Research on 2D SDE Method of Fluid Field Effect above Airfoil at Transonic Speed

Xinhao Li

School of Future Technology, Harbin Institute of Technology, Harbin, 150001, China

2021112257@stu.hit.edu.cn

Abstract. In this research, a Python-centric approach is proposed for simulating fluid dynamics over transonic airfoils. The study employs the finite difference method to solve the small disturbance equation (SDE), incorporating Newman boundary conditions. Iterative computations are adeptly managed using the successive over-relaxation (SOR) method. These methodologies, as validated through rigorous code testing, enable precise calculations of velocity distributions and vivid visualizations of fluid velocity streamlines. For this investigation, the airfoil is conceptualized as a minor obstruction and is abstracted into a trigonometric function. A pivotal aspect of this study is the exploration of the interplay between the Mach number and the 'shark angle'—a critical determinant in aerodynamic performance. Simulation outcomes reveal an inverse relationship between the Mach number and the airfoil's shark angle; a surge in the Mach number corresponds to a reduction in the shark angle. These findings, encapsulated in flow field diagrams, offer invaluable insights for the strategic design and optimization of airfoils, aiming to maximize lift while concurrently curtailing drag, thereby enhancing overall aerodynamic efficiency.

Keywords: Transonic speed, airfoil, SDE, fluid field, computational fluid dynamics.

1. Introduction

Fluid dynamics is the discipline that studies fluid motion, including the interaction between two fluids when they come into contact [1]. Weather forecasting, calculating the forces and moments acting on airplanes, and determining the oil flow rate in oil pipelines are excellent fluid dynamics applications. Even transportation engineering uses some of these concepts. Transportation is seen as a continuous fluid, and determining some of the fluid's properties, such as velocity, pressure, density, and temperature, is necessary to solve a standard fluid dynamics problem. Computational fluid dynamics (CFD), which emerged as an interdisciplinary discipline of mathematics, fluid mechanics, and computer science in the 1950s due to the advent of computers, is currently recognized as the third way to learn fluid dynamics in addition to theory and experiment [2]. Its primary research focuses on simulating and analyzing fluid mechanics issues and using computers and numerical approaches to solve fluid mechanics control equations. Based on CFD, this article uses the finite difference method to investigate fluid flow above an airfoil under transonic conditions and solves the small disturbance equation (SDE) [3].

Studying fluid flow above airfoils under transonic conditions has important application value in upgrading and upgrading aircraft. Airfoils bearing heavier aerodynamic loads have been designed due to modern practice. However, this is restricted and subject to effective viscous effect regulation. Therefore, fundamental fluid dynamics problems immediately include concerns of substantial practical value and the limitations of the efficacy of the methods addressed. The article has tried to concentrate on this inquiry between fundamental fluid mechanics and real aerodynamics since this is the transonic airfoil theory's intermediate location [4]. Firstly, the article will discuss the application of the SDE method in fluid dynamics and the derivation of the SDE method. Then, a 2D SDE model was designed as the basis, and the transonic flow fields around the disturbance were simulated using Python programming. Finally, it will compare the effects of fluid flow on an airfoil by changing the flow velocity.

2. Method

2.1. Theoretical Derivation

The FPE is a nonlinear partial differential equation (a simplified form of the Navier-Stokes equations) that describes inviscid, incompressible fluid flow [5]. It assumes that the fluid is irrotational, implying that the flow has no vorticity.

For steady flow, the continuity equation $\frac{\partial \rho}{\partial t} + div(\rho \nabla \Phi) = 0$ with wall BC. $v_n = \frac{\partial \Phi}{\partial n} = 0$, it can be converted to $\frac{\rho}{\rho_0} = (1 - \frac{|\nabla \Phi|^2}{2H_0})^{\frac{1}{(\gamma-1)}}$. After that, the FPE could be derived.

$$(1 - {M_x}^2)\Phi_{xx} + (1 - {M_y}^2)\Phi_{yy} + (1 - {M_z}^2)\Phi_{zz} - 2M_xM_y\Phi_{xy} - 2M_xM_z\Phi_{xz} - 2M_yM_z\Phi_{yz} = 0 \quad (1)$$

The full potential equations can be simplified for thin obstacles like thin airfoils. And this article restricts the discussion to 2D. The article considers disruption to uniform flow at a velocity of magnitude U_∞ in the x-direction since the obstruction is "small" and its impact on the flow is minimal. There is a representation of the possibilities.

$$\Phi = U_\infty(x + \varphi) \quad (2)$$

From the potential, the article uses the formulas to represent the velocity:

$$u = U_\infty(1 + \varphi_x) \quad (3)$$

$$v = U_\infty \varphi_y \quad (4)$$

The FPE simplified form can be $(1 - {M_\infty}^2)\varphi_{xx} + \varphi_{yy} = 0$. Call it SDE. To be more accurate, the following equation can be used:

$$(1 - {M_x}^2)\varphi_{xx} + \varphi_{yy} = 0 \quad (5)$$

The SDE method has been proven useful in understanding fluid flow characteristics. Compared to another commonly used full potential equation (FPE) method, SDE is a relatively simplified model. It is mainly used for passing through thin obstacles.

The boundary condition becomes

$$v = (U_\infty + u)f'(x) \approx U_\infty f'(x) \quad (6)$$

Where f(x) is the shape of the airfoil.

2.2. Numerical Approximation

The article discretizes the SDE using finite differences and derives the following results:

For M, x<1 (subsonic condition):

$$(1 - M^2)(\varphi_{i+1,j} - 2\varphi_{i,j} + \varphi_{i-1,j}) + (\varphi_{i,j+1} - 2\varphi_{i,j} + \varphi_{i,j-1}) = 0 \quad (7)$$

For M, x>1 (supersonic condition):

$$(1 - M^2)(\varphi_{ij} - 2\varphi_{i-1,j} + \varphi_{i-2,j}) + (\varphi_{i,j+1} - 2\varphi_{i,j} + \varphi_{i,j-1}) = 0 \quad (8)$$

The small disturbance equation for the transonic case combines the two situations above [6, 7].

$$\mu_{i,j}k_{i-1,j}(\varphi_{ij} - 2\varphi_{i-1,j} + \varphi_{i-2,j}) + (\varphi_{i,j+1} - 2\varphi_{i,j} + \varphi_{i,j-1}) + (1 - \mu_{i,j})k_{i,j}(\varphi_{i+1,j} - 2\varphi_{i,j} + \varphi_{i-1,j}) = 0 \quad (9)$$

Among the equation $k_{i,j} = 1 - {M_\infty}^2 - (1 + \gamma)M_\infty \frac{\varphi_{i+1,j} - \varphi_{i-1,j}}{2\Delta x}$, and μ equals 0 for M<1 and 1 for M>1.

For the boundary condition, U_∞ is the flow velocity magnitude, and f(x) is the shape of the boundary. The value of U_∞ can be calculated, and it is $\sqrt{\frac{(\gamma-1)*h_0*{M_\infty}^2}{1+0.5*{M_\infty}^2*(\gamma-1)}}$. In this article, the author

chooses the airfoil and simplifies the 2D shape of the airfoil to a trigonometric function. The shape is modeled as a sinusoidal wall in the (1, 2) domain, while there are no obstacles in the (0, 1) and (2, 3) domains, as shown in Fig. 1.

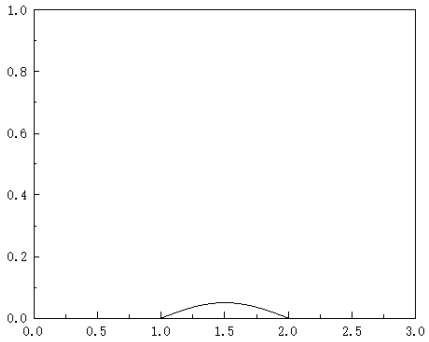


Figure 1. The 2D shape of the airfoil (Photo/Picture credit: Original)

2.3. Code Analysis

2.3.1 The BC functions

The article chooses the Newman boundary condition $-\frac{\partial \varphi}{\partial y} = M_x f'(x)$ and define function BC_F to present the boundary condition in which the $f'(x)$ is $cos(pi * (x - 1))$, as shown in Fig. 2.

```
def BC_F(x, beta):
    pi = np.pi
    if x < 1 or x > 2:
        return 0.0
    else:
        return beta * np.cos(pi * (x - 1))
```

Figure 2. The BC function (Photo/Picture credit: Original)

2.3.2 The error function

```
def error(phi):
    newphi = phi.copy()
    K = phi.copy()
    u = phi.copy()
    Nx = newphi.shape[0]
    Ny = newphi.shape[1]
    err = 0.0
    for i in range(1, newphi.shape[0] - 1):
        for j in range(1, newphi.shape[1] - 1):
            temp = np.power(M_inf, 2) + (gamma + 1) * np.power(M_inf, 2) * \
                   (newphi[i + 1, j] - newphi[i - 1, j]) / (2 * h)
            K[i, j] = 1 - temp
            if temp < 1:
                u[i, j] = 0
            else:
                u[i, j] = 1
    for i in range(2, Nx - 1):
        for j in range(1, Ny - 1):
            B = 1
            C0 = u[i, j] * K[i - 1, j] - 2 * B - 2 * ((1 - u[i, j]) * K[i, j])
            eq = C0 * newphi[i, j] + (u[i, j] * K[i - 1, j]) * \
                 (newphi[i - 2, j] - 2 * newphi[i - 1, j]) + \
                 (B * newphi[i, j + 1] + B * newphi[i, j - 1]) + \
                 ((1 - u[i, j]) * K[i, j]) * (newphi[i + 1, j] + newphi[i - 1, j])
            err += abs(eq)
    return err / (Nx * Ny)
```

Figure 3. The error function (Photo/Picture credit: Original)

Errors always exist in CFD. The author passes each point, computes the equation residuals there, and adds all equation residuals into the error function to obtain precise velocity potential errors. To determine the average error, divide the total by the number of grid points. By computing the difference between the left and right sides of the equation, this function may calculate the inaccuracy of the equation solution and estimate its size, as depicted in Fig. 3.

2.3.3 The velocity vector function

In this function, the author uses the central difference method to calculate the velocities of x and y directions at every point, respectively (computing the gradient of the velocity potential), as shown in Fig. 4.

```
def velocity_vec(phi):
    U = np.zeros((phi.shape[0] - 2, phi.shape[1] - 2))
    V = np.zeros((phi.shape[0] - 2, phi.shape[1] - 2))

    print(U.shape, V.shape)

    hx = 1 / phi.shape[0]
    hy = 1 / phi.shape[1]

    for i in range(1, phi.shape[0] - 1):
        for j in range(1, phi.shape[1] - 1):
            U[i - 1, j - 1] = 0.5 * (phi[i + 1, j] - phi[i - 1, j]) / hx
            V[i - 1, j - 1] = 0.5 * (phi[i, j + 1] - phi[i, j - 1]) / hy
    return U, V
```

Figure 4. The velocity vector function (Photo/Picture credit: Original)

2.3.4 The SOR function

Use the SOR technique to solve the Euler equation [8]. This program aims to solve the Euler equation for 2D inviscid flow and provide the accompanying flow field diagram. Solve the flux equation to determine the velocity field, and then solve the Euler equation again to determine the pressure field. Additionally, this code has undergone unique processing for bounds.

The velocity potential value in the transonic discretization equation is then updated by running the SOR iterative technique over each grid point. Mach number is represented by the corresponding coefficient temp, where k=1-temp. The introduction of u (miu) in the computation process regulates the choice of tiny perturbation equations. Set u = 1 if the local Mach number at this location exceeds 1. If not, set u=0. Calculate and update the potential function phi in the flow field, excluding the boundary through coefficients and weights. The author chose the left (inflow boundary) and bottom borders as the boundaries, and the iterative format at the boundaries was particularly treated, as shown in Fig. 5.

```
def SOR(phi,beta):
    h = 3.0 / phi.shape[0]
    newphi = phi.copy()
    sor = 1.0
    K = calculate_k(phi)
    for i in range(2, newphi.shape[0] - 1):
        for j in range(1, newphi.shape[1] - 1):
            if M_inf**2 + (Gamma + 1) * M_inf**2 * (phi[i + 1, j] - phi[i - 1, j]) / (2 * h) >= 1:
                miu = 1
            else:
                miu = 0
            A1 = miu * K[i - 1, j]
            A2 = 1
            A3 = (1 - miu) * K[i, j]
            C0 = (2 * A2 + 2 * A3 - A1)
            dp = -newphi[i, j] + (A1 * (newphi[i - 2, j] - 2*newphi[i - 1, j]) + (newphi[i, j + 1]
                + newphi[i, j - 1]) + A3 * (newphi[i + 1, j] + newphi[i - 1, j])) / C0
            newphi[i, j] += sor * dp
        j = 0
        for i in range(2, newphi.shape[0]-1):
            if M_inf**2 + (Gamma + 1) * M_inf**2 * (phi[i + 1, j] - phi[i - 1, j]) / (2 * h)>=1:
                miu=1
            else:
                miu=0
            A1 = miu*K[i-1,j]
            A2 = 1
            A3 = (1-miu)*K[i,j]
            C0 = (2*A2+2*A3-A1)
            dp = -newphi[i,j] + (2*A2 * newphi[i,j+1] +A1 *( newphi[i-2, j]-2*newphi[i-1, j])
                                + A3 * (newphi[i+1, j] + newphi[i-1,j]) - 2 * h * BC_F(i*h,beta))/C0
            newphi[i, j] += sor * dp
    return newphi
```

Figure 5. The SOR function (Photo/Picture credit: Original)

2.3.5 Main function

In this proportion, define some constant parameters first, setting M_inf as the figure ranging from 0.8 to 1.2 (transonic case) and fluid adiabatic coefficient gamma as 1.4 (the fluid is air) [9, 10]. Then, traversing each grid point, the code initializes the phi and plots the flow field diagrams (Fig. 6).

```
N = 32
H_0 = 1
Gamma = 1.4
phi = np.zeros((3 * N, N))
M_inf = 1.2
U_inf = np.sqrt((np.square(M_inf) * H_0 * (Gamma - 1)) / (1 + 0.5 * (np.square(M_inf) * (Gamma - 1))))
h = 1 / N
beta = 0.05


for i in range(3 * N):
    for j in range(N):
        phi[i, j] = U_inf * i * h

for iteration in range(1700):
    phi = SOR(phi,beta)
    if iteration % 100 == 0:
        print(iteration, error(phi))

U, V = velocity_vec(phi)
x = np.linspace(0, 3, 94)
y = np.linspace(0, 1, 30)
X, Y = np.meshgrid(x, y)

plt.streamplot(X, Y, np.transpose(U), np.transpose(V), density=1)
plt.show()
```

Figure 6. Main function (Photo/Picture credit: Original)

3. Result and Discussion

When the inflow velocity is transonic, the author selected four representative initial values of the inflow velocity. The errors and fluid velocity streamline results are shown in Figs. 7-10:

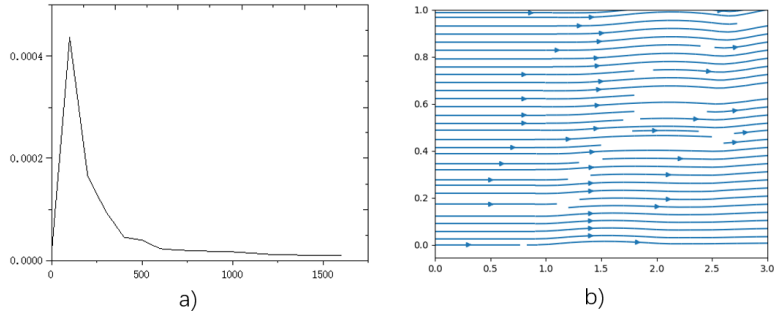


Figure 7. a) Errors when inflow Mach numbers = 0.8, b) Fluid velocity streamlines when inflow Mach numbers = 0.8 (Photo/Picture credit: Original)

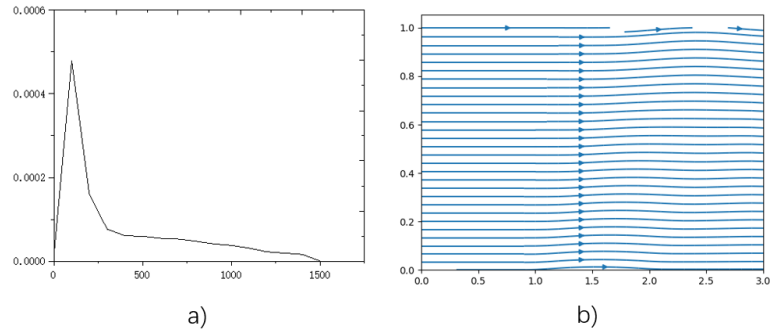


Figure 8. a) Errors when inflow Mach numbers = 0.9, b) Fluid velocity streamlines when inflow Mach numbers = 0.9 (Photo/Picture credit: Original)

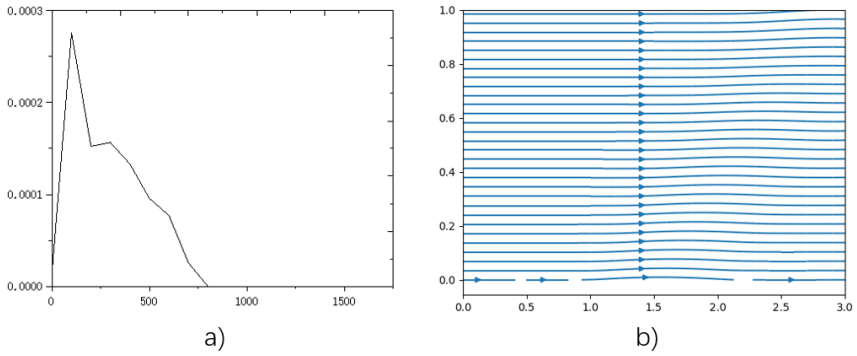


Figure 9. a) Errors when inflow Mach numbers = 1.1, b) Fluid velocity streamlines when inflow Mach numbers = 1.1 (Photo/Picture credit: Original)

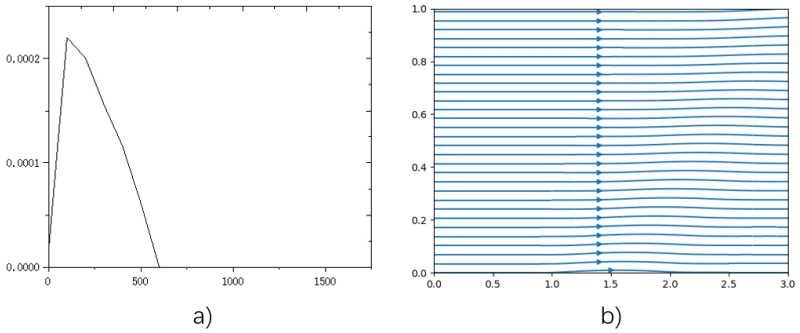


Figure 10. a) Errors when inflow Mach numbers = 1.2, b) Fluid velocity streamlines when inflow Mach numbers = 1.2 (Photo/Picture credit: Original)

As seen in these figures, the velocity streamline begins by moving horizontally to the right, but after passing through the middle, a collection of little disturbances in the range of 1-2 causes it to start changing its original movement mode. The size of the β will impact the magnitude of streamline fluctuations and is the sine function's amplitude. After ensuring that = 0.05, as shown in Figs. 7-10, alter the M value from 0.8 to 1.2. The airflow effect above the wings is stimulated when the airplane accelerates. The horns of the shark steadily decrease as M rises.

4. Conclusion

The article studies fluid flow above an airfoil under transonic conditions based on CFD and uses the finite difference method to solve the SDE. In this simulation, the streamlines visually represent the flow pattern. It is absolutely a great method to analyze the flow behavior.

Generally, in the supercritical state of the airfoil, the supersonic flow on the surface of the wing will form a shock wave, and as M increases, the size and intensity of the shock wave increase sharply, and the loss increases. The separation of the surface layer caused by the shock wave's abrupt rise in pressure increases airflow turbulence, drastically increases resistance, dramatically decreases lift force, rapidly varies torque, and generates jitter. Failure to act will result in a flight crisis. The study of fluid flow above an airfoil under transonic conditions can be helpful to the development of transonic airfoils.

According to the findings, when the Mach number rises, the airfoil's shark angle decreases. In other words, the Mach number and shark angle are inversely related. This implies that greater Mach numbers can result in smoother flow patterns and better aerodynamic performance. The author of this study also uses the oversimplified premise that the fluid is an incompressible, inviscid fluid. Fluid viscosity and compressibility must be changed for actual applications because they are frequently unique. Therefore, more research is needed to fully understand the complex interactions during fluid flow across the airfoil.

References

[1] Han Xiangke and Qian Ruojun. Review of the basic theories of fluid mechanics. Spatial Structures, 2008, 14 (3): 9-12.

[2] Zhu Guolin and Xv Qingxin. Review of parallel computing techniques in computational fluid mechanics. Acta Aerodynamica Sinica, 2002, 20: 1-6.

[3] Shan Shuang, Chen Weiyu and Chen Zelong. A survey of application and development of small disturbance equation method in the aviation area. Journal of Physics: Conference Series, 2021, 2012 (1): 012025.

[4] Nieuwland G. Y. and Spee B. M. Transonic airfoils: recent developments in theory, experiment, and design. Annual Review of Fluid Mechanics, 1973, 5 (1): 119-150.

[5] Jiang Hongde and Cai Ruixian. An accurate spatial differencing scheme for a three-dimensional full potential equation. AIAA Journal, 1986, 24 (7): 1057-1058.

[6] Kwon Jae Ryong and Vepa Ranjan. Transonic small disturbance unsteady potential flow over very high aspect ratio wings. The Aeronautical Journal, 2022, 126 (1305): 1834-1852.

[7] Holst Terry L and Ballhaus William F. Fast, conservative schemes for the full potential equation applied to transonic flows. AIAA Journal, 1979, 17 (2): 145-152.

[8] Liu Jia, Wu Lina and Fang Xingang. Using Python to solve the Navier-stokes equations-applications in the preconditioned iterative methods. Journal of Scientific Research and Reports, 2015, 7 (3): 207-217.

[9] Xu Yaohua. Dictionary of Missile Technology Missile Systems. Aerospace Press, 1991.

[10] Yan Jialu. Engineering Thermodynamics (6th Edition). Beijing: Higher Education Press, 2021.