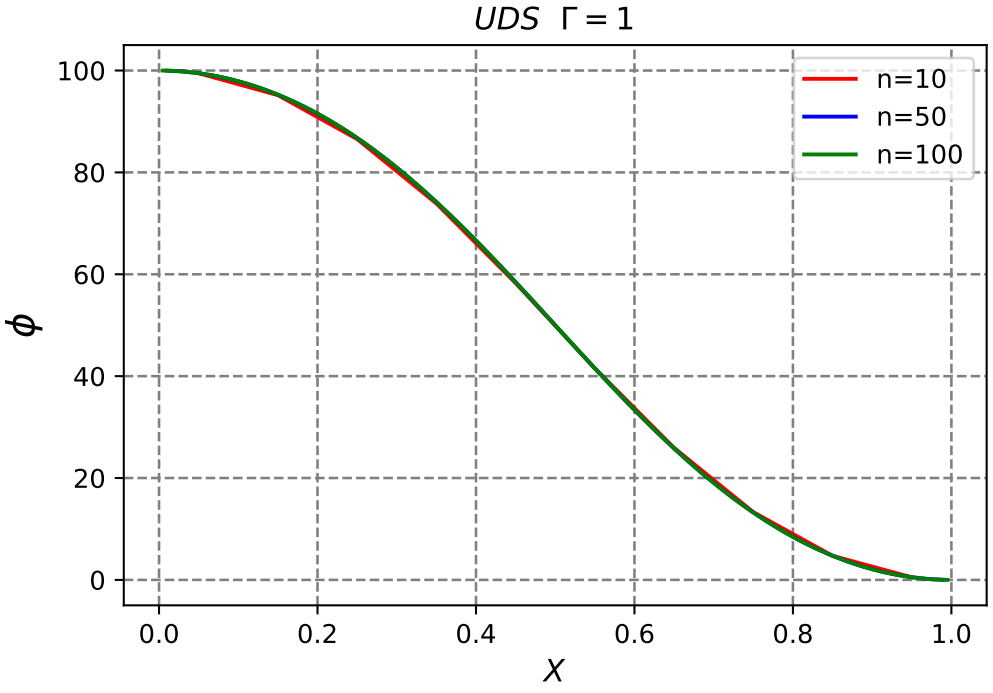
5 Conclusions

As observed in the results, the difference between methods lies in their accuracy. In central differences, it is crucial to know what happens in both directions of the node, while in forward differences, only the conditions of the node of interest and what is ahead matter.

The forward difference method has the disadvantage of showing false diffusion results, but this issue can be corrected by increasing the number of nodes, thereby obtaining a relatively low Peclet number, which stabilizes the system solution.

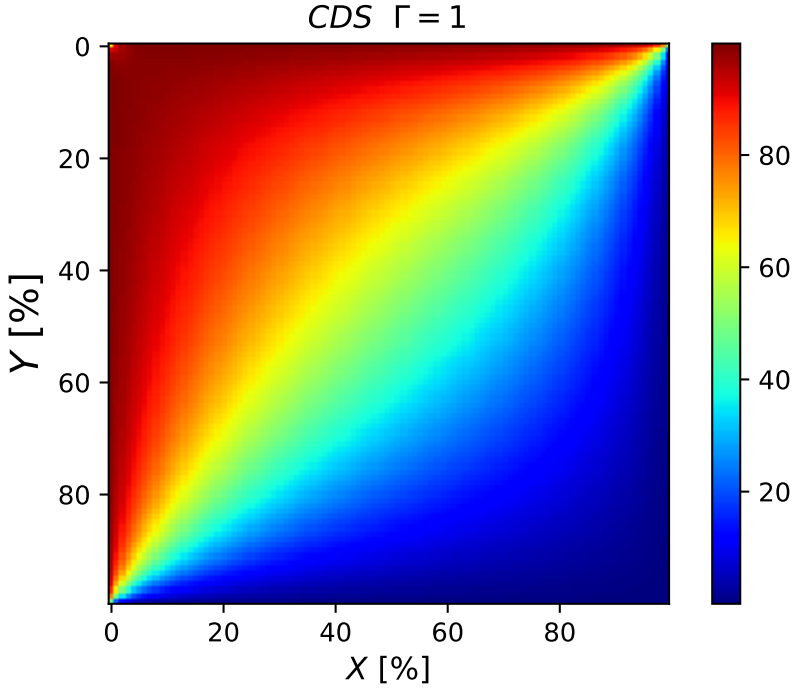


(a) φ distribution

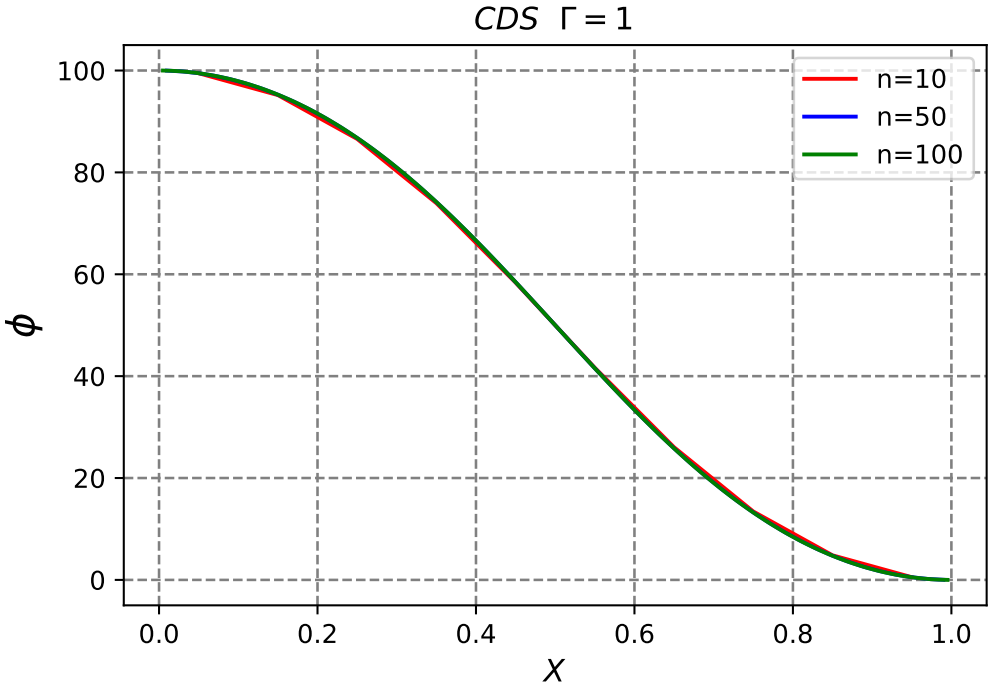


(b) Mesh independence

Figure 2: UDS for $\Gamma = 1$



(a) φ distribution



(b) Mesh independence

Figure 3: CDS for $\Gamma = 1$

References

[1] H. Versteeg and W. Malalasekera, "An introduction to computational fluid dynamics," *Finite Volume Method, Essex, Longman Scientific & Technical*, 1995.

Appendix I: Solution Codes

```
print('\033[H\033[J')
import numpy as np
import matplotlib.pyplot as plt
from Problema2 import Problema2
from scipy import interpolate


n=10
gamma=1
L=1
dx=dy=L/n

esquema='CDS'

if esquema=='UDS':
    alpha=1
    betha=0
if esquema=='CDS':
    alpha=0
    betha=1

Dcons=np.zeros(((n)),float)
Dphi=np.zeros(((n)),float)
x=np.zeros(((n)),float)


sol=Problema2(n,dx,np,gamma,alpha,betha)
for k in range(0,n):
    Dcons[k]=sol[k*(n)+k]
    Dphi[k]=sol[-k*n-n+k]
    x[k]=dx/2+dx*k

n=50
dx=dy=L/n

Dcons2=np.zeros(((n)),float)
Dphi2=np.zeros(((n)),float)
x2=np.zeros(((n)),float)

sol2=Problema2(n,dx,np,gamma,alpha,betha)
for k in range(0,n):
    Dcons2[k]=sol2[k*(n)+k]
    Dphi2[k]=sol2[-k*n-n+k]
    x2[k]=dx/2+dx*k

n=100
dx=dy=L/n

Dcons3=np.zeros(((n)),float)
Dphi3=np.zeros(((n)),float)
x3=np.zeros(((n)),float)

sol3=Problema2(n,dx,np,gamma,alpha,betha)
for k in range(0,n):
    Dcons3[k]=sol3[k*(n)+k]
    Dphi3[k]=sol3[-k*n-n+k]
    x3[k]=dx/2+dx*k
#
```

```
#
A=np.zeros(((n,n)),float)
arr = sol3.reshape((n,n))
for m in range (0,n):
    A[m]=arr[n-1-m]

plt.figure (1)
plt.plot(x,Dphi, label = "n=10", color = 'red')
plt.plot(x2,Dphi2, label = "n=50", color = 'blue')
plt.plot(x3,Dphi3, label = "n=100", color = 'green')
plt.legend(loc="upper right")
plt.hold(True)
plt.grid(True)
plt.grid(color = '0.5', linestyle = '--', linewidth = 1)
plt.xlabel(r"$X$", fontsize = 12, color = 'black')
plt.ylabel(r"$\phi$", fontsize = 15, color = 'black')
plt.title('$CDS\ \ \Gamma=1$',fontsize = 12, color = 'black', verticalalignment = 'baseline',
    horizontalalignment = 'center')
plt.savefig('CDS_Gamma=1.pdf')
plt.show()


xm=np.arange(dx/2,L,dx)
ym=np.arange(dx/2,L,dx)
xx,yy=np.meshgrid(xm,ym)
f=interpolate.interp2d(xm,ym,sol)
im = plt.imshow(A,cmap='jet')
plt.colorbar()
plt.hsv
plt.xlabel(r"$X\ [\%]$", fontsize = 12, color = 'black')
plt.ylabel(r"$Y\ [\%]$", fontsize = 15, color = 'black')
plt.title('$CDS\ \ \Gamma=1$',fontsize = 12, color = 'black', verticalalignment = 'baseline',
    horizontalalignment = 'center')
plt.savefig('CDS_Gamma=1_Dis.pdf')


def Problema2(n,dx,np,gamma,alpha,betha):
   #nodosX=nodosY
    nt=n**2 #nodos totales del sistema
    error=.001 #error propuesto
    A=np.zeros(((nt,nt)),float)
    X=np.zeros(((nt)),float)
    B=np.zeros(((nt)),float)
    Xc=np.zeros(((nt)),float)
    E=np.zeros(((nt)),float)
    Ln=np.zeros(((nt)),float)
    A2=np.zeros(((n,n)),float)      #matriz de visualizacion de malla

    rho=1
    u=v=2
    Fe=Fw=rho*u
    Fn=Fs=rho*v
    De=Dw=Ds=Dn=gamma/dx
    print(Dn)

    Cs=0
    Ce=0
    Cn=100
    Cw=100

```

```
26
27      for i in range(0,nt):
28          #nodos de las esquinas
29          if i==0:                    #nodo inferior izquierda
30              A[i,i]=alpha*(Fn+Fe+Fw+Fs+De+2*Dw+Dn+2*Ds)+betha*(.5*(Fn+Fe-Fs-Fw+Fs+Fw)+De+Dn+2*Ds
        +2*Dw)
31              A[i,n]=alpha*(-Dn)+betha*(.5*Fn-Dn)
32              A[i,i+1]=alpha*(-De)+betha*(.5*Fe-De)
33              B[i]=alpha*((Fw+Dw)*2*Cw+(Fs+Ds)*2*Cs)+betha*(2*Cs*(.5*Fs+Ds)+2*Cw*(.5*Fw+Dw))
34          if i==(n-1):                #nodo inferior derecha
35              A[i,i]=alpha*(Fn+Fe+Fs+2*De+Dw+2*Ds+Dn)+betha*(.5*(Fn+Fe-Fs-Fw+Fs-Fe)+2*De+Dn+Dw+2*
        Ds)
36              A[i,2*n-1]=alpha*(-Dn)+betha*(.5*Fn-Dn)
37              A[i,i-1]=alpha*(-Fw-Dw)+betha*(-.5*Fw-Dw)
38              B[i]=alpha*((De)*2*Ce+(Fs+Ds)*2*Cs)+betha*(2*Cs*(.5*Fs+Ds)+2*Ce*(-.5*Fe+De))
39
40          if i==((n-1)*n):            #nodo superior izquierda
41              A[i,i]=alpha*(Fe+Fn+Fw+De+2*Dw+2*Dn+Ds)+betha*(.5*(Fn+Fe-Fw-Fs-Fn+Fw)+De+2*Dn+Ds+2*
        Dw)
42              A[i,i+1]=alpha*(-De)+betha*(.5*Fe-De)
43              A[i,i-n]=alpha*(-Fs-Ds)+betha*(-.5*Fs-Ds)
44              B[i]=alpha*((Fw+Dw)*2*Cw+(Dn)*2*Cn)+betha*(2*Cs*(-.5*Fe+Dn)+2*Cw*(.5*Fw+Dw))
45
46          if i==(nt-1):               #nodo superior derecha
47              A[i,i]=alpha*(Fe+Fn+2*De+Dw+2*Dn+Ds)+betha*(.5*(Fn+Fe-Fs-Fw-Fn-Fe)+2*De+2*Dn+Ds+Dw)
48              A[i,i-1]=alpha*(-Dw-Fw)+betha*(-.5*Fw-Dw)
49              A[i,i-n]=alpha*(-Ds-Fs)+betha*(-.5*Fs-Ds)
50              B[i]=alpha*((Dn)*2*Cn+(De)*2*Ce)+betha*(2*Cn*(-.5*Fn+Dw)+2*Ce*(-.5*Fe+De))
51
52
53          #nodos de las fronteras
54          if i>0 and i<n-1:           #frontera sur
55              A[i,i]=alpha*(Fn+Fe+Fs+De+Dw+Dn+2*Ds)+betha*(.5*(Fn+Fe-Fs-Fw+Fs)+De+Dn+Dw+2*Ds)
56              A[i,i+1]=alpha*(-De)+betha*(.5*Fe-De)
57              A[i,i+(n)]=alpha*(-Dn)+betha*(.5*Fn-Dn)
58              A[i,i-1]=alpha*(-Fw-Dw)+betha*(-.5*Fw-Dw)
59              B[i]=alpha*((Fs+Ds)*2*Cs)+betha*(2*Cs*(.5*Fs+Ds))
60
61             #nodos centrales...
62              for j in range(1,(n-1)):
63                  A[j+n*i,j+n*i]=alpha*(Fn+Fe+De+Dw+Dn+Ds)+betha*(.5*(Fn+Fe-Fs-Fw)+De+Dn+Ds+Dw)
64                  A[j+n*i,j+n*i+1]=alpha*(-De)+betha*(.5*Fe-De)
65                  A[j+n*i,j+n*i-1]=alpha*(-Fw-Dw)+betha*(-.5*Fw-Dw)
66                  A[j+n*i,j+n*i-n]=alpha*(-Fs-Ds)+betha*(-.5*Fs-Ds)
67                  A[j+n*i,j+n*i+n]=alpha*(-Dn)+betha*(.5*Fn-Dn)
68
69          if i<n-1 and i>0:           #forntera oeste
70              A[i*n,i*n]=alpha*(Fn+Fe+Fw+De+2*Dw+Dn+Ds)+betha*(.5*(Fn+Fe-Fs-Fw+Fw)+De+Dn+Ds+2*Dw)
71              A[i*n,i*n+n]=alpha*(-Dn)+betha*(.5*Fn-Dw)
72              A[i*n,i*n+1]=alpha*(-De)+betha*(.5*Fe-De)
73              A[i*n,i*n-n]=alpha*(-Fs-Ds)+betha*(-.5*Fs-Ds)
74              B[i*n]=alpha*(2*Cw*(Fw+Dw))+betha*(2*Cw*(.5*Fw+Dw))
75
76          if i<n-1 and i>0:           #frontera este
77              A[i*n+n-1,i*n+n-1]=alpha*(Fn+Fe+2*De+Dw+Dn+Ds)+betha*(.5*(Fn+Fe-Fs-Fw-Fe)+Dn+Ds+Dw
        +2*De)
78              A[i*n+n-1,i*n+2*n-1]=alpha*(-Dn)+betha*(.5*Fn-Dn)
79              A[i*n+n-1,i*n-1]=alpha*(-Ds-Fs)+betha*(-.5*Fs-Ds)
80              A[i*n+n-1,i*n+n-2]=alpha*(-Fw-Dw)+betha*(-.5*Fw-Dw)
81              B[i*n+n-1]=alpha*(2*Ce*(De))+betha*(2*Ce*(-.5*Fe+De))
```

```
82
83          if i>(nt-n) and i<nt-1:    #frontera norte
84              A[i,i]=alpha*(Fe+Fn+De+Dw+2*Dn+Ds)+betha*(.5*(Fn+Fe-Fs-Fw-Fn)+Dn+Ds+Dw+2*Dn)
85              A[i,i-1]=alpha*(-Fw-Dw)+betha*(-.5*Fw-Dw)
86              A[i,i+1]=alpha*(-De)+betha*(.5*Fe-De)
87              A[i,i-n]=alpha*(-Ds-Fs)+betha*(-.5*Fs-Ds)
88              B[i]=alpha*(2*Cn*(Dn))+betha*(2*Cn*(-.5*Fn+Dn))
89
90  #X=gauss_seidel(A,B,X,Xc,E,n,error,np)
91      AA=np.linalg.inv(A)
92      sol=np.dot(AA,B)
93      print(A)
94      print(B)
95      return(sol)
```