SIMPLE Algorithm Project

Diego Alejandro Cepeda Cuartas - dacepedac1@gmail.com - CFD Mentorship program: Batch 3

AIM

The SIMPLE (Semi-Implicit Method for Pressure Linked Equations) algorithm was the first way to solve the Navier-Stokes Partial Differential Equations, to get the flow behavior from a determined fluid; This algorithm was developed in 1970 by professor Brian Spalding and his student, Suhas Patankar. Nowadays, the algorithm is very useful in computational programs that involve fluid simulations like ANSYS Fluent.

The SIMPLE algorithm is linked with an important grid assumption, named staggered grid, where the velocity and the pressure are not stored in the same center of the cell. The staggered grid is obtained because the pressure gradient in the N-S equation will never converge if the pressure is taken as a mean value. Thus, using the staggered grid concept, the following equation is obtained:

$$a_e u_e = \sum a_{nb} u_{nb} + (P_P - P_E) \cdot A_e + b_s \quad (1)$$

Where $\sum a_{nb} u_{nb}$ is the sum of all the neighbor velocities which are addressed over the same axis where the velocity direction is; A_e is the cross-sectional area of the analyzed node; P_P and P_E are the pressure values in the neighbor cell center nodes and; b_s is the source term, belonging to other physical phenomenon that affects the flow.

In this project, the SIMPLE Algorithm will be programmed in MATLAB, to get the solution of the 1D Navier-Stokes equation.

OBJECTIVES

The objective of this project is to get a code that can solve all the terms from the Navier-Stokes equation in one dimension, without considering any other source term of the equation. Furthermore, to make sure that the code is correctly working, exercise 6.2 from the “An Introduction to Computational Fluid Dynamics – Finite Volume Method” book will be solved by using the program.

DEVELOPMENT

The first step is to arrange the equation previously attached, in terms of the SIMPLE algorithm. In the SIMPLE code, the initial velocity and the pressure in each node are taken as guessed values; these values can be represented with the sign *.

$$P^* = Guessed\ Pressure$$

$$u^* = Guessed\ X\ Velocity$$

$$v^* = Guessed\ Y\ Velocity$$

$$w^* = Guessed\ Z\ Velocity$$

As it was described before, the N-S equation solution will be only solved in one dimension, then, the momentum equation for this case will be:

$$a_e u_e^* = \sum a_{nb} u_{nb}^* + (P_P^* - P_E^*) \cdot A_e + b_s^* \quad (2)$$

However, the equation (2) cannot update the pressure values. To get the update of the pressure value, the SIMPLE Algorithm uses a pressure correction (P'), based on the finite volume method of the continuity equation, but takes the solution from the velocity nodes.

$$(\rho u A)_e - (\rho u A)_w = 0 \quad (3)$$

Rearranging the equation (3), and knowing that the velocity equation correction is equal to:

$$u_e = u_e^* + (P'_P - P'_E) \cdot d_e \quad (4)$$

Where P'_P and P'_E are the pressure belonging to the neighbor nodes; $d_e = \frac{A_e}{a_e}$ and; u_e^* the guessed velocity value obtained from the equation (2). The equation (3) can be expressed as the equation (5).

$$a_P P'_P = a_E P'_E + a_W P'_W + b' \quad (5)$$

Where $a_P = a_E + a_W$; $a_E = \rho_e A_e d_e$; $a_W = \rho_w A_w d_w$ and; $b' = \rho_w A_w u_W^* - \rho_e A_e u_E^*$.

The velocity correction is only established as the second term from the equation (4).

$$u'_e = (P'_P - P'_E) \cdot d_e \quad (6)$$

Finally, the calculated velocity and pressure values are obtained from the sum between the guessed values and the correction values.

$$u_e^{calculated} = u_e^* + u'_e \quad (7)$$

$$P_P^{calculated} = P_P^* + P'_P \quad (8)$$

Another thing that the SIMPLE Algorithm takes into its equations is a iterative solver method that helps stabilize the equation in each new iteration; This method is known as the under-relaxation method. The under-relaxation method is used to get the pressure correction and the new terms get the convergence; this method adds a new coefficient terms into the equations (2) and (5), and generates new velocity and pressure values, described in the equations (10) and (11).

$$\frac{a_e}{\alpha_u} u_e^* = \sum a_{nb} u_{nb}^* + (P_P^* - P_E^*) \cdot A_e + b_s^* + \left[(1-\alpha_u)\frac{a_e}{\alpha_u}\right] u_e^{old} \quad (9)$$

$$u_e^{new} = (1-\alpha_u) u_e^{old} + \alpha_u u_e^{calculated} \quad (10)$$

$$P_P^{new} = (1-\alpha_u) P_P^{old} + \alpha_u P_P^{calculated} \quad (11)$$

The term d in the equation (5) will value $d = \frac{A_e \alpha_u}{a_e}$ in the next iterations. Furthermore, the α_u value must be greater than 0 but less than 1.

Once the SIMPLE algorithm equations are described and understood, the next step is to define the geometry. The geometry in this case is a geometry which will vary symmetrically along its y axis; that means that the body will be equal as at the upper section as at the lower section of the x axis. For example:

- Convergent nozzle

Likewise, the cross-sectional area of the body will be always a circular area. Knowing this, the area equation of each section will be equal to the equation (12) for the velocity grid and, equation (13) for the pressure grid.

$$A_v = \frac{\pi}{4}[2(L-(n-1)dx)\tan(\theta) + d_o]^2 \quad (12)$$

$$A_v = \frac{\pi}{4}\left[2\left(L-(n-1)dx+\frac{dx}{2}\right)\tan(\theta) + d_o\right]^2 \quad (13)$$

Where d_o is the outlet diameter; n is the node number; θ is the angle found between the nozzle wall and the axis and; dx the length of the cell.

The final step is to perform the codes that were attached in the appendix section of this project, and corroborate them with the exercise that is shown in the book referenced (H. K., & Malalasekera, W.). To start the code, it has to be clear that the cell split is performed as it is shown in the image 1.

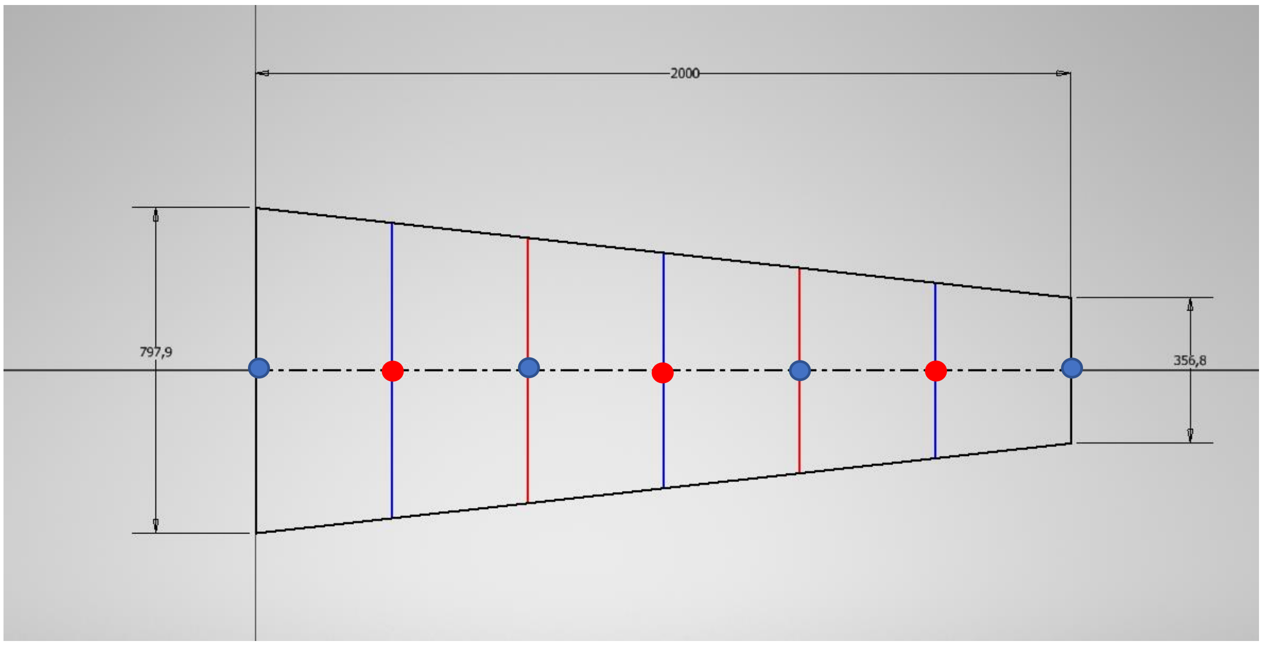
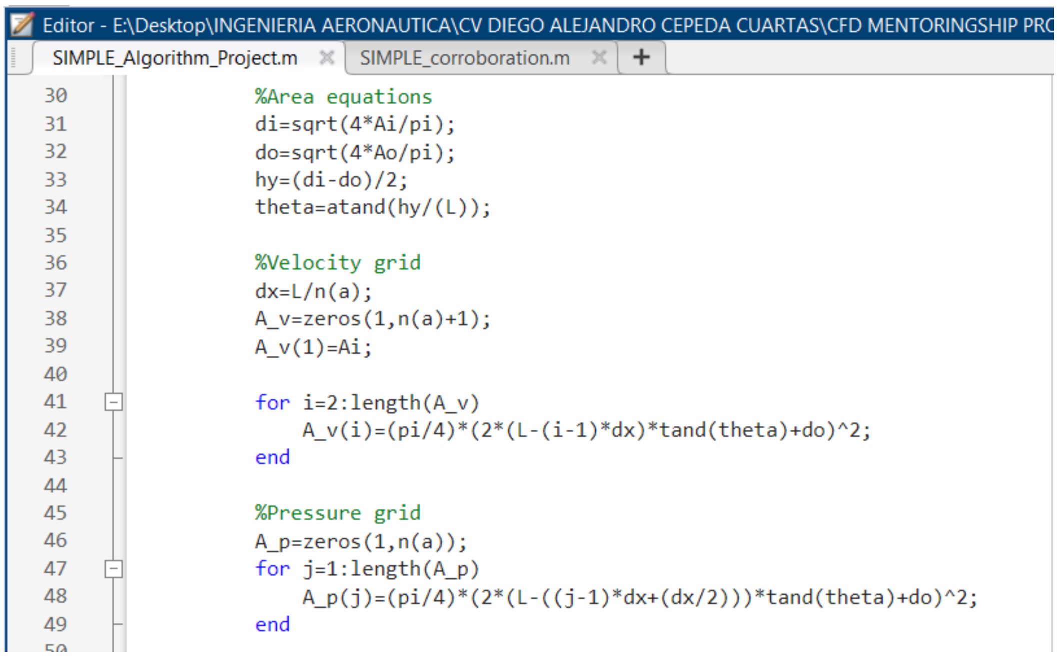


Figure 1. Geometry shape and, velocity and pressure grid split.

The red dots and lines represent the velocity grid and, the blue ones represent the pressure grid. Thus, the velocity and pressure area matrix must have different lengths, storing n values in the pressure grid and n+1 value in the other. The inlet and outlet areas belong to the velocity grid. The nomenclature of the number of cells is leaded by the velocity grid, then the number of cells in the image is equal to 3.

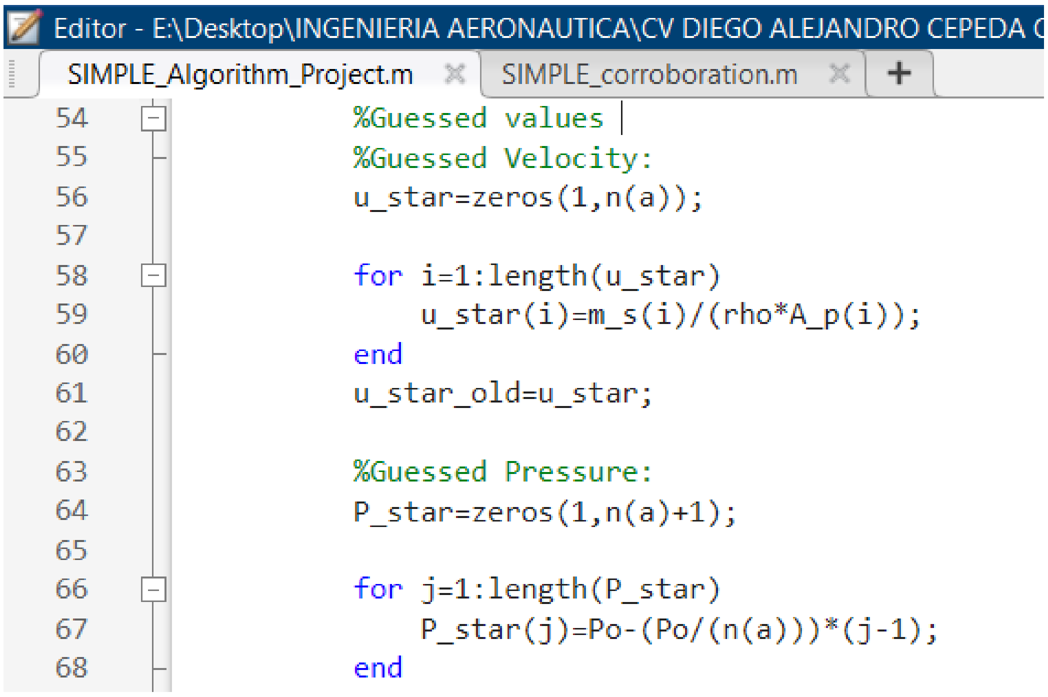
Thus, the areas are stored in the code as it is shown either in the following image or in the appendix.


Editor - E:\Desktop\INGENIERIA AERONAUTICA\CV DIEGO ALEJANDRO CEPEDA CUARTAS\CFD MENTORINGSHIP PRO
SIMPLE_Algorithm_Project.m
SIMPLE_corroboration.m

```
            %Area equations
            di=sqrt(4*Ai/pi);
            do=sqrt(4*Ao/pi);
            hy=(di-do)/2;
            theta=atand(hy/(L));

            %Velocity grid
            dx=L/n(a);
            A_v=zeros(1,n(a)+1);
            A_v(1)=Ai;

            for i=2:length(A_v)
                A_v(i)=(pi/4)*(2*(L-(i-1)*dx)*tand(theta)+do)^2;
            end

            %Pressure grid
            A_p=zeros(1,n(a));
            for j=1:length(A_p)
                A_p(j)=(pi/4)*(2*(L-((j-1)*dx+(dx/2)))*tand(theta)+do)^2;
            end
```

Figure 2. Areas calculation and storing.

As the same way, the velocity and pressure values will be stored in a matrix with the same length of the number of nodes, being n for the velocity and n+1 for the pressure.


Editor - E:\Desktop\INGENIERIA AERONAUTICA\CV DIEGO ALEJANDRO CEPEDA C
SIMPLE_Algorithm_Project.m
SIMPLE_corroboration.m

```
            %Guessed values
            %Guessed Velocity:
            u_star=zeros(1,n(a));

            for i=1:length(u_star)
                u_star(i)=m_s(i)/(rho*A_p(i));
            end
            u_star_old=u_star;

            %Guessed Pressure:
            P_star=zeros(1,n(a)+1);

            for j=1:length(P_star)
                P_star(j)=Po-(Po/(n(a)))*(j-1);
            end
```

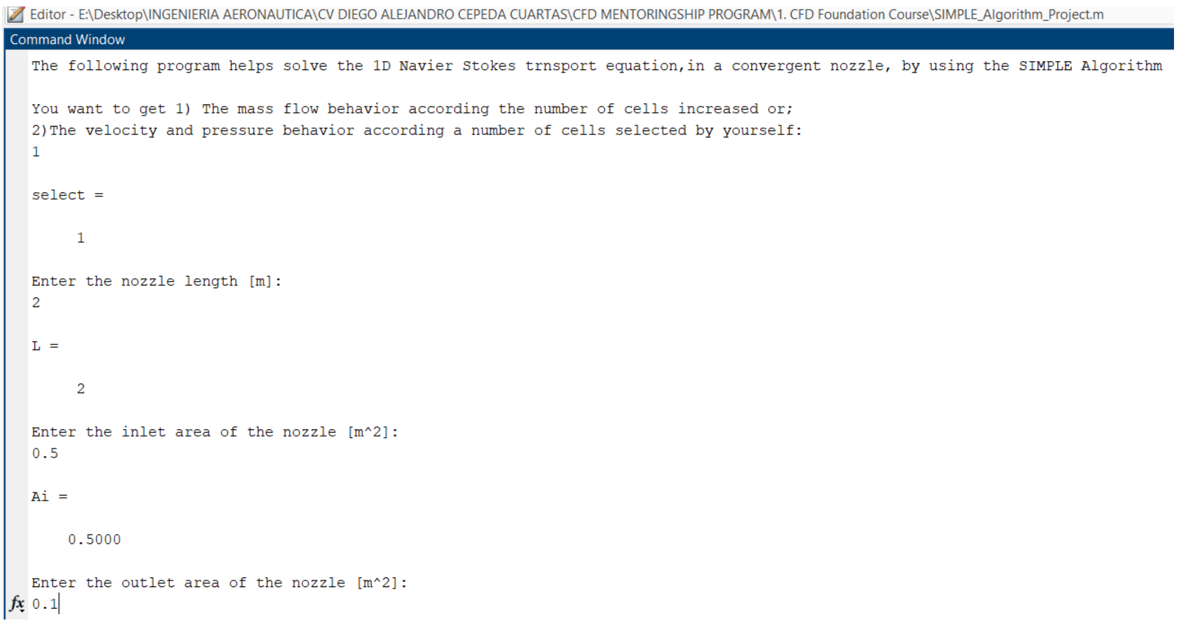
Figure 3. Guessed pressure and velocity storing.

The guessed velocity value is obtained from the mass flow equation, and the guessed pressure value from divide the inlet pressure over the number of nodes; the outlet pressure in this project will be always 0 Pa and the inlet will be equal to 10 Pa stagnation pressure.

The pressure and velocity correction and, the pressure and velocity values will have the same matrix length as the guessed pressure. All these values are entered and iterate into the SIMPLE algorithm equations, obtaining the code shown in section B of the appendix.

The section A was developed because of the book exercise does not divide symmetrically the nozzle sections, so that the areas in this section will be always the same, together with the cells split, being 4 cells. The area division assumption could have been a problem in the book, if the exercise had considered the diffusion term.

Likewise, the initial parameters as density and analytical mass flow rate were also taken from the book, being 1 $\frac{Kg}{m^3}$ of density and 0.44721 $\frac{Kg}{s}$ of mass flow rate. The appendix B code counts with two options; the first option shows the mass flow behavior every time that the number of cells is increased and, the second option is used for getting the pressure and velocity profile along the nozzle length.

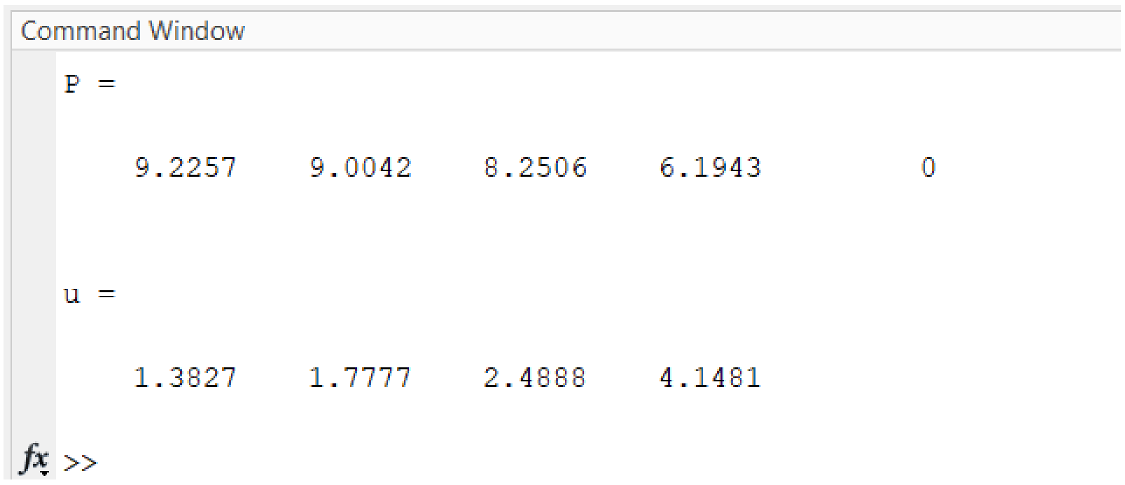


```
Editor - E:\Desktop\INGENIERIA AERONAUTICA\CV DIEGO ALEJANDRO CEPEDA CUARTAS\CFD MENTORINGSHIP PROGRAM\1. CFD Foundation Course\SIMPLE_Algorithm_Project.m
Command Window
The following program helps solve the 1D Navier Stokes trnsport equation,in a convergent nozzle, by using the SIMPLE Algorithm

You want to get 1) The mass flow behavior according the number of cells increased or;
2)The velocity and pressure behavior according a number of cells selected by yourself:
1

select =

     1

Enter the nozzle length [m]:
2

L =

     2

Enter the inlet area of the nozzle [m^2]:
0.5

Ai =

    0.5000

Enter the outlet area of the nozzle [m^2]:
fx 0.1
```

Figure 4. Appendix B code environment.

RESULTS

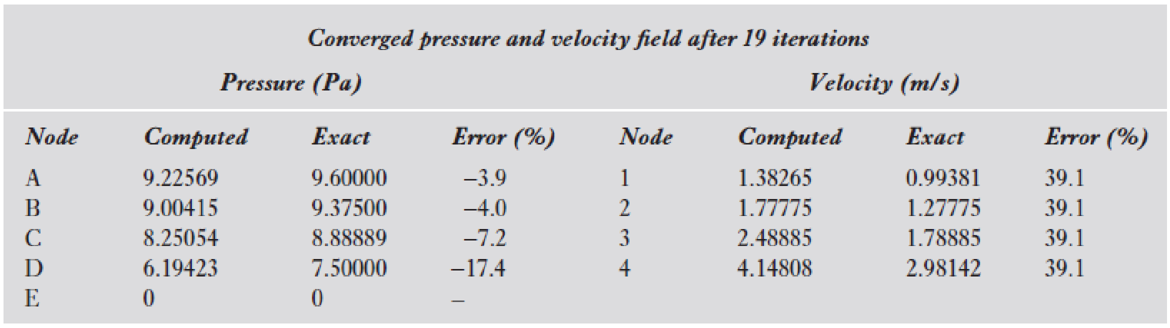
The results obtained by running the code described in the section A from the appendix is:



```
Command Window
P =

    9.2257    9.0042    8.2506    6.1943         0


u =

    1.3827    1.7777    2.4888    4.1481

fx >>
```

Figure 5. Appendix A code, results.

Now, the results obtained in the "An Introduction to Computational Fluid Dynamics - The Finite Volume Method" book are shown below.



Converged pressure and velocity field after 19 iterations

Pressure (Pa)				Velocity (m/s)			
Node	*Computed*	*Exact*	*Error (%)*	*Node*	*Computed*	*Exact*	*Error (%)*
A	9.22569	9.60000	−3.9	1	1.38265	0.99381	39.1
B	9.00415	9.37500	−4.0	2	1.77775	1.27775	39.1
C	8.25054	8.88889	−7.2	3	2.48885	1.78885	39.1
D	6.19423	7.50000	−17.4	4	4.14808	2.98142	39.1
E	0	0	–				

Figure 6. Book results. (H. K., & Malalasekera, W. 2010)

It can be noticed that the results obtained from the code are exactly similar to the results obtained in the book. Thus, the first option described in the section B from the appendix gets the following mass flow results.

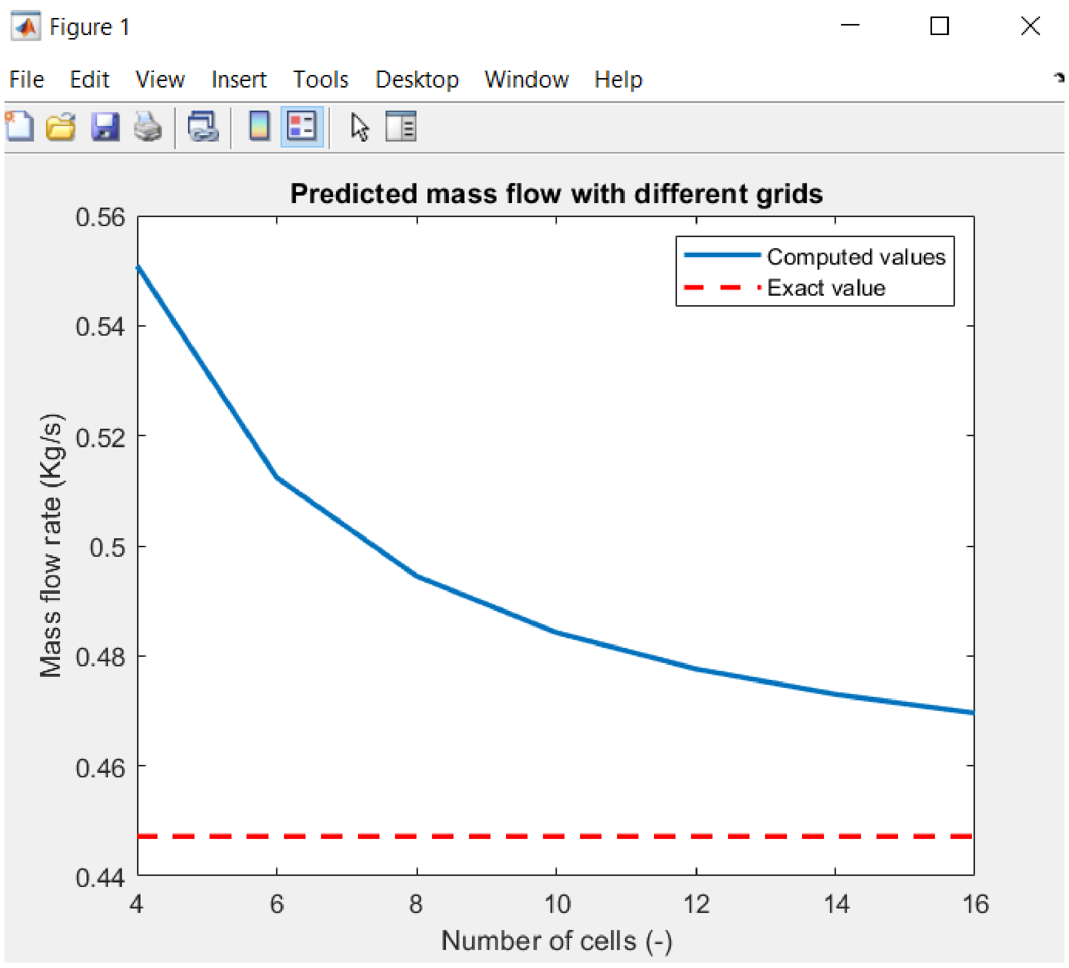


Figure 8. Mass flow result obtained by splitting the length in a determined number of cells.

Similar to the value obtained in the appendix A, the graphic plotted in the first option of the appendix B can be compared with the graphic obtained in the book.

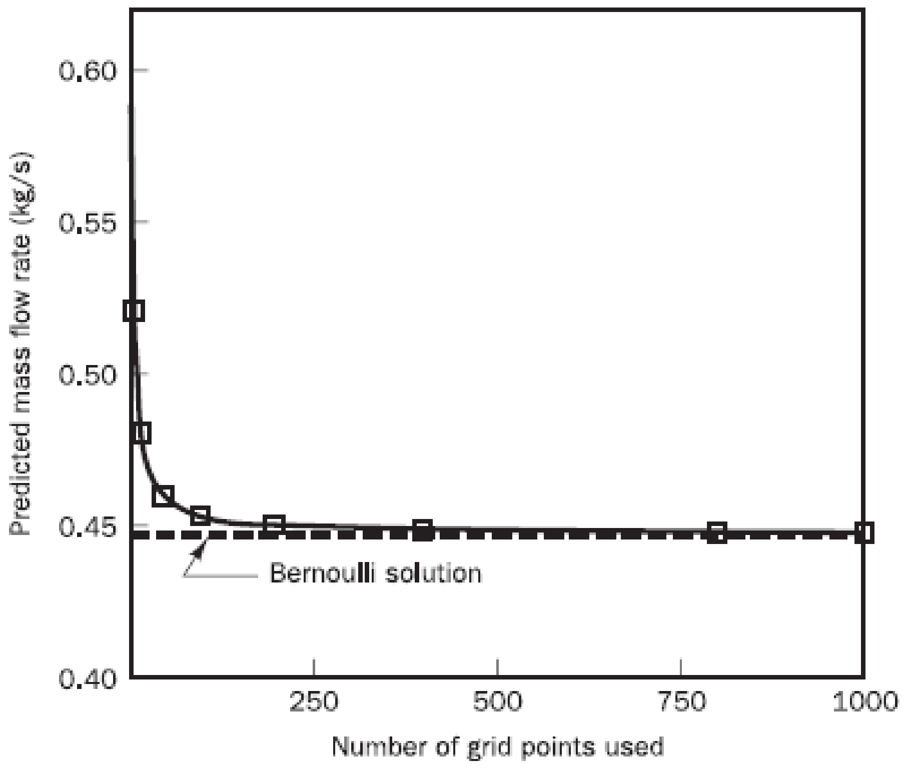


Figure 8. Mass flow result obtained by splitting the length in a determined number of cells. (H. K., & Malalasekera, W. 2010)

Finally, the results obtained in the second option from the appendix B code are:

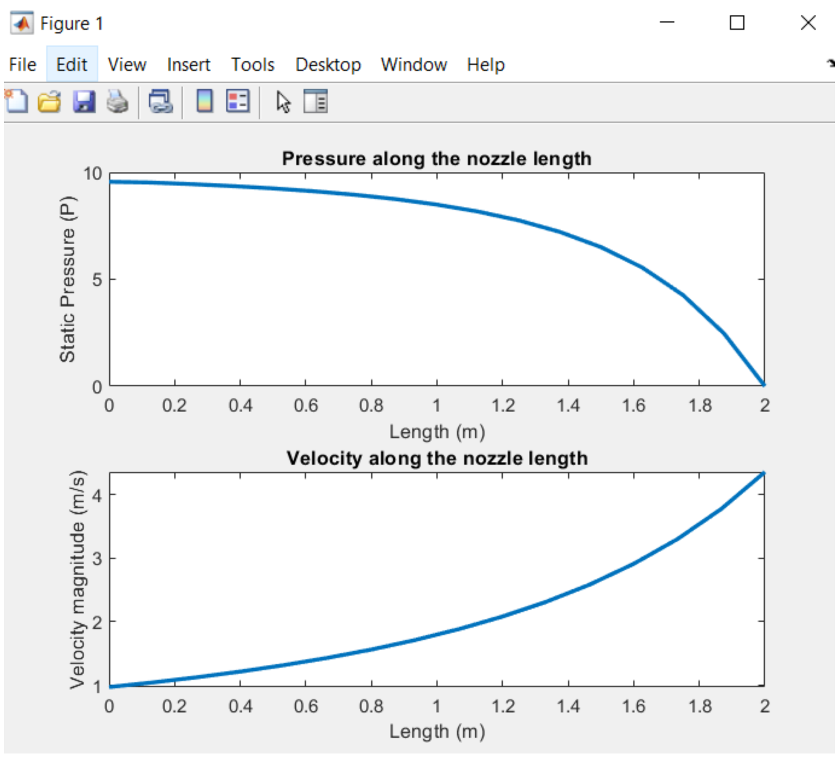


Figure 9. Predicted velocity and pressure profile along the convergent nozzle.

CONCLUSIONS

The number of cells in the first part of the appendix B section was not able to exceed the number 17, obtaining a fluctuation in the results because the convection term cannot be stabilized. However, the under-relaxation method helps converge the pressure.

The SIMPLE algorithm becomes easier the way to solve the Navier-Stokes equation and the way to simplify the equations can be well understood as for beginners as for advanced CFD developers.

REFERENCES

H. K., & Malalasekera, W. (2010). An introduction to computational fluid dynamics: The Finite Volume Method. Pearson/Prentice Hall.

APPENDIX

A. SIMPLE Algorithm – Corroboration

```
%SIMPLE Algorithm Project - Program Corroboration
%Diego Alejndro Cepeda Cuartas

%In this case, the SIMPLE Algorithm will be preformed to calculate the
%solution of a determined case, porposed by the "An Introduction to
% Computational Fluid Dynamics - The Finite Volume Method" book...

clear all
clc

display(['The following program helps solve the 1D Navier Stokes trnsport equation,'
...
'in a convergent nozzle, by using the SIMPLE Algorithm'])

n=4 %Number of cells of the exercise
L=input("Enter the nozzle length [m]:\n")

rho=1; %Kg/m^3
Po=10; %Pa
P_end=0; %Pa
x=linspace(0,L,n);

%Area equations
%Velocity grid
dx=L/n;
A_v=zeros(1,n+1);
A_v=[0.5 0.4 0.3 0.2 0.1];


%Pressure grid
A_p=zeros(1,n);
A_p=[0.45 0.35 0.25 0.15];

%Mass flow guessed
m_s=ones(1,n); %Kg/s

%Guessed values
%Guessed Velocity:
u_star=zeros(1,n);

for i=1:length(u_star)
    u_star(i)=m_s(i)/(rho*A_p(i));
end
u_star_old=u_star;

%Guessed Pressure:
P_star=zeros(1,n+1);

for j=1:length(P_star)
    P_star(j)=Po-(Po/(n))*(j-1);
end

d=zeros(1,n);

%Pressure correction
```

```
P_correction=ones(1,n+1);
P_correction(1)=0;
P_correction(end)=0;
P_correction_old=P_correction;

%Velocity
u=zeros(1,n);

%Pressure
P=zeros(1,n+1);

s=1;
tol=1e-7;
err=5;

while err>tol
    tol_p=1e-9;
    error=5;
    GS=0;

    %1-D Momentum - guessed values
    for i=1:length(u_star)
        if s==1
            if i==1
                %u guessed updating
                F_w=(rho*((u_star(i)+u_star(i+1))/2)*A_v(i+1));
                F_a=(rho*u_star_old(i)*A_p(i));
                a_w=F_w+(1/2)*F_a*(A_p(i)/A_v(i))^2;
                a_p=0 ;
                d(i)=A_p(i)/a_w;

                u_star(i)=d(i)*(Po-
                P_star(i+1))+(F_a*u_star_old(i)*(A_p(i)/A_v(i))/a_w); %New guessed
                value
            elseif i==length(u_star)
                %u guessed updating
                F_e=rho*((u_star_old(i)+u_star_old(i-1))/2)*A_v(i);
                F_b=rho*(A_p(i))*u_star(i);
                a_e=F_b;
                a_p=F_e;
                d(i)=A_p(i)/a_e;
                r_p=a_p/a_e;

                u_star(i)=(a_p*u_star(i-1)/a_e)+d(i)*(P_star(i)-P_star(i+1));

            else
                 %u guessed updating
                F_w=rho*((u_star_old(i-1)+u_star(i))/2)*A_v(i);
                F_e=rho*((u_star(i+1)+u_star(i))/2)*A_v(i+1);
                a_p=F_e;
                a_w=F_w;
                d(i)=A_p(i)/a_p;

                u_star(i)=(u_star(i-1)*a_w/a_p)+d(i)*(P_star(i)-   P_star(i+1));
            end
        else
            if i==1
                %u guessed updating
                F_w=(rho*((u_star_old(i)+u_star_old(i+1))/2)*A_v(i+1));
                F_a=(rho*u_star_old(i)*A_p(i));
                a_w=(F_w+(1/2)*F_a*(A_p(i)/A_v(i))^2)/0.8;
                a_p=0;
                d(i)=A_p(i)/a_w;
                m=(1-0.8)*(a_w)*u_star_old(i);

                u_star(i)=d(i)*(Po-
                P_star(i+1))+(F_a*u_star_old(i)*(A_p(i)/A_v(i))+m)/a_w;%New guessed
                value

        elseif i==length(u_star)
                %u guessed updating
                F_e=rho*((u_star_old(i)+u_star_old(i-1))/2)*A_v(i);
                F_b=rho*(A_p(i))*u_star_old(i);
                a_e=F_b/0.8;
                a_p=F_e;
                d(i)=A_p(i)/a_e;
                r_p=a_p/a_e;
                m=(1-0.8)*(a_e)*u_star_old(i);

                 u_star(i)=(a_p*u_star(i-1)/a_e)+d(i)*(P_star(i)-
                P_star(i+1))+(m/a_e);

        else
                %u guessed updating
                F_w=rho*((u_star_old(i-1)+u_star_old(i))/2)*A_v(i);
                F_e=rho*((u_star(i+1)+u_star(i))/2)*A_v(i+1);
                a_p=F_e/0.8;
                a_w=F_w;
                d(i)=A_p(i)/a_p;
                m=(1-0.8)*(a_p)*u_star_old(i);

                 u_star(i)=(u_star(i-1)*a_w/a_p)+d(i)*(P_star(i)-
                P_star(i+1))+(m/a_p);

        end
    end
end

while error>tol_p
    for i=2:length(P_correction_old)-1
        a_a=rho*A_p(i-1)*d(i-1);
        a_e=rho*A_p(i)*d(i);
        a_w=a_a+a_e;
        b=rho*A_p(i-1)*u_star(i-1)-rho*A_p(i)*u_star(i);
        P_correction(i)=(a_e*P_correction_old(i+1)+a_a*P_correction(i-1)+b)/a_w;
    end
    error=max(abs(P_correction-P_correction_old));
    P_correction_old=P_correction;
    GS=GS+1;
end

%Velocity
for i=1:length(u)
    u(i)=u_star(i)+d(i)*(P_correction(i)-P_correction(i+1));
end
```

```
        %Pressure
        if s==1
            P(1)=Po-(rho/2)*(u(1)^2)*(A_p(1)/A_v(1))^2;
            for j=2:length(P)-1;
                P(j)=P_star(j)+P_correction(j);
            end
        else
            P(1)=Po-(rho/2)*(u(1)^2)*(A_p(1)/A_v(1))^2;
            for j=2:length(P)-1
                P(j)=P_star(j)+0.8*P_correction(j);
            end
        end

        for i=1:length(u)
            u(i)=0.2*u_star_old(i)+0.8*u(i);
        end

        for j=1:length(P)
            P(j)=0.2*P_star(j)+0.8*P(j);
        end

        for z=1:length(m_s)
            m_s(z)=rho*u(z)*A_p(z);
        end

        err=max(abs(u_star_old-u));
        u_star=u;
        u_star_old=u_star;
        P_star=P;
        s=s+1
    end
    P
    u
```

B. SIMPLE Algorithm – Program

```
%SIMPLE Algorithm Project
%Diego Alejndro Cepeda Cuartas

%In this case, the SIMPLE Algorithm will be preformed to calculate the
%solution of a determined case, porposed by the "An Introduction to
% Computational Fluid Dynamics - The Finite Volume Method" book...

clear all
clc

display(['The following program helps solve the 1D Navier Stokes trnsport equation,' ...
    'in a convergent nozzle, by using the SIMPLE Algorithm'])
display(' ')
select=input("You want to get 1) The mass flow behavior according the number of cells
increased or;\n2)" + ...
    "The velocity and pressure behavior according a number of cells selected by
yourself:\n")
if select==1
    L=input("Enter the nozzle length [m]:\n")
    Ai=input("Enter the inlet area of the nozzle [m^2]:\n")
    Ao=input("Enter the outlet area of the nozzle [m^2]:\n")

    n=[4,6,8,10,12,14,16];
    o=ones(1,length(n));

    for a=1:length(n);
        rho=1; %Kg/m^3
        Po=10; %Pa
        P_end=0; %Pa
        x=linspace(0,L,n(a));

        %Area equations
        di=sqrt(4*Ai/pi);
        do=sqrt(4*Ao/pi);
        hy=(di-do)/2;
        theta=atand(hy/(L));

        %Velocity grid
        dx=L/n(a);
        A_v=zeros(1,n(a)+1);
        A_v(1)=Ai;

        for i=2:length(A_v)
            A_v(i)=(pi/4)*(2*(L-(i-1)*dx)*tand(theta)+do)^2;
        end

        %Pressure grid
        A_p=zeros(1,n(a));
        for j=1:length(A_p)
            A_p(j)=(pi/4)*(2*(L-((j-1)*dx+(dx/2)))*tand(theta)+do)^2;
        end

        %Mass flow guessed
        m_s=ones(1,n(a)); %Kg/s

        %Guessed values
        %Guessed Velocity:
        u_star=zeros(1,n(a));

        for i=1:length(u_star)
            u_star(i)=m_s(i)/(rho*A_p(i));
        end
        u_star_old=u_star;

        %Guessed Pressure:
        P_star=zeros(1,n(a)+1);

        for j=1:length(P_star)
            P_star(j)=Po-(Po/(n(a)))*(j-1);
        end

        d=zeros(1,n(a));

        %Pressure correction
        P_correction=ones(1,n(a)+1);
        P_correction(1)=0;
        P_correction(end)=0;
        P_correction_old=P_correction;


        %Velocity
        u=zeros(1,n(a));
        u_old=u;
        %Pressure
        P=zeros(1,n(a)+1);
        P_old=P;

        s=1;
        tol=1e-7;
        err=5;


        while err>tol
            tol_p=1e-9;
            error=5;
            GS=0;

            %1-D Momentum - guessed values
            for i=1:length(u_star)
                if s==1
                    if i==1
                            %u guessed updating
                            F_w=(rho*((u_star(i)+u_star(i+1))/2)*A_v(i+1));
                            F_a=(rho*u_star_old(i)*A_p(i));
                            a_w=F_w+(1/2)*F_a*(A_p(i)/A_v(i))^2;
                            a_p=0 ;
                            d(i)=A_p(i)/a_w;

                            u_star(i)=d(i)*(Po-
P_star(i+1))+(F_a*u_star_old(i)*(A_p(i)/A_v(i))/a_w)%New guessed value

                    elseif i==length(u_star)
                            %u guessed updating
                            F_e=rho*((u_star_old(i)+u_star_old(i-1))/2)*A_v(i);
                            F_b=rho*(A_p(i))*u_star(i)
                    a_e=F_b;
                    a_p=F_e;
                    d(i)=A_p(i)/a_e;
                    r_p=a_p/a_e;

                    u_star(i)=(a_p*u_star(i-1)/a_e)+d(i)*(P_star(i)-P_star(i+1))

                else
                    %u guessed updating
                    F_w=rho*((u_star_old(i-1)+u_star(i))/2)*A_v(i);
                    F_e=rho*((u_star(i+1)+u_star(i))/2)*A_v(i+1);
                    a_p=F_e;
                    a_w=F_w;
                    d(i)=A_p(i)/a_p;

                    u_star(i)=(u_star(i-1)*a_w/a_p)+d(i)*(P_star(i)-P_star(i+1))
                end
            else
                if i==1
                    %u guessed updating

                  F_w=(rho*((u_star_old(i)+u_star_old(i+1))/2)*A_v(i+1));
                    F_a=(rho*u_star_old(i)*A_p(i));
                    a_w=(F_w+(1/2)*F_a*(A_p(i)/A_v(i))^2)/0.8;
                    a_p=0;
                    d(i)=A_p(i)/a_w;
                    m=(1-0.8)*(a_w)*u_star_old(i);

                    u_star(i)=d(i)*(Po-
P_star(i+1))+(F_a*u_star_old(i)*(A_p(i)/A_v(i))+m)/a_w%New guessed value

                elseif i==length(u_star)
                    %u guessed updating
                    F_e=rho*((u_star_old(i)+u_star_old(i-1))/2)*A_v(i);
                    F_b=rho*(A_p(i))*u_star_old(i)
                    a_e=F_b/0.8;
                    a_p=F_e;
                    d(i)=A_p(i)/a_e;
                    r_p=a_p/a_e;
                    m=(1-0.8)*(a_e)*u_star_old(i);

                    u_star(i)=(a_p*u_star(i-1)/a_e)+d(i)*(P_star(i)-
P_star(i+1))+(m/a_e)

                else
                    %u guessed updating
                    F_w=rho*((u_star_old(i-1)+u_star_old(i))/2)*A_v(i);
                    F_e=rho*((u_star(i+1)+u_star(i))/2)*A_v(i+1);
                    a_p=F_e/0.8;
                    a_w=F_w;
                    d(i)=A_p(i)/a_p;
                    m=(1-0.8)*(a_p)*u_star_old(i);

                    u_star(i)=(u_star(i-1)*a_w/a_p)+d(i)*(P_star(i)-
P_star(i+1))+(m/a_p)

                end
            end
        end
```

```
            while error>tol_p
                for i=2:length(P_correction_old)-1
                    a_a=rho*A_p(i-1)*d(i-1);
                    a_e=rho*A_p(i)*d(i);
                    a_w=a_a+a_e;
                    b=rho*A_p(i-1)*u_star(i-1)-rho*A_p(i)*u_star(i);
                    P_correction(i)=(a_e*P_correction_old(i+1)+a_a*P_correction(i-
1)+b)/a_w;
                end
                error=max(abs(P_correction-P_correction_old));
                P_correction_old=P_correction;
                GS=GS+1
            end

            %Velocity
            for i=1:length(u)
                u(i)=u_star(i)+d(i)*(P_correction(i)-P_correction(i+1))
            end

            %Pressure
            if s==1
                P(1)=Po-(rho/2)*(u(1)^2)*(A_p(1)/A_v(1))^2
                for j=2:length(P)-1
                    P(j)=P_star(j)+P_correction(j)
                end
            else
                P(1)=Po-(rho/2)*(u(1)^2)*(A_p(1)/A_v(1))^2
                for j=2:length(P)-1
                    P(j)=P_star(j)+0.8*P_correction(j)
                end
            end


            for i=1:length(u)
                u(i)=0.2*u_star_old(i)+0.8*u(i)
            end

            for j=1:length(P)
                P(j)=0.2*P_star(j)+0.8*P(j)
            end


            for z=1:length(m_s)
                m_s(z)=rho*u(z)*A_p(z);
            end

            err=max(abs(u_star_old-u));
            u_star=u;
            u_star_old=u_star;
            P_star=P;
            s=s+1
        end
        o(a)=m_s(a)
    end
    plot(n,o,'LineWidth',2)
    exact=ones(1,length(n))*0.44721;
    hold on
    plot(n,exact,'r--','LineWidth',2)
```

```
        xlabel("Number of cells (-)")
        ylabel("Mass flow rate (Kg/s)")
        title("Predicted mass flow with different grids")
        legend("Computed values","Exact value")

    elseif select==2
        n=input("Enter the number of cells (No greater than 17!):\n")
        L=input("Enter the nozzle length [m]:\n")
        Ai=input("Enter the inlet area of the nozzle [m^2]:\n")
        Ao=input("Enter the outlet area of the nozzle [m^2]:\n")

        rho=1; %Kg/m^3
        Po=10; %Pa
        P_end=0; %Pa
        x=linspace(0,L,n); %Velocity grid
        x_p=linspace(0,L,n+1); %Pressure grid

        %Area equations
        di=sqrt(4*Ai/pi);
        do=sqrt(4*Ao/pi);
        hy=(di-do)/2;
        theta=atand(hy/(L));

        %Velocity grid
        dx=L/n;
        A_v=zeros(1,n+1);
        A_v(1)=Ai;

        for i=2:length(A_v)
            A_v(i)=(pi/4)*(2*(L-(i-1)*dx)*tand(theta)+do)^2;
        end

        %Pressure grid
        A_p=zeros(1,n);
        for j=1:length(A_p)
            A_p(j)=(pi/4)*(2*(L-((j-1)*dx+(dx/2)))*tand(theta)+do)^2;
        end

        %Mass flow guessed
        m_s=ones(1,n); %Kg/s

        %Guessed values

        %Guessed Velocity:
        u_star=zeros(1,n);

        for i=1:length(u_star)
            u_star(i)=m_s(i)/(rho*A_p(i));
        end
        u_star_old=u_star;

        %Guessed Pressure:
        P_star=zeros(1,n+1);

        for j=1:length(P_star)
            P_star(j)=Po-(Po/(n))*(j-1);
        end

        d=zeros(1,n);
    %Pressure correction
    P_correction=ones(1,n+1);
    P_correction(1)=0;
    P_correction(end)=0;
    P_correction_old=P_correction;


    %Velocity old
    u=zeros(1,n);
    u_old=u;
    %Pressure old
    P=zeros(1,n+1);
    P_old=P;

    s=1;
    tol=1e-7;
    err=5;


    while err>tol
        tol_p=1e-9;
        error=5;
        GS=0;

        %1-D Momentum - guessed values
        for i=1:length(u_star)
            if s==1
                if i==1
                        %u guessed updating
                        F_w=(rho*((u_star(i)+u_star(i+1))/2)*A_v(i+1));
                        F_a=(rho*u_star_old(i)*A_p(i));
                        a_w=F_w+(1/2)*F_a*(A_p(i)/A_v(i))^2;
                        a_p=0 ;
                        d(i)=A_p(i)/a_w;

                        u_star(i)=d(i)*(Po-
P_star(i+1))+(F_a*u_star_old(i)*(A_p(i)/A_v(i))/a_w)%New guessed value

                elseif i==length(u_star)
                        %u guessed updating
                        F_e=rho*((u_star_old(i)+u_star_old(i-1))/2)*A_v(i);
                        F_b=rho*(A_p(i))*u_star(i)
                        a_e=F_b;
                        a_p=F_e;
                        d(i)=A_p(i)/a_e;
                        r_p=a_p/a_e;

                        u_star(i)=(a_p*u_star(i-1)/a_e)+d(i)*(P_star(i)-P_star(i+1))

                else
                        %u guessed updating
                        F_w=rho*((u_star_old(i-1)+u_star(i))/2)*A_v(i);
                        F_e=rho*((u_star(i+1)+u_star(i))/2)*A_v(i+1);
                        a_p=F_e;
                        a_w=F_w;
                        d(i)=A_p(i)/a_p;

                        u_star(i)=(u_star(i-1)*a_w/a_p)+d(i)*(P_star(i)-P_star(i+1))
                end
            else
                if i==1
                        %u guessed updating
                        F_w=(rho*((u_star_old(i)+u_star_old(i+1))/2)*A_v(i+1));
                        F_a=(rho*u_star_old(i)*A_p(i));
                        a_w=(F_w+(1/2)*F_a*(A_p(i)/A_v(i))^2)/0.8;
                        a_p=0;
                        d(i)=A_p(i)/a_w;
                        m=(1-0.8)*(a_w)*u_star_old(i);

                        u_star(i)=d(i)*(Po-
P_star(i+1))+(F_a*u_star_old(i)*(A_p(i)/A_v(i))+m)/a_w%New guessed value

                elseif i==length(u_star)
                        %u guessed updating
                        F_e=rho*((u_star_old(i)+u_star_old(i-1))/2)*A_v(i);
                        F_b=rho*(A_p(i))*u_star_old(i)
                        a_e=F_b/0.8;
                        a_p=F_e;
                        d(i)=A_p(i)/a_e;
                        r_p=a_p/a_e;
                        m=(1-0.8)*(a_e)*u_star_old(i);

                        u_star(i)=(a_p*u_star(i-1)/a_e)+d(i)*(P_star(i)-
P_star(i+1))+(m/a_e)

                else
                        %u guessed updating
                        F_w=rho*((u_star_old(i-1)+u_star_old(i))/2)*A_v(i);
                        F_e=rho*((u_star(i+1)+u_star(i))/2)*A_v(i+1);
                        a_p=F_e/0.8;
                        a_w=F_w;
                        d(i)=A_p(i)/a_p;
                        m=(1-0.8)*(a_p)*u_star_old(i);

                        u_star(i)=(u_star(i-1)*a_w/a_p)+d(i)*(P_star(i)-
P_star(i+1))+(m/a_p)

                end
            end
        end

        while error>tol_p
            for i=2:length(P_correction_old)-1
                a_a=rho*A_p(i-1)*d(i-1);
                a_e=rho*A_p(i)*d(i);
                a_w=a_a+a_e;
                b=rho*A_p(i-1)*u_star(i-1)-rho*A_p(i)*u_star(i);
                P_correction(i)=(a_e*P_correction_old(i+1)+a_a*P_correction(i-1)+b)/a_w;
            end
            error=max(abs(P_correction-P_correction_old));
            P_correction_old=P_correction;
            GS=GS+1
        end

        %Velocity
        for i=1:length(u)
            u(i)=u_star(i)+d(i)*(P_correction(i)-P_correction(i+1))
        end

        %Pressure
        if s==1
            P(1)=Po-(rho/2)*(u(1)^2)*(A_p(1)/A_v(1))^2
            for j=2:length(P)-1
                P(j)=P_star(j)+P_correction(j)
            end
        else
            P(1)=Po-(rho/2)*(u(1)^2)*(A_p(1)/A_v(1))^2
            for j=2:length(P)-1
                P(j)=P_star(j)+0.8*P_correction(j)
            end
        end


        for i=1:length(u)
            u(i)=0.2*u_star_old(i)+0.8*u(i)
        end

        for j=1:length(P)
            P(j)=0.2*P_star(j)+0.8*P(j)
        end


        for z=1:length(m_s)
            m_s(z)=rho*u(z)*A_p(z);
        end

        err=max(abs(u_star_old-u));
        u_star=u;
        u_star_old=u_star;
        P_star=P;
        s=s+1
    end
    subplot(2,1,1)
    plot(x_p,P,'Linewidth',2)
    title("Pressure along the nozzle length")
    xlabel("Length (m)")
    ylabel("Static Pressure (P)")
    subplot(2,1,2)
    plot(x,u,'Linewidth',2)
    title("Velocity along the nozzle length")
    xlabel("Length (m)")
    ylabel("Velocity magnitude (m/s)")

end
```