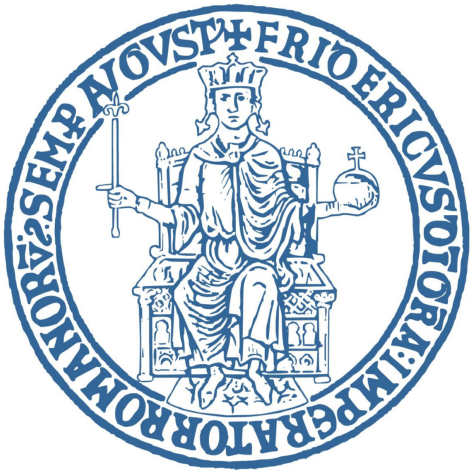

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA INDUSTRIALE

CLASSE DELLE LAUREE MAGISTRALI IN INGEGNERIA AEROSPAZIALE E ASTRONAUTICA (LM 20)

COURSE

FLUIDODINAMICA NUMERICA

Project Work

A study on two-dimensional wake flows: fluid simulation of a Kármán vortex street past a circular cylinder by computing *streaklines* and comparison with a four-digit NACA airfoil

---

SUPERVISOR:
PROF. GENNARO COPPOLA

STUDENT:
MATTEO DE LUCA PICIONE
ID: M53001994

ACADEMIC YEAR 2024 – 2025

COMPUTATIONAL FLUID DYNAMICS

A study on two-dimensional wake flows: fluid simulation of a Kármán vortex street past a circular cylinder by computing *streaklines* and comparison with a four-digit NACA airfoil

Using *CFD* techniques, the study employs the ψ-ζ model to solve the incompressible Navier-Stokes equations and track *streaklines*, allowing for qualitative analyses of wake structures. Simulations are performed on a collocated grid with appropriate boundary conditions, ensuring stability and accuracy through a finite difference approach and an explicit Runge-Kutta method. Results highlight the distinct wake behavior of each geometry, with the circular cylinder generating a well-defined vortex street, whereas the airfoil exhibits reduced wake turbulence. The findings demonstrates the impact of body shape on vortex shedding phenomena.

Contents

1 *Psi-Zita* model **1**
1.1 Vorticity . . . 2
1.2 Stream function . . . 2
1.3 Flow model . . . 3

2 Lines of flow **3**
2.1 Streamlines, pathlines and streaklines . . . 3

3 Flow simulation **4**
3.1 Domain . . . 5
3.2 Boundary conditions . . . 6
3.3 Discretization of diffusive and convective terms . . . 7
3.4 Time integration . . . 9
3.5 Implementation . . . 10

4 Results **18**

References **30**

1 *Psi-Zita* model

Incompressible Navier-Stokes equations (*INS*) are the starting point to obtain the *Psi-Zita* model (ψ-ζ), which is a flow model particularly suited for two-dimensional incompressible fluid flow simulations.

INS are the mass conservation equation and the momentum balance equation. They constitute the set of partial differential equations (1), where (i) $\rho = \rho(\boldsymbol{x}, t)$ is the density field, (ii) $\boldsymbol{u} = \boldsymbol{u}(\boldsymbol{x}, t) = [u, v, w]^{\mathrm{T}}$ is the

velocity vector field, (*iii*) $p = p(\boldsymbol{x}, t)$ is the pressure field, (*iv*) μ is the dynamic viscosity of the fluid, and (*v*) $\boldsymbol{f} = \boldsymbol{f}(\boldsymbol{x}, t)$ is the body force vector field.

$$\begin{cases} \nabla \cdot \boldsymbol{u} = 0 \\ \rho \dfrac{\partial \boldsymbol{u}}{\partial t} + \rho \boldsymbol{u} \cdot \nabla \boldsymbol{u} + \nabla p = \mu \nabla^2 \boldsymbol{u} + \rho \boldsymbol{f} \end{cases} \tag{1}$$

In most cases the body force vector field is conservative, thereby it can be rewritten in terms of the gradient of its scalar potential ξ as

$$\boldsymbol{f} = -\nabla \xi \tag{2}$$

and the *INS* become

$$\begin{cases} \nabla \cdot \boldsymbol{u} = 0 \\ \dfrac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} + \nabla P = \nu \nabla^2 \boldsymbol{u} \end{cases} \tag{3}$$

where ν is the kinematics viscosity of the fluid and $P = p + \xi/\rho$ is the generalized pressure field.

Applying the divergence operator at both sides of the momentum balance equation yields

$$\nabla^2 P = -\nabla \cdot (\boldsymbol{u} \cdot \nabla \boldsymbol{u}) \tag{4}$$

that is an elliptic equation in the generalized pressure field known as the pressure evolution equation. It shows how pressure and velocity field affect each other, instantaneously adapting at every instant. This behavior is caused by the speed of sound being infinite, as assumed in the *INS*.

A further investigation of equation (4) shows that with this model the pressure field does not depend on the fluid temperature field T, but only on the kinematics of the flow. This mathematically translates to the energy balance equation being unrelated from the set of equations (3). This allow to exclude the energy equation from the *INS*, since they already constitute a well-posed differential problem in the unknowns $\boldsymbol{u}$ and p.

Furthermore, the pressure in this model does not have a physical meaning, since it depends on the velocity field and its initial condition cannot be chosen arbitrarily. As a result, it cannot be assumed any fluid thermodynamical model, e.g. ideal gas law $p = \rho RT$ must not be considered.

1.1 Vorticity

The flow vorticity is defined as

$$\boldsymbol{\omega} = \nabla \times \boldsymbol{u} \tag{5}$$

and it allows to rewrite the convective term in the momentum balance equation as follows

$$\boldsymbol{u} \cdot \nabla \boldsymbol{u} = \boldsymbol{\omega} \times \boldsymbol{u} + \nabla \left(\frac{|\boldsymbol{u}|^2}{2} \right) \tag{6}$$

Substituting equation (6) into the momentum balance equation and applying the curl operator to both of its sides gives the following

$$\frac{\partial \boldsymbol{\omega}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{\omega} - \boldsymbol{\omega} \cdot \nabla \boldsymbol{u} = \nu \nabla^2 \boldsymbol{\omega} \tag{7}$$

known as the vorticity equation.

It is worth noticing how this equation can be can be simplified when analyzing a two-dimensional flow. In particular, in this case the vorticity is

$$\boldsymbol{\omega} = \nabla \times \boldsymbol{u} = \left(\frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \right) \boldsymbol{k} = \omega \boldsymbol{k} \tag{8}$$

with $\boldsymbol{k}$ being the z-axis versor orthogonally to the motion plane xy and ω the projection of $\boldsymbol{\omega}$ along that direction. As a result, equation (7) now becomes

$$\frac{\partial \omega}{\partial t} + \boldsymbol{u} \cdot \nabla \omega = \nu \nabla^2 \omega \tag{9}$$

1.2 Stream function

In the case of a non-compressible fluid flow, the velocity vector field is non-divergent, thereby there exists a vector field $\boldsymbol{\psi}$ called vector potential, such that

$$\boldsymbol{u} = \nabla \times \boldsymbol{\psi} \tag{10}$$

It can easily be verified that when the flow is two-dimensional, the vector potential is $\boldsymbol{\psi} = \psi \boldsymbol{k}$ and ψ is commonly known as the stream function. It follows that

$$u = \frac{\partial \psi}{\partial y} \quad \text{and} \quad v = -\frac{\partial \psi}{\partial x} \tag{11}$$

With this setting the mass conservation equation becomes trivially satisfied, as shown here

$$\nabla \cdot \boldsymbol{u} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = \frac{\partial^2 \psi}{\partial y \partial x} - \frac{\partial^2 \psi}{\partial y \partial x} = 0 \tag{12}$$

The relation between the stream function and the vorticity in two dimensions can be found by substituting equations (11) into equation (8). This yields

$$\nabla^2 \psi = -\omega \tag{13}$$

1.3 Flow model

Combining equations (9, 13) results in the *Psi-Zita* model, namely

$$\begin{cases} \dfrac{\partial \zeta}{\partial t} + \boldsymbol{u} \cdot \nabla \zeta = \nu \nabla^2 \zeta \\ \nabla^2 \psi = \zeta \end{cases} \tag{14}$$

where the new flow variable $\zeta = -\omega$ has been defined and will be used from now on.

An adimensionalization can be done on the set of equations (14) by introducing constant reference dimensional values, denoted with a subscript «ref», and dimensionless variables denoted with a tilde. In particular, these new positions are considered

$$\boldsymbol{u} = u_{\text{ref}} \tilde{\boldsymbol{u}} \tag{15}$$

$$\zeta = \frac{u_{\text{ref}}}{L_{\text{ref}}} \tilde{\zeta} \tag{16}$$

$$\psi = L_{\text{ref}} u_{\text{ref}} \tilde{\psi} \tag{17}$$

$$t = \frac{L_{\text{ref}}}{u_{\text{ref}}} \tilde{t} \tag{18}$$

$$\nabla = \frac{1}{L_{\text{ref}}} \tilde{\nabla} \tag{19}$$

$$\nabla^2 = \frac{1}{L_{\text{ref}}^2} \tilde{\nabla}^2 \tag{20}$$

where (*i*) u_{ref} is the reference speed, (*ii*) L_{ref} is the reference length, (*iii*) $\tilde{\boldsymbol{u}}$ is the adimensional velocity vector field, (*iv*) $\tilde{\zeta}$ is the opposite of the adimensional vorticity, (*v*) $\tilde{\psi}$ is the adimensional stream function, and (*vi*) $\tilde{t}$ is the adimensional time. In addition, also the dimensionless gradient and Laplacian operator have been defined, those are $\tilde{\nabla}$ and $\tilde{\nabla}^2$ respectively.

This adimensional setup allows to rewrite the ψ-ζ model as follows

$$\begin{cases} \dfrac{\partial \tilde{\zeta}}{\partial t} + \tilde{\boldsymbol{u}} \cdot \tilde{\nabla} \tilde{\zeta} = \dfrac{1}{\text{Re}} \tilde{\nabla}^2 \tilde{\zeta} \\ \tilde{\nabla}^2 \tilde{\psi} = \tilde{\zeta} \end{cases} \tag{21}$$

with Re being the Reynolds' number. It indicates the ratio between inertial and viscous forces and in this context it has been defined as

$$\text{Re} = \frac{u_{\text{ref}} L_{\text{ref}}}{\nu} \tag{22}$$

2 Lines of flow

In addition to knowing the physical parameters of a fluid flow, it can be useful to draw pictures of specific curves that give an intuitive understanding of the motion. Those curves are so called lines of flow and they can be of three different types, with their own definitions and usage cases.

A discussion of their mathematical characterization and some examples and applications follow below.

2.1 Streamlines, pathlines and streaklines

A *streamline* is a curve whose tangent at any point is in the direction of the velocity vector at that point. In particular, a *streamline* can be mathematically defined as follows [1]

$$\boldsymbol{u} \times \mathrm{d}\boldsymbol{\ell} = 0 \tag{23}$$

where $\mathrm{d}\boldsymbol{\ell}$ is the directed element of the *streamline*. This relation corresponds to the

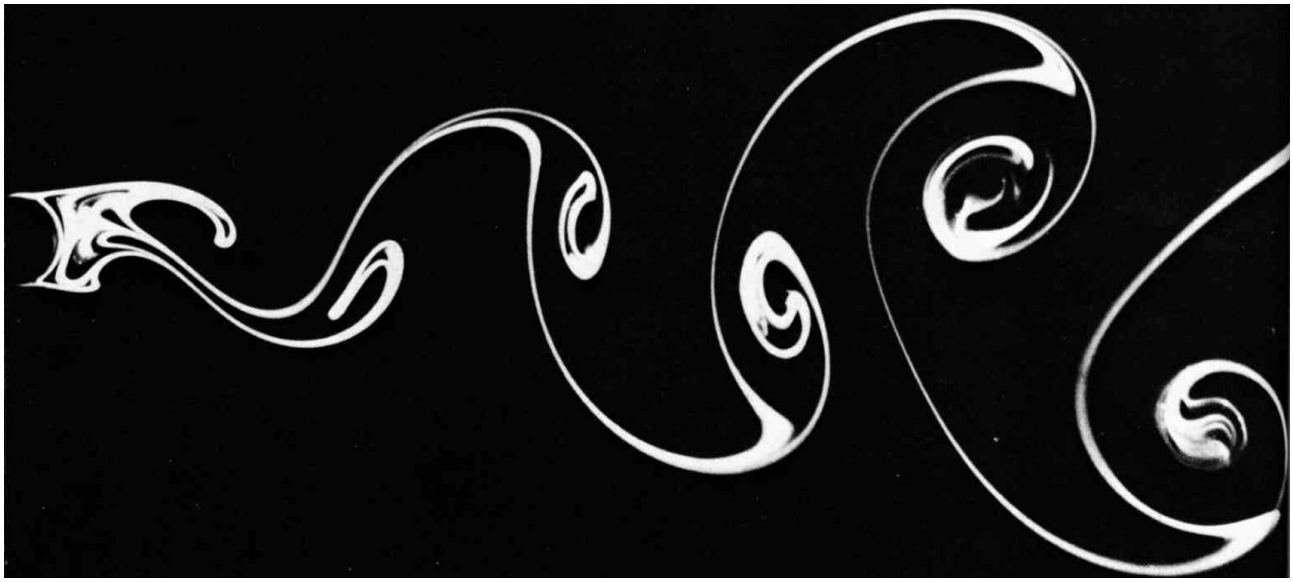

Figure 1 Kármán vortex street behind a circular cylinder [2]. *Streaklines* are the easiest lines of flow to visualize during experimental procedures. They can be obtained by adding dye at a specific position, then the fluid flows and transports the dye realizing colored and well distinguishable *streaklines*.

tangent and the velocity being parallel at every point.

A *pathline* is the trajectory that a fluid particle follows through its motion. It is defined as the curve $\boldsymbol{x}_q = \boldsymbol{x}_q(t)$ satisfying the mathematical relation

$$\frac{\mathrm{d}\boldsymbol{x}_q}{\mathrm{d}t} = \boldsymbol{u}(\boldsymbol{x}_q(t), t) \tag{24}$$

with q being an arbitrary fluid particle and $\boldsymbol{x}_q$ its position.

It can easily be noticed that the trajectory of q is determined by the Eulerian velocity vector field $\boldsymbol{u}(\boldsymbol{x}, t)$, thereby every *pathline* depends on the *streamlines*.

Pathlines and *streamlines* are different curves for unsteady flows. In the general case *streamlines* represent a snapshot of the flow at any given instant t, since the velocity vector field $\boldsymbol{u}(\boldsymbol{x}, t)$ explicitly depends on time, whereas *pathlines* give an overview on the previous history of the flow.

A *streakline* is a curve $\boldsymbol{x}_\mathrm{s} = \boldsymbol{x}_\mathrm{s}(t)$ joining the isochronous positions of all the particles that have passed through a reference point $\boldsymbol{x}_0$ at a previous instant. Those isochronous positions $\boldsymbol{x}_{q_k}(t)$ can be found by solving the following differential equations, together with its initial condition

$$\begin{cases} \dfrac{\mathrm{d}\boldsymbol{x}_{q_k}}{\mathrm{d}t} = \boldsymbol{u}(\boldsymbol{x}_{q_k}(t), t) \\ \boldsymbol{x}_{q_k}(t_0 + k\Delta t) = \boldsymbol{x}_0 \end{cases} \tag{25}$$

where q_k is the k-th tracked particle that passed through $\boldsymbol{x}_0$ at time $t_0 + k\Delta t$.

It is worth noticing that to evaluate the *streaklines*, one can before compute all the required *pathlines* $\boldsymbol{x}_{q_k} = \boldsymbol{x}_{q_k}(t)$ that have a common initial position at different initial instants.

Figure 1 shows a real visualization of *streaklines* in the wake of a cylinder, that has been photographed during an experiment.

3 Flow simulation

In this section of the study all the actual simulation concepts and data will be presented and discussed.

The simulation will integrate the set of equations ψ-ζ (21) to solve fluid flow inside a rectangular two-dimensional duct containing a circular cylinder as obstacle. Therefore,

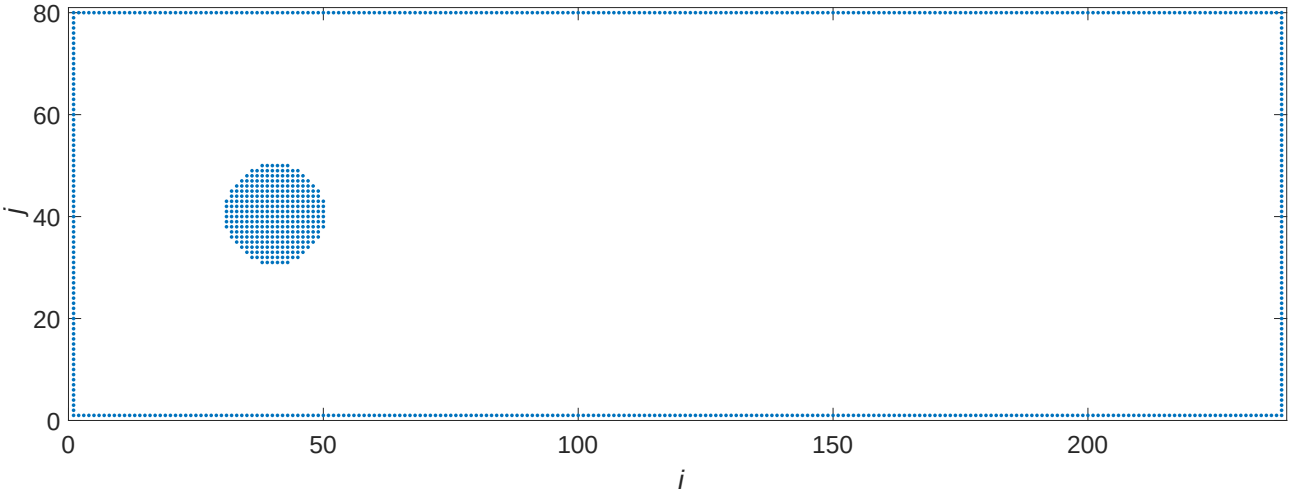

Figure 2 Discretized mesh for the flow around cylinder simulation. Blue nodes represent non-flowing zones, in this case the boundaries and the inside of the circular cylinder, and white space represents where the fluid is allowed to flow. As the diagram shows, indices i and j distinguish every node of the mesh, ordering them in equivalence with Cartesian coordinates x and y respectively.

anything will be carried out with dimensionless physical quantities, as the model requires.

For ease of notation tildes will be omitted from the adimensional variables hence quantities $\boldsymbol{x}$, $\boldsymbol{u}$, ζ, ψ, etc. must be considered as pure numbers from now on.

Here some new parameters characterizing the simulation are introduced, namely (*i*) L_x and L_y are the rectangular domain dimensions, (*ii*) h is the cell spacing of the mesh, the same for both the x and y directions, (*iii*) N_x and N_y are the number of nodes in the x and y directions respectively, (*iv*) T is the ending time of the simulation, (*v*) Δt is the timestep width, and (*vi*) N_t is the number of timesteps.

In addition, to account for stability two parameters are used. The convective stability parameter and the diffusive stability parameter, respectively the Courant's number Cou and the β number. In this context they are defined as

$$\mathrm{Cou} = \frac{u_{\mathrm{ref}}\Delta t}{h} \qquad \beta = \frac{\Delta t}{h^2 \mathrm{Re}} \tag{26}$$

Cou and β will be assigned at sufficiently low values and the temporal discretization will be found by evaluating the timestep width as

$$\Delta t = \min\left\{\mathrm{Cou}\frac{h}{u_{\mathrm{ref}}},\ \beta h^2 \mathrm{Re}\right\} \tag{27}$$

3.1 Domain

The domain is discretized with an uniform mesh of cell spacing $\Delta x = \Delta y = h$. To do so, the physical domain is defined using an aspect ratio parameter. In particular, after assigned the short side of the domain L_y, the other one can be obtained using

$$L_x = \mathcal{R} L_y \tag{28}$$

with $\mathcal{R} > 1$. This allows to define the mesh in the same way, by first assigning the number of nodes in the y direction and then compute the number of nodes in the x direction. In particular, the following relation must be used

$$(N_x - 1) = \mathcal{R}(N_y - 1) \tag{29}$$

where $(N_x - 1)$ and $(N_y - 1)$ are the number of cells in the x and y directions respectively. This effectively allows the mesh to be equally spaced.

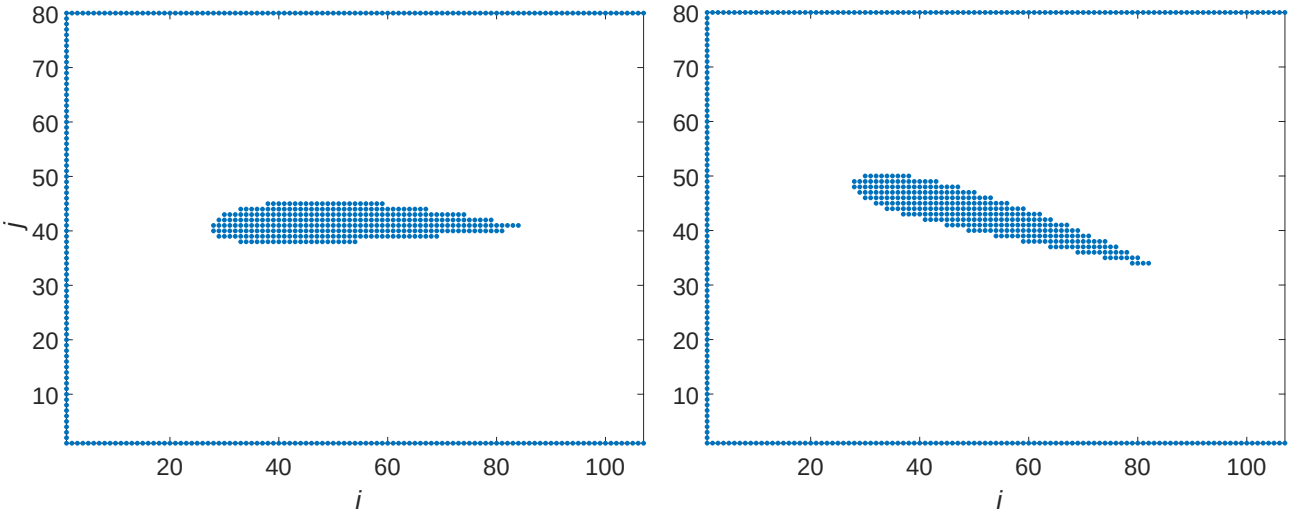

Figure 3 Discretized meshes for the flow around airfoil simulation. The shape of a 2418 NACA airfoil has been approximated at angles 0 deg and 15 deg. Same mesh conventions already introduced with figure 2 are assumed here. Increasing N_y will eventually result in more accurate shapes.

Algorithm 1 Assignment of a topological matrix $\boldsymbol{G}$ to enumerate mesh nodes in case of a cylindrical obstacle.

Data: Mesh sizes N_x and N_y
Position of every node $\boldsymbol{x}_{ij}$
Cylinder center $\boldsymbol{x}_C$
Cylinder radius R
$k = 0$;
$\boldsymbol{G} = \text{ZeroMatrix}(N_x \times N_y)$;
for $j = 2, 3, 4, ..., (N_y - 1)$ **do**
  for $i = 2, 3, 4, ..., (N_x - 1)$ **do**
    if $|\boldsymbol{x}_{ij} - \boldsymbol{x}_C|^2 \leq R^2$ **then**
      $G_{ij} = -1$;
    else
      $k = k + 1$;
      $G_{ij} = k$;
    end
  end
end

As shown in figures 2 and 3, nodes in the mesh are ordered in equivalence with Cartesian coordinates x and y. In particular, indices i and j are used to indicate a generic node n sitting at position $\boldsymbol{x}_n = (x_i, y_j)$, with the discrete coordinate values x_i and y_j being monotonically increasing.

To take into account the non-flowing zones of the domain, a topological matrix $\boldsymbol{G}$ can be defined, having N_x rows and N_y columns, such that each of its elements corresponds to a node of the mesh. The elements corresponding to boundary nodes are assigned as zero, the elements corresponding to the inside of the obstacle are assigned as -1, and the elements corresponding to flowing zone nodes are assigned as a counting index k, starting from 1 and ending at the last available node. Algorithms 1 and 2 show how such matrix can be computed.

3.2 Boundary conditions

To accurately solve for the fluid flow, it is crucial to assign appropriate boundary conditions in the variables $\boldsymbol{u}$, ψ and ζ.

At west and east walls uniform axial inflow and outflow are assigned, thereby for the velocity vector field a Dirichlet condition is used, namely

$$\boldsymbol{u}_\text{W} = \boldsymbol{u}_\text{E} = \boldsymbol{u}_\text{ref} = [u_\text{ref}, 0]^\text{T} \tag{30}$$

where u_ref is evaluated from an assigned fluid volume flow $Q = u_\text{ref}/L_y$.

To take into account the required fluid flow, the stream function must be assigned here as a linear function of y, from a value of 0 to a value of Q. In particular,

$$\psi_\text{W} = \psi_\text{E} = \frac{Q}{L_y} y \tag{31}$$

Algorithm 2 Assignment of a topological matrix $\boldsymbol{G}$ to enumerate mesh nodes in case of an airfoil obstacle [3].

Data: Mesh sizes N_x and N_y
Position of every node $\boldsymbol{x}_{ij} = (x_{ij}, y_{ij})$
Leading edge position $\boldsymbol{x}_{\text{le}} = (x_{\text{le}}, y_{\text{le}})$
Trailing edge position $\boldsymbol{x}_{\text{te}} = (x_{\text{te}}, y_{\text{te}})$
Profile equations $y_{\text{U}}(x_c)$ and $y_{\text{L}}(x_c)$
$k = 0$;
$\boldsymbol{G} = \text{ZeroMatrix}(N_x \times N_y)$;
for $j = 2, 3, 4, ..., (N_y - 1)$ **do**
  for $i = 2, 3, 4, ..., (N_x - 1)$ **do**
    if $x_{\text{le}} \leq x_{ij} \leq x_{\text{le}} + c$ **then**
      $x_{ij,c} = \dfrac{(\boldsymbol{x}_{ij} - \boldsymbol{x}_{\text{le}}) \cdot (\boldsymbol{x}_{\text{te}} - \boldsymbol{x}_{\text{le}})}{|\boldsymbol{x}_{te} - \boldsymbol{x}_{\text{le}}|^2}$;
      if $y_{\text{L}}(x_{ij,c}) \leq y_{ij} \leq y_{\text{U}}(x_{ij,c})$ **then**
        $G_{ij} = -1$;
      end
    else
      $k = k + 1$;
      $G_{ij} = k$;
    end
  end
end

South and North walls are threated as *streamlines*, thereby for the stream function a Dirichlet boundary condition is used, namely

$$\psi_{\text{S}} = 0 \quad \text{and} \quad \psi_{\text{N}} = Q \tag{32}$$

Interfaces with the obstacle are threated as streamlines, with an assigned Dirichlet boundary condition

$$\psi_{\text{O}} = \frac{Q}{2} \tag{33}$$

such that the volume flow rate on the sides is exactly $2 \cdot Q/2 = Q$, thereby the conservation of mass is satisfied.

In addition, south, north, and obstacle walls are threated with a no-slip condition for the velocity boundary condition. Therefore, given a wall w with directed element $\mathrm{d}\boldsymbol{\ell}$, the following relation is imposed

$$\boldsymbol{u}_w \cdot \mathrm{d}\boldsymbol{\ell} = 0 \tag{34}$$

To decide what values to assign to ζ at the boundaries, a Taylor series expansion of the stream function can be used, namely

$$\psi_n = \psi_w + \left.\frac{\partial \psi}{\partial y}\right|_w h + \left.\frac{\partial^2 \psi}{\partial y^2}\right|_w \frac{h^2}{2} + \mathcal{O}(h^3) \tag{35}$$

where n is a mesh node near to a wall node w. Substituting equation (11) and the second equation of (21) yields

$$\psi_n = \psi_w + u_w h + \zeta_w \frac{h^2}{2} + \mathcal{O}(h^3) \tag{36}$$

if the examined wall is threated as a streamline as in this case. This allows to obtain the Dirichlet boundary condition for ζ by rearranging

$$\zeta_w = \frac{2\,(\psi_n - \psi_w - u_w h)}{h^2} + \mathcal{O}(h) \tag{37}$$

It is worth noticing that formula (37) is suitable for a south wall, but can be easily generalized for an arbitrary oriented wall. Formulae obtained with that same approach are known as Thom's formulae.

In case of a no-slip boundary condition on the velocity, equation (37) can be further simplified using $u_w = 0$, yielding

$$\zeta_w = \frac{2\,(\psi_n - \psi_w)}{h^2} + \mathcal{O}(h) \tag{38}$$

that is the one required in this context.

3.3 Discretization of diffusive and convective terms

To solve the differential problem (21), unknowns ζ and ψ are substituted with their discrete counterparts $\boldsymbol{\zeta}$ and $\boldsymbol{\psi}$. Those are column vector containing the values ζ and ψ assume at each node of the mesh, ordered as specified by algorithm 1.

Similarly, every differential operator D acting on ζ or ψ will be substituted with its discrete form. It will act on vectors $\boldsymbol{\zeta}$ or $\boldsymbol{\psi}$, thereby it will be a matrix $\boldsymbol{D}$.

Diffusive term $\nabla^2 \zeta$ in the first equation of (21) can be discretized with a discrete Laplacian operator $\boldsymbol{L}$ such that $\nabla^2 \zeta$ will be substituted with $\boldsymbol{L\zeta}$. It can be obtained by

using finite difference central schemes for the second order derivatives. In particular,

$$\left.\frac{\partial^2\zeta}{\partial x^2}\right|_{ij} = \frac{\zeta_{i+1,j} - 2\zeta_{ij} + \zeta_{i-1,j}}{h^2} + \mathcal{O}(h^2) \quad (39)$$

$$\left.\frac{\partial^2\zeta}{\partial y^2}\right|_{ij} = \frac{\zeta_{i,j+1} - 2\zeta_{ij} + \zeta_{i,j-1}}{h^2} + \mathcal{O}(h^2) \quad (40)$$

yielding

$$\begin{aligned}\nabla^2\zeta|_{ij} = \frac{1}{h^2}\big(&\zeta_{i+1,j} + \zeta_{i-1,j} + \zeta_{i,j+1} \\ &+ \zeta_{i,j-1} - 4\zeta_{ij}\big) + \mathcal{O}(h^2)\end{aligned} \quad (41)$$

Algorithm 3 shows how such matrix $\boldsymbol{L}$ can be computed.

Algorithm 3 Example of definition of the discrete Laplacian operator $\boldsymbol{L}$.

Data: Mesh sizes N_x and N_y
Nodes spacing h
Topological matrix $\boldsymbol{G}$
Number of internal nodes N
$\boldsymbol{L} = \text{ZeroMatrix}(N \times N)$;
for $j = 2, 3, 4, ..., (N_y - 1)$ **do**
  for $i = 2, 3, 4, ..., (N_x - 1)$ **do**
    if $G_{ij} > 0$ **then**
      $k = G_{ij}$;
      $L_{kk} = -4/h^2$;
      $k_\text{E} = G_{i+1,j}$;
      $k_\text{W} = G_{i-1,j}$;
      $k_\text{N} = G_{i,j+1}$;
      $k_\text{S} = G_{i,j-1}$;
      if $k_\text{E} > 0$ **then**
        $L_{kk_\text{E}} = 1/h^2$;
      end
      if $k_\text{W} > 0$ **then**
        $L_{kk_\text{W}} = 1/h^2$;
      end
      if $k_\text{N} > 0$ **then**
        $L_{kk_\text{N}} = 1/h^2$;
      end
      if $k_\text{S} > 0$ **then**
        $L_{kk_\text{S}} = 1/h^2$;
      end
    end
  end
end

The convective term $\boldsymbol{u}\cdot\nabla\zeta = \nabla\cdot(\boldsymbol{u}\zeta)$ in the first equation of (21) can be discretized following various approaches. It can be seen in the advective form $\boldsymbol{u}\cdot\nabla\zeta$, in the divergence form $\nabla\cdot(\boldsymbol{u}\zeta)$, or in any appropriate linear combination of them.

The advective form of the convective term gives the following discretization

$$\left(\boldsymbol{U}\boldsymbol{D}_x + \boldsymbol{V}\boldsymbol{D}_y\right)\boldsymbol{\zeta} \quad (42)$$

whereas the divergence form yields

$$\left(\boldsymbol{D}_x\boldsymbol{U} + \boldsymbol{D}_y\boldsymbol{V}\right)\boldsymbol{\zeta} \quad (43)$$

where (*i*) $\boldsymbol{U} = \text{diag}\,(u_k)$, (*ii*) $\boldsymbol{V} = \text{diag}\,(v_k)$, and (*iii*) $\boldsymbol{D}_x$ and $\boldsymbol{D}_y$ are the discrete first order derivative operators in the x and y directions respectively.

An appropriate linear combination of the above two forms of the convective term is

$$\alpha\boldsymbol{u}\cdot\nabla\zeta + \beta\nabla\cdot(\boldsymbol{u}\zeta) \quad (44)$$

with $\alpha + \beta = 1$. This yields the following discretization

$$\left[\alpha\left(\boldsymbol{U}\boldsymbol{D}_x + \boldsymbol{V}\boldsymbol{D}_y\right) + \beta\left(\boldsymbol{D}_x\boldsymbol{U} + \boldsymbol{D}_y\boldsymbol{V}\right)\right]\boldsymbol{\zeta} \quad (45)$$

To decide which discretization approach to follow, it is useful to introduce the total enstrophy $\mathcal{E}$ over the domain Ω. In this context it is defined as

$$\mathcal{E} = \int_\Omega \frac{\zeta^2}{2}\,\mathrm{d}\Omega \quad (46)$$

and it must conserve, thereby the condition $\mathrm{d}\mathcal{E}/\mathrm{d}t = 0$ must be satisfied. In particular,

$$\int_\Omega \left[\frac{\partial}{\partial t}\left(\frac{\zeta^2}{2}\right) + \boldsymbol{u}\cdot\nabla\left(\frac{\zeta^2}{2}\right)\right]\mathrm{d}\Omega = 0 \quad (47)$$

having used Reynolds' transport equation.

Since the domain Ω is stationary, both of the integrals that arise in equation (47) must be zero. This gives the following relation

$$\int_\Omega \boldsymbol{u}\cdot\nabla\left(\frac{\zeta^2}{2}\right)\mathrm{d}\Omega = \int_\Omega \zeta\boldsymbol{u}\cdot\nabla\zeta\,\mathrm{d}\Omega = 0 \quad (48)$$

that when discretized is equivalent to

$$\begin{aligned}\boldsymbol{\zeta}^\mathrm{T}\big[&\alpha\left(\boldsymbol{U}\boldsymbol{D}_x + \boldsymbol{V}\boldsymbol{D}_y\right) \\ &+ \beta\left(\boldsymbol{D}_x\boldsymbol{U} + \boldsymbol{D}_y\boldsymbol{V}\right)\big]\boldsymbol{\zeta} = 0\end{aligned} \quad (49)$$

assuming the most general form of the convective term.

The imposed condition (49) is equivalent to require a quadratic form to always be zero, thereby it is equivalent to require the matrix of the form to be anti-symmetrical. This can be achieved only when

$$\alpha = \beta = 1/2 \tag{50}$$

that gives

$$\frac{1}{2}\left(\boldsymbol{U}\boldsymbol{D}_x + \boldsymbol{V}\boldsymbol{D}_y + \boldsymbol{D}_x\boldsymbol{U} + \boldsymbol{D}_y\boldsymbol{V}\right)\boldsymbol{\zeta} \tag{51}$$

as the most appropriate discretization of the convective term to satisfy the conservation of enstrophy.

Discretization (51) is known as the skew-symmetric form of the discretized convective term and from now on its matrix operator will be denoted as

$$\boldsymbol{C} = \frac{1}{2}\left(\boldsymbol{U}\boldsymbol{D}_x + \boldsymbol{V}\boldsymbol{D}_y + \boldsymbol{D}_x\boldsymbol{U} + \boldsymbol{D}_y\boldsymbol{V}\right) \tag{52}$$

3.4 Time integration

Using the spatial discretizations (41, 51) of the diffusive and convective terms, the set of partial differential equations (21) becomes the following set of constrained ordinary differential equations

$$\begin{cases} \dfrac{\mathrm{d}\boldsymbol{\zeta}}{\mathrm{d}t} = -\boldsymbol{C}\boldsymbol{\zeta} + \dfrac{1}{\mathrm{Re}}\boldsymbol{L}\boldsymbol{\zeta} \\ \boldsymbol{L}\boldsymbol{\psi} = \boldsymbol{\zeta} \end{cases} \tag{53}$$

that can be integrated using a numerical procedure, e.g. using a multi-step method or a single-step multi-stage method.

In this context a 4th order explicit Runge-Kutta method [6] is used to integrate the differential problem, given by the following Butcher tableau

$$\begin{array}{c|cccc} 0 & 0 & & & \\ 1/2 & 1/2 & 0 & & \\ 1/2 & 0 & 1/2 & 0 & \\ 1 & 0 & 0 & 1 & 0 \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array} \tag{54}$$

Algorithm 4 A Runge-Kutta iteration to solve for $\boldsymbol{\zeta}$ and $\boldsymbol{\psi}$ through integration over a timestep.

Data: Timestep width Δt
Discrete Laplacian operator $\boldsymbol{L}$
Convective discretization operator $\boldsymbol{C}$
Previous timestep value $\boldsymbol{\zeta}_{\text{old}}$
Boundary conditions for $\boldsymbol{\zeta}$

$\boldsymbol{\zeta}_1 = \boldsymbol{\zeta}_{\text{old}}$;
$\boldsymbol{f}_1 = -\boldsymbol{C}\boldsymbol{\zeta}_1 + \dfrac{1}{\mathrm{Re}}\boldsymbol{L}\boldsymbol{\zeta}_1$;
$\boldsymbol{\zeta}_2 = \boldsymbol{\zeta}_{\text{old}} + \dfrac{\Delta t}{2}\boldsymbol{f}_1$;
$\boldsymbol{f}_2 = -\boldsymbol{C}\boldsymbol{\zeta}_2 + \dfrac{1}{\mathrm{Re}}\boldsymbol{L}\boldsymbol{\zeta}_2$;
$\boldsymbol{\zeta}_3 = \boldsymbol{\zeta}_{\text{old}} + \dfrac{\Delta t}{2}\boldsymbol{f}_2$;
$\boldsymbol{f}_3 = -\boldsymbol{C}\boldsymbol{\zeta}_3 + \dfrac{1}{\mathrm{Re}}\boldsymbol{L}\boldsymbol{\zeta}_3$;
$\boldsymbol{\zeta}_4 = \boldsymbol{\zeta}_{\text{old}} + \Delta t\boldsymbol{f}_3$;
$\boldsymbol{f}_4 = -\boldsymbol{C}\boldsymbol{\zeta}_4 + \dfrac{1}{\mathrm{Re}}\boldsymbol{L}\boldsymbol{\zeta}_4$;
$\boldsymbol{\zeta}_{\text{new}} = \boldsymbol{\zeta}_{\text{old}} + \Delta t\left[\dfrac{1}{6}\left(\boldsymbol{f}_1 + \boldsymbol{f}_4\right) + \dfrac{1}{3}\left(\boldsymbol{f}_2 + \boldsymbol{f}_3\right)\right]$;
Adjust $\boldsymbol{\zeta}_{\text{new}}$ for boundary conditions;
$\boldsymbol{\psi}_{\text{new}} = \boldsymbol{L}^{-1}\boldsymbol{\zeta}_{\text{new}}$;

At each timestep the first set of equations in (53) is integrated through four stages using the Runge-Kutta method provided by the Butcher tableau (54). Then, the second set of linear equations can be solved as follows

$$\boldsymbol{\psi} = \boldsymbol{L}^{-1}\boldsymbol{\zeta} \tag{55}$$

and the procedure can be repeated after adjusting for the required boundary conditions.

The algorithm that can carry out such computation for a single iteration is shown here

In order to compute the *streaklines*, after solving for $\boldsymbol{\zeta}$ and $\boldsymbol{\psi}$ also sets of equations (25) must be integrated at each timestep, with k ranging from 0 to the timestep index i_t. To do so the velocity vector field must be evaluated from the stream function discretizing equations (11).

Algorithm 5 A Runge-Kutta iteration to solve for $\boldsymbol{x}_{q_k}$ through integration over a timestep.

Data: Timestep width Δt
Previous timestep value $\boldsymbol{x}^k_{\text{old}}$
u instantaneous node values $\boldsymbol{U}$
v instantaneous node values $\boldsymbol{V}$
An interpolation procedure

$\boldsymbol{x}^k_1 = \boldsymbol{x}^k_{\text{old}}$;
$\boldsymbol{f}_1 = \text{Interpolate}(\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{x}^k_1)$;
$\boldsymbol{x}^k_2 = \boldsymbol{x}^k_{\text{old}} + \frac{\Delta t}{2}\boldsymbol{f}_2$;
$\boldsymbol{f}_2 = \text{Interpolate}(\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{x}^k_2)$;
$\boldsymbol{x}^k_3 = \boldsymbol{x}^k_{\text{old}} + \frac{\Delta t}{2}\boldsymbol{x}^k_2$;
$\boldsymbol{f}_3 = \text{Interpolate}(\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{x}^k_3)$;
$\boldsymbol{x}^k_4 = \boldsymbol{x}^k_{\text{old}} + \Delta t \boldsymbol{f}_3$;
$\boldsymbol{f}_4 = \text{Interpolate}(\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{x}^k_4)$;
$\boldsymbol{x}^k_{\text{new}} = \boldsymbol{x}^k_{\text{old}} + \Delta t \left[\frac{1}{6}(\boldsymbol{f}_1 + \boldsymbol{f}_4) + \frac{1}{3}(\boldsymbol{f}_2 + \boldsymbol{f}_3)\right]$;

This can be achieved using finite difference central schemes for the first order derivative, namely

$$u_{ij} = \frac{\psi_{i,j+1} - \psi_{i,j-1}}{2h} + \mathcal{O}(h^2) \tag{56}$$

$$v_{ij} = -\frac{\psi_{i+1,j} - \psi_{i-1,j}}{2h} + \mathcal{O}(h^2) \tag{57}$$

Since the right-hand side of the k-th set of equations (25) for the k-th *pathline* is $\boldsymbol{u}(\boldsymbol{x}_{q_k}, t)$, the velocity vector field must be interpolated from its node values to the required particle position $\boldsymbol{x}_{q_k}$. This can be achieved by a trivial bi-linear interpolation, as shown below

$$u_{q_k} = \begin{bmatrix} 1-\sigma_x & \sigma_x \end{bmatrix} \begin{bmatrix} u_{\text{SW}} & u_{\text{NW}} \\ u_{\text{SE}} & u_{\text{NE}} \end{bmatrix} \begin{bmatrix} 1-\sigma_y \\ \sigma_y \end{bmatrix} \tag{58}$$

$$v_{q_k} = \begin{bmatrix} 1-\sigma_x & \sigma_x \end{bmatrix} \begin{bmatrix} v_{\text{SW}} & v_{\text{NW}} \\ v_{\text{SE}} & v_{\text{NE}} \end{bmatrix} \begin{bmatrix} 1-\sigma_y \\ \sigma_y \end{bmatrix} \tag{59}$$

where

$$\sigma_x = \frac{x_{q_k} - x_i}{h} \qquad \sigma_y = \frac{y_{q_k} - y_j}{h} \tag{60}$$

and subscripts «SW», «NW», «SE», and «NE» refer to the values that u and v assume at the nodes of the cell containing particle q_k.

At each timestep the particle position $\boldsymbol{x}_{q_k}$ can be integrated with the same Runge-Kutta method (54) already used. An example of an algorithm that interpolates and integrates over a single timestep to solve for the k-th particle position is algorithm 5.

From the results obtained at each timestep and for every *pathline* $\boldsymbol{x}^k$ with algorithm 5, it is possible to draw the relative *streakline* by joining all most recent positions.

3.5 Implementation

Three simulations will be carried out: one with the obstacle being a circular cylindrical, one with it being a 2418 NACA airfoil at angle $\alpha = 0$ deg, and one near the stall, at angle $\alpha = 15$ deg.

To achieve those MATLAB has being used, with two main scripts, suitable for each case, and some useful routines to ease the management of the code.

As shown below, a list of the most important MATLAB files is included here. Some graphics routine has been omitted, since they are not much of interest. Also the main script used for the second and third simulations are not shown, since it can be obtained from the first by only changing the computation of the topological matrix G. In particular, an approach based on algorithm 2 should be followed, code 8 achieves this.

In addition, during the first simulation the history of the total enstrophy previously defined in (46) is computed at each timestep, to later plot it and see if its conservation is satisfied. It is numerically computed with the following integral approximation

$$\mathcal{E}(t_{i_t}) \approx h^2 \sum_{i=1}^{N_x} \sum_{j=1}^{N_y} \zeta_{ij}(t_{i_t}), \qquad i_t = 1, 2, ..., N_t \tag{61}$$

The geometrical data chosen for these simulations domain are: (*i*) $L_y = 1$, (*ii*) $N_y = 80$, and (*iii*) $\mathcal{AR} = 3$. Time and stability chosen parameters are mentioned and shown in code 1.

Code 1 `FlowAroundCylinder.m`. Main script that simulates the flow around a cylinder of assigned size and position. $\mathrm{Re} = 500$, $\mathrm{Cou} = 0.3$, $\beta = 0.5$, and $T = 30$.

```
close all; clear; clc

% Domain geometry
Ly = 1;
AR = 3;
Lx = AR * Ly;
Nhy = 79;
Nhx = AR * Nhy;
Ny = Nhy + 1;
Nx = Nhx + 1;
y = linspace(0, Ly, Ny);
x = linspace(0, Lx, Nx);
h = x(2) - x(1);
hq = h * h;
[X, Y] = meshgrid(x, y);

% Boundary conditions
Q = 1;
PsiN = Q * ones(1, Nx)';
PsiS = zeros(1, Nx)';
PsiW = linspace(0, Q, Ny);
PsiE = PsiW;
PsiCyl = Q / 2;

% Physical parameters
T = 30;
Re = 500;
Cou = 0.3;
beta = 0.5;
uRef = Q / Ly;
Dt = min(Cou * h / uRef, beta * Re * hq);
Nt = ceil(T / Dt) + 1;
t = linspace(0, T, Nt);
Dt = t(2) - t(1);

% Obstacle geometry
Center = [Lx/6, Ly/2];
Radius = Ly / 8;

% Domain topology and discrete Laplacian operator
k = 0;
G = zeros(Nx, Ny);
for j = 2 : Ny-1
    for i = 2 : Nx-1
        if ([x(i), y(j)] - Center) * ([x(i), y(j)] - Center)' <= Radius^2
            G(i, j) = 0;
        else
            k = k + 1;
            G(i, j) = k;
```

```
        end
    end
end
Lap = -delsq(G) ./ hq;
for i = 2 : Nx-1
    for j = 2 : Ny-1
        if G(i,j) == 0
            G(i,j) = -1;
        end
    end
end

% Domain visualization
figurePosition = [0, 50, 1640, 1640/AR];
f = figure(Position=figurePosition, Units="pixels");
D = zeros(Nx, Ny);
D(G == -1) = 1;
D(G == 0) = 1;
D = D';
spy(D)
xlabel('\it i')
ylabel('\it j')
set(gca, "YDir", "normal", "FontSize", 18)
exportgraphics(f, 'Domain.pdf', Resolution=300)

% Array initialization
zita = zeros(k, 1);
PSI = zeros(Nx, Ny);
ZITA = PSI;
U = PSI;
V = PSI;

% Boundary conditions into the arrays
U(1, :) = uRef;
U(end, :) = uRef;
PSI(1, :) = PsiW;
PSI(end, :) = PsiE;
PSI(:, 1) = PsiS;
PSI(:, end) = PsiN;
PSI(G == -1) = PsiCyl;

% Pathlines initial condition (point of injection)
xP{1} = [0, Ly/2+0.100];
xP{2} = [0, Ly/2+0.125];
xP{3} = [0, Ly/2+0.150];
xP{4} = [0, Ly/2-0.100];
xP{5} = [0, Ly/2-0.125];
xP{6} = [0, Ly/2-0.150];

% Streaklines array initialization
xS{1} = zeros(2, Nt);
xS{2} = zeros(2, Nt);
xS{3} = zeros(2, Nt);
xS{4} = zeros(2, Nt);
xS{5} = zeros(2, Nt);
xS{6} = zeros(2, Nt);
```

```
for it = 1 : Nt
    xS{1}(:, it) = xP{1};
    xS{2}(:, it) = xP{2};
    xS{3}(:, it) = xP{3};
    xS{4}(:, it) = xP{4};
    xS{5}(:, it) = xP{5};
    xS{6}(:, it) = xP{6};
end

% Graphics preparation
f = figure(Position=figurePosition, Units="pixels");
Uquiv = zeros(Nx, Ny);
Vquiv = zeros(Nx, Ny);
iq = 1 : 3 : Nx;
jq = [1 : 6 : floor(Ny/3), ...
      floor(Ny/3+3) : 3 : floor(2*Ny/3), floor(2*Ny/3+6) : 6 : Ny];
Map = [linspace(75/255, 1, 500)', linspace(90/255, 1, 500)', ...
linspace(241/255, 1, 500)'; linspace(1, 180/255, 500)', ...
linspace(1, 10/255, 500)', linspace(1, 32/255, 500)'];
DeltaPictures = round(Nt / 20, 1, "significant");

% Evolution and graphics
for it = 1 : Nt
    time = t(it);
    ZITA(G == -1) = 0;
    ZITA = ThomFormulae(ZITA, PSI, G, hq);
    ZITA = RK4(time, ZITA, ...
               @(t_, y_) ZitaRHS(t_, y_, Re, h, hq, U, V, G, Nx, Ny), ...
               Dt);
    [PSI, zita] = zita2psi(zita, ZITA, PSI, Lap, G, Nx, Ny, hq, ...
                           PsiS, PsiN, PsiW, PsiE, PsiCyl);
    [U, V] = psi2uv(PSI, U, V, G, h, Nx, Ny, uRef);
    for k = 1 : it
        xS{1}(:, k) = ...
            RK4(time, xS{1}(:, k), @(t_, y_) ...
            PathlinesRHS(t_, y_, x, y, U, V, h, h, Nx, Ny), ...
            Dt);
        xS{2}(:, k) = ...
            RK4(time, xS{2}(:, k), @(t_, y_) ...
            PathlinesRHS(t_, y_, x, y, U, V, h, h, Nx, Ny), ...
            Dt);
        xS{3}(:, k) = ...
            RK4(time, xS{3}(:, k), @(t_, y_) ...
            PathlinesRHS(t_, y_, x, y, U, V, h, h, Nx, Ny), ...
            Dt);
        xS{4}(:, k) = ...
            RK4(time, xS{4}(:, k), @(t_, y_) ...
            PathlinesRHS(t_, y_, x, y, U, V, h, h, Nx, Ny), ...
            Dt);
        xS{5}(:, k) = ...
            RK4(time, xS{5}(:, k), @(t_, y_) ...
            PathlinesRHS(t_, y_, x, y, U, V, h, h, Nx, Ny), ...
            Dt);
        xS{6}(:, k) = ...
            RK4(time, xS{6}(:, k), @(t_, y_) ...
            PathlinesRHS(t_, y_, x, y, U, V, h, h, Nx, Ny), ...
```

```
            Dt);
    end
    indices = linspace(1, it, 20000);
    if mod(it, 10) == 0
        ZITA(G == -1) = NaN;
        pcolor(x, y, ZITA');
        colormap(Map)
        shading interp
        colorbar
        clim([0.7*min(min(ZITA(G > 0))), 0.7*max(max(ZITA(G > 0)))])
        hold on
        Uquiv(iq, jq) = U(iq, jq);
        Vquiv(iq, jq) = V(iq, jq);
        quiver(X, Y, Uquiv', Vquiv', 3, Color='k')
        rectangle( ...
          Position=[Center-Radius*2.145/2, 2.145*Radius, 2.145*Radius], ...
          Curvature=[1, 1], ...
          LineWidth=2.5, FaceColor='k')
        xSplot{1} = interp1(1:it, xS{1}(1, 1:it), indices, "pchip");
        ySplot{1} = interp1(1:it, xS{1}(2, 1:it), indices, "pchip");
        xSplot{2} = interp1(1:it, xS{2}(1, 1:it), indices, "pchip");
        ySplot{2} = interp1(1:it, xS{2}(2, 1:it), indices, "pchip");
        xSplot{3} = interp1(1:it, xS{3}(1, 1:it), indices, "pchip");
        ySplot{3} = interp1(1:it, xS{3}(2, 1:it), indices, "pchip");
        xSplot{4} = interp1(1:it, xS{4}(1, 1:it), indices, "pchip");
        ySplot{4} = interp1(1:it, xS{4}(2, 1:it), indices, "pchip");
        xSplot{5} = interp1(1:it, xS{5}(1, 1:it), indices, "pchip");
        ySplot{5} = interp1(1:it, xS{5}(2, 1:it), indices, "pchip");
        xSplot{6} = interp1(1:it, xS{6}(1, 1:it), indices, "pchip");
        ySplot{6} = interp1(1:it, xS{6}(2, 1:it), indices, "pchip");
        plot(xSplot{1}, smooth(ySplot{1}), LineWidth=1.5, Color='r')
        plot(xSplot{2}, smooth(ySplot{2}), LineWidth=1.5, Color='r')
        plot(xSplot{3}, smooth(ySplot{3}), LineWidth=1.5, Color='r')
        plot(xSplot{4}, smooth(ySplot{4}), LineWidth=1.5, Color='b')
        plot(xSplot{5}, smooth(ySplot{5}), LineWidth=1.5, Color='b')
        plot(xSplot{6}, smooth(ySplot{6}), LineWidth=1.5, Color='b')
        title([ ...
            '{\it t} = ', num2str(time, 2), '. {\it i_t} = ', ...
            num2str(it), ' of ', num2str(Nt) ...
            ], FontWeight="normal")
        axis image
        axis([0, Lx, 0, Ly])
        xlabel('{\it x}')
        ylabel('{\it y}')
        set(gca, "FontSize", 18)
        drawnow limitrate
        hold off
    end
end
```

Code 2 `ThomFormulae.m`. Routine that computes boundary conditions for ζ using Thom's formulae.

```
function ZITA = ThomFormulae(ZITA, PSI, G, hq)

[Nx, Ny] = size(ZITA);

i = 2 : Nx - 1;
j = 1;
ZITA(i, j) = 2 * (PSI(i, j+1) - PSI(i, j)) / hq;

i = 2 : Nx - 1;
j = Ny;
ZITA(i, j) = 2 * (PSI(i, j-1) - PSI(i, j)) / hq;

for i = 2 : Nx - 1
    for j = 2 : Ny - 1
        if G(i, j) == -1 && G(i+1, j) > 0
            ZITA(i, j) = 2 * (PSI(i+1, j) - PSI(i, j)) / hq;
        elseif G(i, j) == -1 && G(i, j-1) > 0
            ZITA(i, j) = 2 * (PSI(i, j-1) - PSI(i, j)) / hq;
        elseif G(i, j) == -1 && G(i, j+1) > 0
            ZITA(i, j) = 2 * (PSI(i, j+1) - PSI(i, j)) / hq;
        end
    end
end

end
```

Code 3 `RK4.m`. Routine that integrates using the previously discussed Runge-Kutta method.

```
function yNew = RK4(t, yOld, RHSfunction, Dt)

t1 = t;
y1 = yOld;
f1 = RHSfunction(t1, y1);

t2 = t + Dt / 2;
y2 = yOld + Dt * 0.5 * f1;
f2 = RHSfunction(t2, y2);

t3 = t + Dt / 2;
y3 = yOld + Dt * 0.5 * f2;
f3 = RHSfunction(t3, y3);

t4 = t + Dt;
y4 = yOld + Dt * f3;
f4 = RHSfunction(t4, y4);

yNew = yOld + Dt * ((f1 + f4) ./ 6 + (f2 + f3) ./ 3);

end
```

Code 4 `ZitaRHS.m`. Function of the right-hand side of the ζ partial differential equation.

```
function dZdt = ZitaRHS(~, ZITA, Re, h, hq, U, V, G, Nx, Ny)

dZdt = zeros(Nx, Ny);

i = 2 : Nx-1;
j = 2 : Ny-1;
dZdt(i, j) = -0.5 * ( ...
                U(i+1, j) .* ZITA(i+1, j) - U(i-1, j) .* ZITA(i-1, j) + ...
                V(i, j+1) .* ZITA(i, j+1) - V(i, j-1) .* ZITA(i, j-1) + ...
                U(i,j) .* (ZITA(i+1, j) - ZITA(i-1, j)) + ...
                V(i, j) .* (ZITA(i, j+1) - ZITA(i, j-1)) ...
                ) / (2 * h) + ( ...
                ZITA(i+1, j) + ZITA(i-1, j) + ...
                    ZITA(i, j+1) + ZITA(i, j-1) - 4 * ZITA(i, j) ...
                ) / (Re * hq);

dZdt(G < 0) = 0;

end
```

Code 5 `PathlinesRHS.m`. Function of the right-hand side of a *pathline* partial differential equation.

```
function dxdt = PathlinesRHS(~, xP, x, y, U, V, Dx, Dy, Nx, Ny)

i = floor(xP(1) / Dx) + 1;
j = floor(xP(2) / Dy) + 1;

xNode = [x(min(Nx, i)); y(j)];

csix = (xP(1) - xNode(1)) / Dx;
csiy = (xP(2) - xNode(2)) / Dy;

if i+1 <= Nx && j+1 <= Ny && i > 0 && j > 0
    u = [1-csix, csix] * U([i, i+1], [j, j+1]) * [1-csiy; csiy];
    v = [1-csix, csix] * V([i, i+1], [j, j+1]) * [1-csiy; csiy];
else
    u = 0.05;
    v = 0;
end

dxdt = [u; v];

end
```

Code 6 `zita2psi.m`. Routine that computes the stream function ψ from ζ.

```
function [PSI, zita] = zita2psi(zita, ZITA, PSI, Lap, G, Nx, Ny, hq, ...
                          PsiS, PsiN, PsiW, PsiE, PsiCyl)

zita(G(G > 0)) = ZITA(G > 0);

zita(G(2:end-1, 2)) = zita(G(2:end-1, 2)) - PsiS(2:end-1) / hq;
zita(G(2:end-1, end-1)) = zita(G(2:end-1, end-1)) - PsiN(2:end-1) / hq;
zita(G(2, 2:end-1)) = zita(G(2, 2:end-1)) - PsiW(2:end-1)' / hq;
```

```
zita(G(end-1, 2:end-1)) = zita(G(end-1, 2:end-1)) - PsiE(2:end-1)' / hq;

i = 2 : Nx-1;
j = 2 : Ny-1;
D = G(i, j);
zita(D(and(G(i, j) > 0, G(i, j+1) == -1))) = ...
    zita(D(and(G(i, j) > 0, G(i, j+1) == -1))) ...
    - PsiCyl / hq;
zita(D(and(G(i, j) > 0, G(i, j-1) == -1))) = ...
    zita(D(and(G(i, j) > 0, G(i, j-1) == -1))) ...
    - PsiCyl / hq;
zita(D(and(G(i, j) > 0, G(i+1, j) == -1))) = ...
    zita(D(and(G(i, j) > 0, G(i+1, j) == -1))) ...
    - PsiCyl / hq;
zita(D(and(G(i, j) > 0, G(i-1, j) == -1))) = ...
    zita(D(and(G(i, j) > 0, G(i-1, j) == -1))) ...
    - PsiCyl / hq;

psi = Lap \ zita;
PSI(G > 0) = psi;

end
```

Code 7 `psi2uv.m`. Routine that computes velocity components u and v from the stream function ψ.

```
function [U, V] = psi2uv(PSI, U, V, G, h, Nx, Ny, uRef)

i = 2 : Nx-1;
j = 2 : Ny-1;
U(i, j) =  (PSI(i, j+1) - PSI(i, j-1)) / (2 * h);
V(i, j) = -(PSI(i+1, j) - PSI(i-1, j)) / (2 * h);

U(G < 0) = 0;
V(G < 0) = 0;
U(1, :) = uRef;
U(end, :) = uRef;

end
```

Code 8 `isInsideNACA2418.m`. Routine that determines whether or not a given position $\boldsymbol{x} = (x, y)$ lies inside a NACA2418 airfoil.

```
function isInside = isInsideNACA2418(x, y, c, le, alpha_deg)

isInside = false;

m = 0.02;
p = 0.40;
t = 0.18 * c;

theta = @(xc) ...
    atan(2*m / p^2 * (p - xc) * (xc <= p && xc >= 0) + ...
    2*m / (1 - p^2) * (p - xc) * (xc > p && xc <= 1));
yt = @(xc) ...
    5 * t * (0.2969 * sqrt(xc) - 0.1260 * xc - 0.3516 * xc.^2 + ...
```

```
    0.2843 * xc.^3 - 0.1015 * xc.^4);
yc = @(xc) ...
    m / p^2 * (2 * p * xc - xc.^2) * (xc >= 0 && xc <= p) + ...
        m / (1 - p)^2 * (1 - 2*p + 2*p*xc - xc.^2) * (xc > p && xc <= 1);

xU = @(xc) c * (xc - yt(xc) * sin(theta(xc)));
xL= @(xc) c * (xc + yt(xc) * sin(theta(xc)));

yU = @(xc) c * (yc(xc) + yt(xc) .* cos(theta(xc)));
yL = @(xc) c * (yc(xc) - yt(xc) .* cos(theta(xc)));

Upper = @(xc) Rotate([xU(xc), yU(xc)], alpha_deg) + le(:);
Lower = @(xc) Rotate([xL(xc), yL(xc)], alpha_deg) + le(:);

yUpper = @(xc) [0, 1] * Upper(xc);
yLower = @(xc) [0, 1] * Lower(xc);

xte = le(1) + c * cosd(alpha_deg);

if x >= le(1) && x <= xte
    if y >= yLower((x - le(1)) / (c * cosd(alpha_deg))) && y <= yUpper((x -
       le(1)) / (c * cosd(alpha_deg)))
        isInside = true;
    end
end

end
```

4 Results

Results of each simulation are shown here. It is worth pointing out that black regions in the plot are only graphical representations of the obstacles and that the actual computed obstacles are the ones previously shown in figures 2 and 3.

Figure 4 shows the time history of the total enstrophy as previously defined in equation (46).

Wake flow past the circular cylinder is shown in figures 5 to 12, while wake flow past the airfoil is shown in figures 13 to 26, for two different angles of attack. Resulted diagrams have been captured at different time instants.

It is worth noticing the qualitative differences between the two geometries. As expected, the wake flow past the airfoil is significantly less intense that the one past the circular cylinder, even when the airfoil is near stall. In fact, the wake flow past the airfoil quickly mitigates during the beginning of the simulation.

On the other hand, the cylinder simulation allows to easily appreciate a Kármán vortex street and a significant wake turbulence.

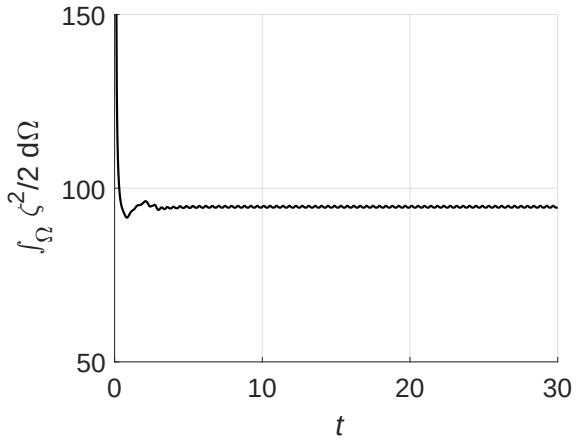

Figure 4 Time history of the total enstrophy. Other than a significant overshoot error during the beginning of the simulation, the total enstrophy is approximately constant as required and expected through the entire simulation.

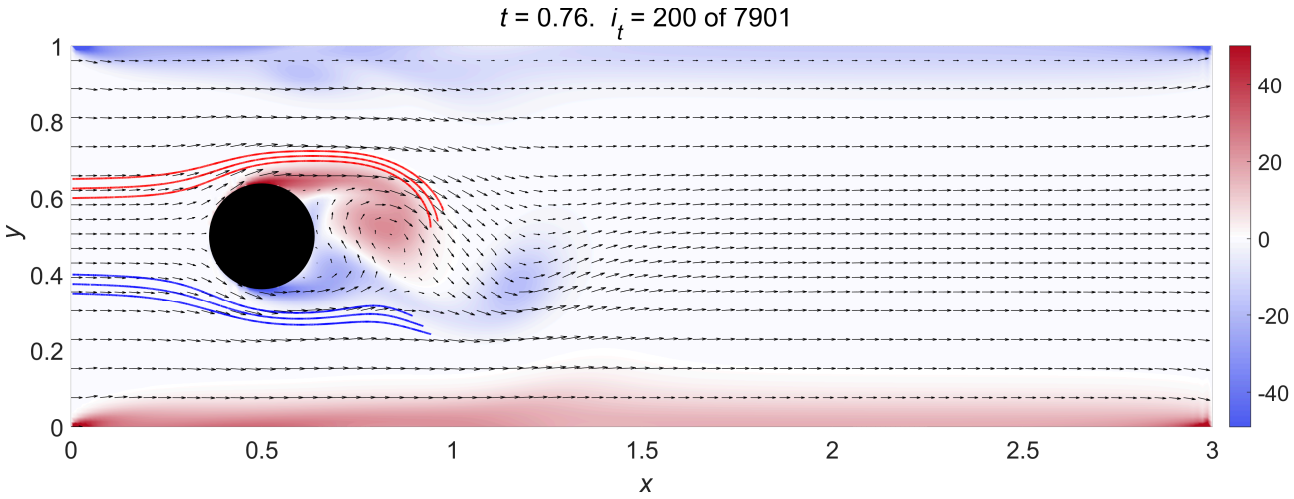

Figure 5 Wake flow past a circular cylinder at $t = 0.76$. Heavy red and blue lines are *streaklines* and the color plot refers to ζ values over the domain.

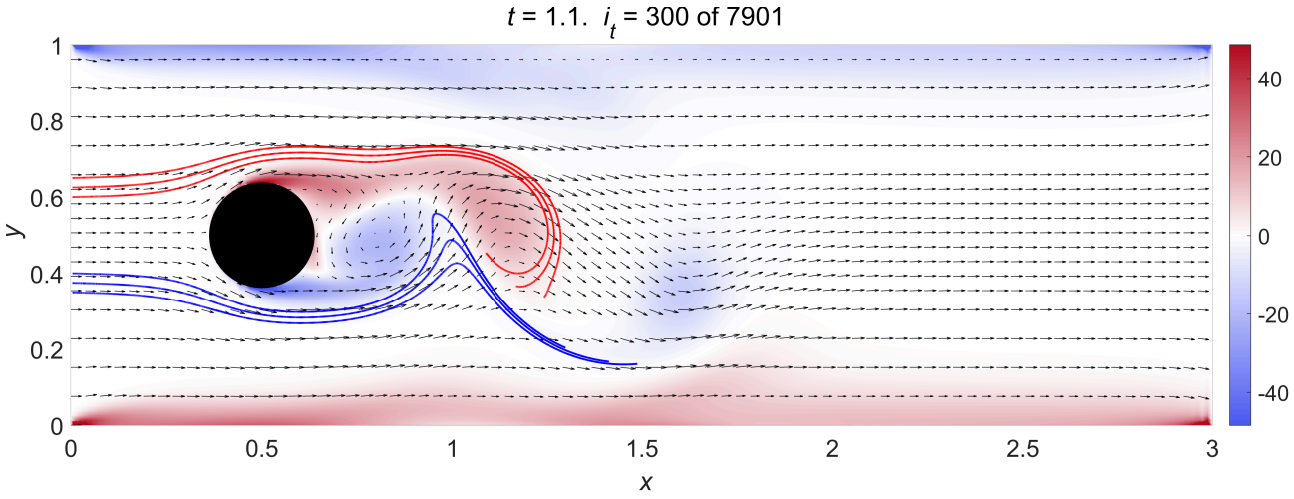

Figure 6 Wake flow past a circular cylinder at $t = 1.1$. Heavy red and blue lines are *streaklines* and the color plot refers to ζ values over the domain.

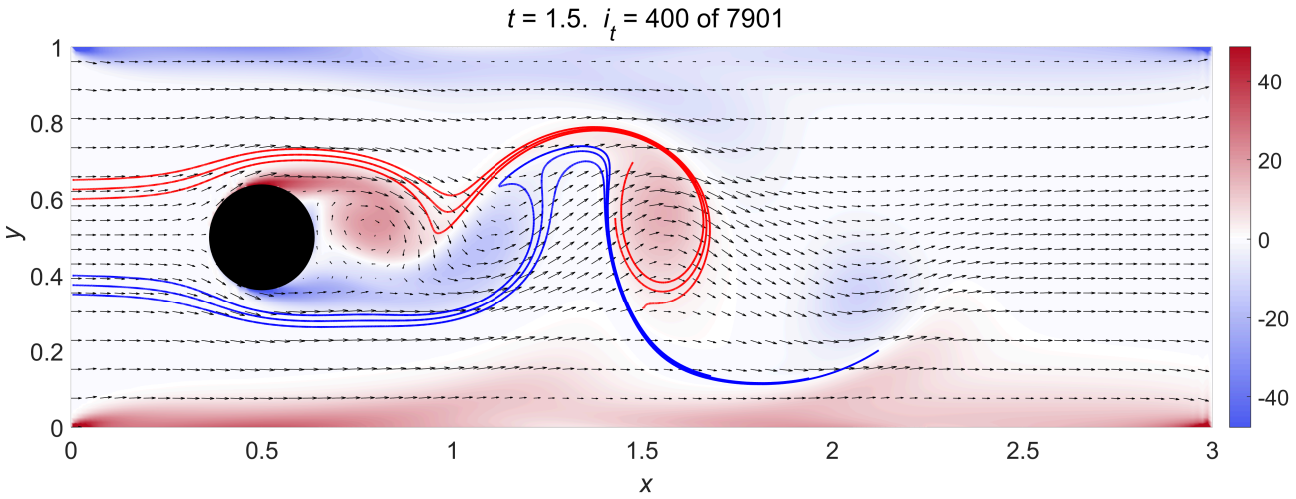

Figure 7 Wake flow past a circular cylinder at $t = 1.5$. Heavy red and blue lines are *streaklines* and the color plot refers to ζ values over the domain.

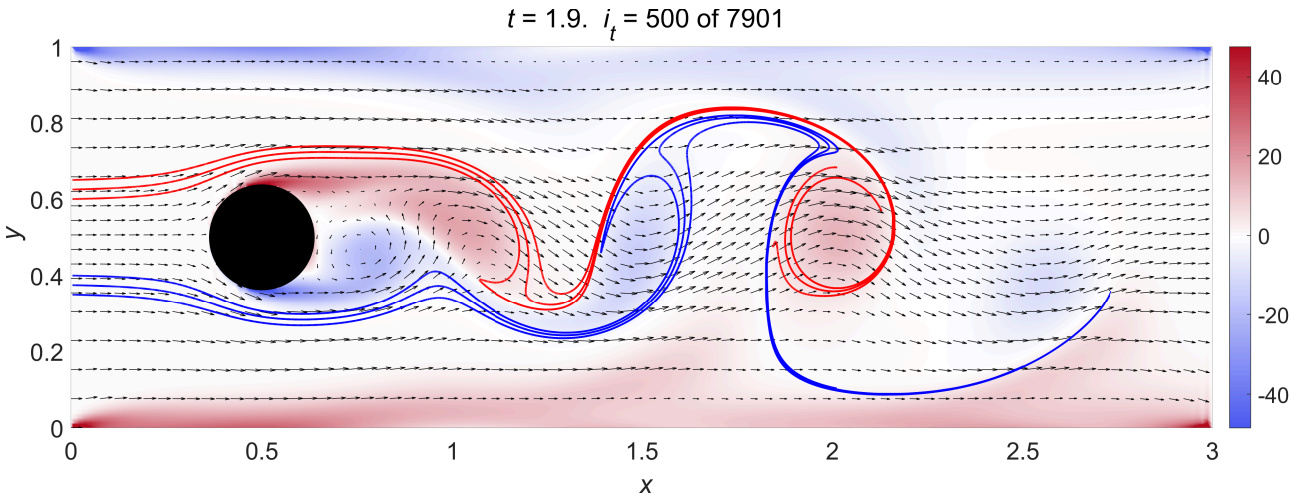

Figure 8 Wake flow past a circular cylinder at $t = 1.9$. Heavy red and blue lines are *streaklines* and the color plot refers to ζ values over the domain.

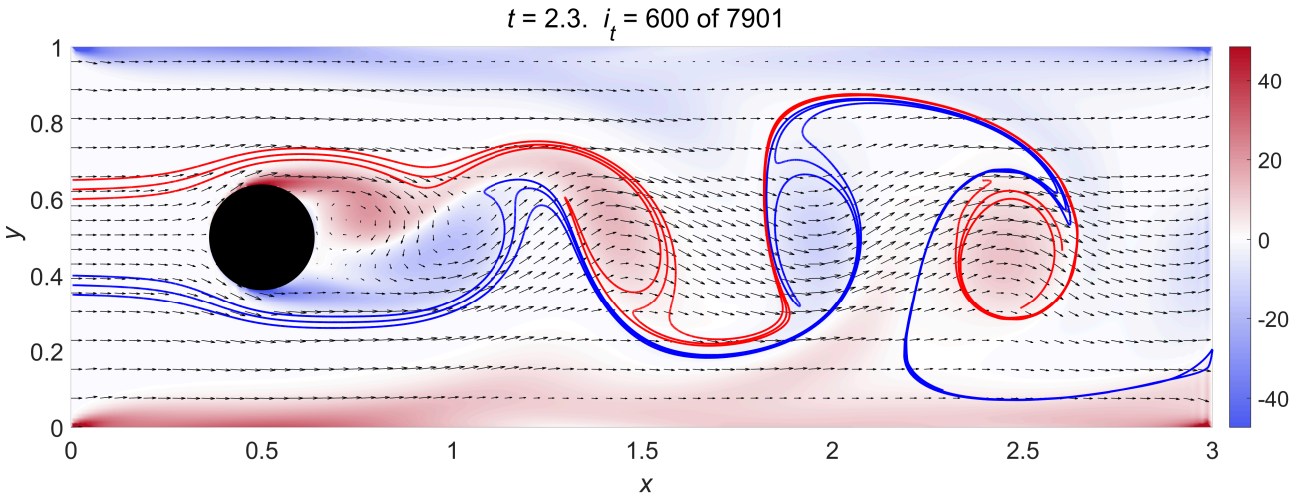

Figure 9 Wake flow past a circular cylinder at $t = 2.3$. Heavy red and blue lines are *streaklines* and the color plot refers to ζ values over the domain.

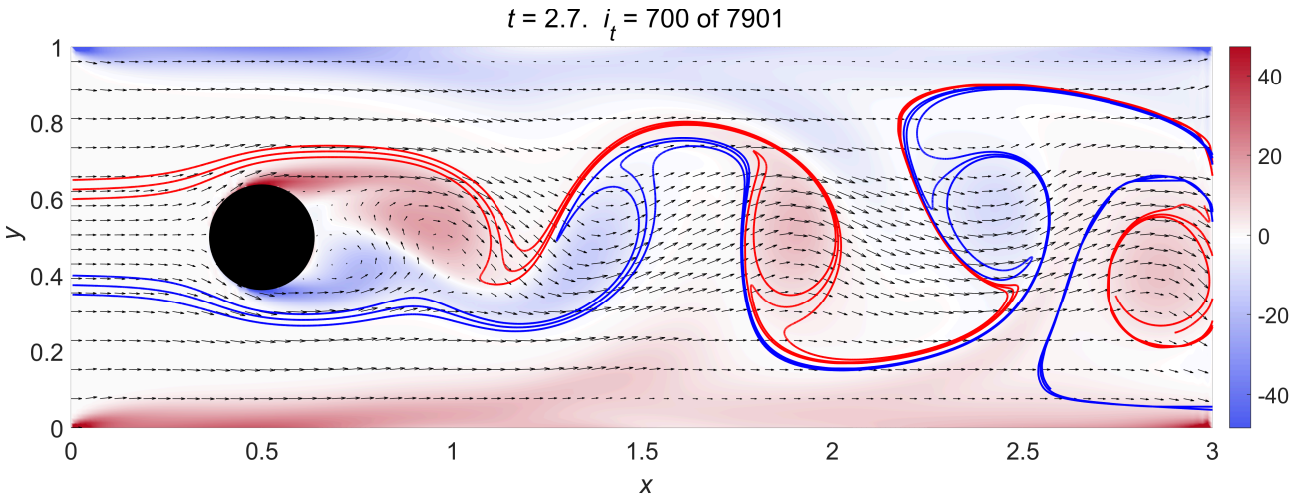

Figure 10 Wake flow past a circular cylinder at $t = 2.7$. Heavy red and blue lines are *streaklines* and the color plot refers to ζ values over the domain.

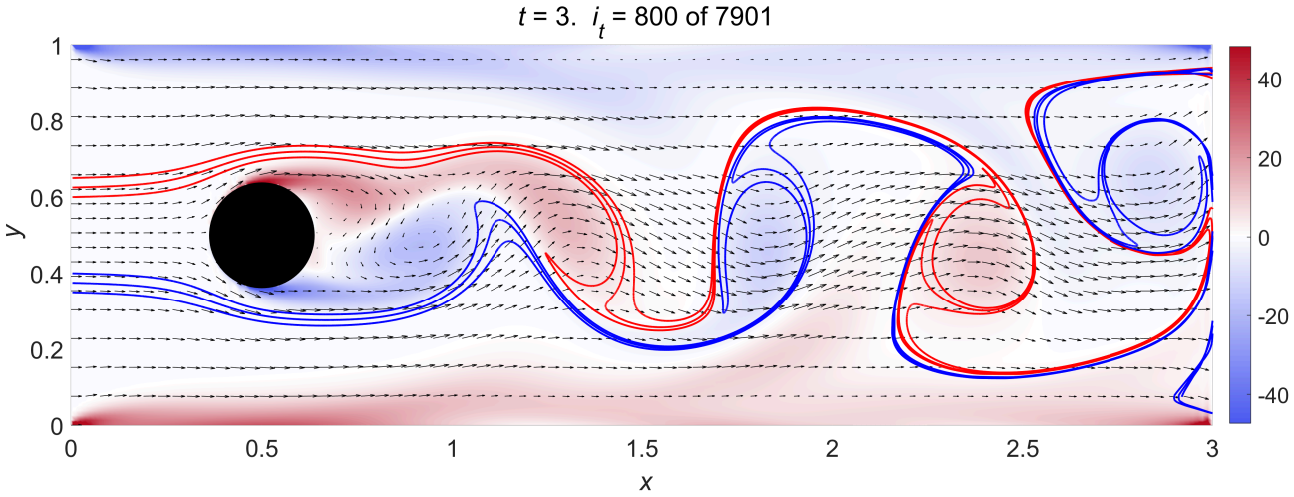

Figure 11 Wake flow past a circular cylinder at $t = 3.0$. Heavy red and blue lines are *streaklines* and the color plot refers to ζ values over the domain.

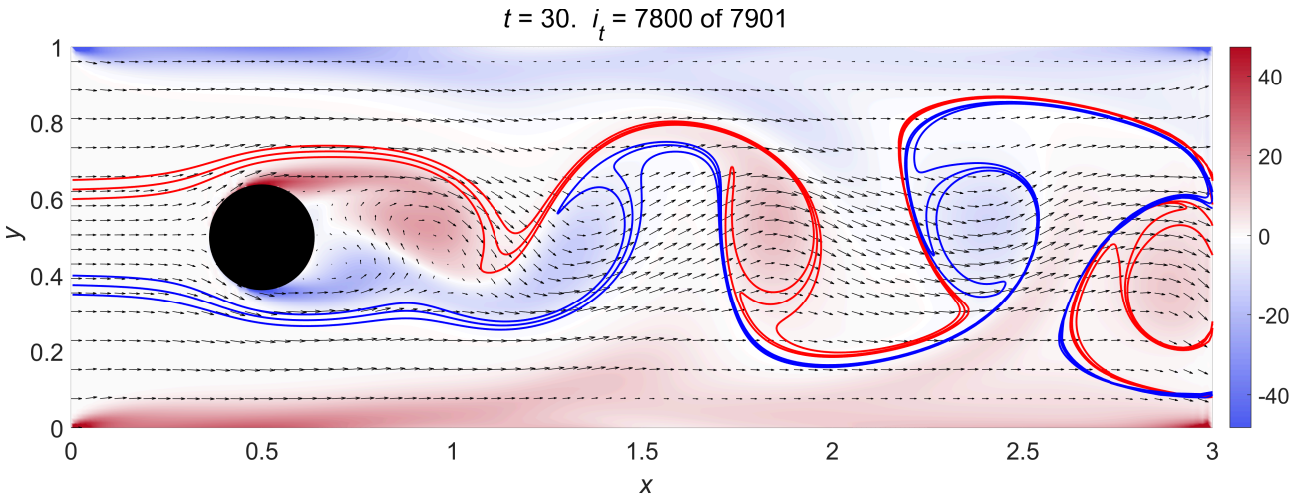

Figure 12 Wake flow past a circular cylinder at $t = T = 30$. Heavy red and blue lines are *streaklines* and the color plot refers to ζ values over the domain.

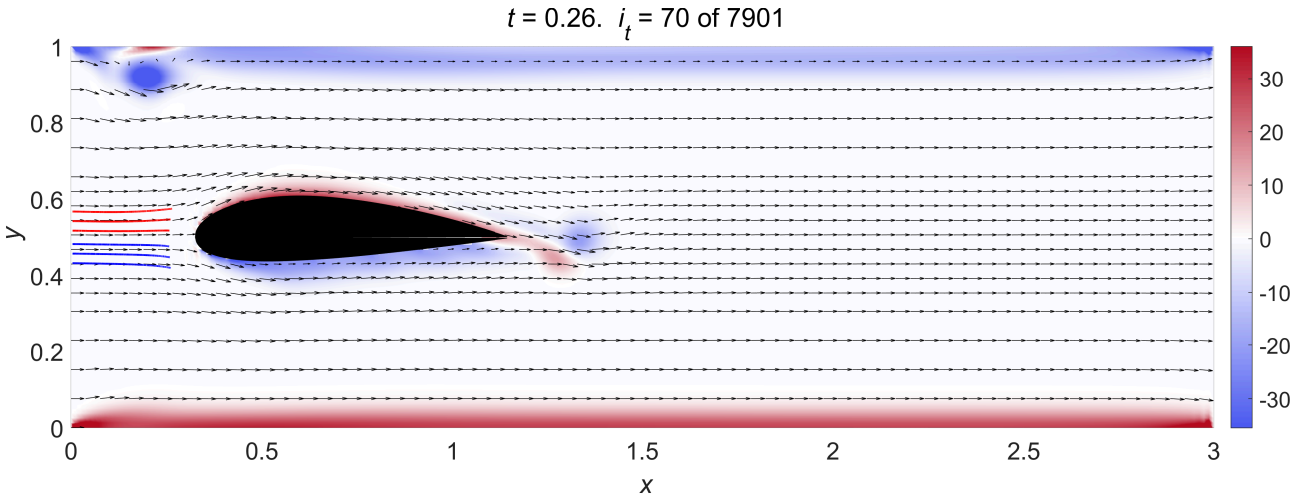

Figure 13 Wake flow past a 2418 NACA airfoil at angle of attack $\alpha = 0$ deg, $t = 0.26$. Heavy red and blue lines are *streaklines* and the color plot refers to ζ values over the domain.

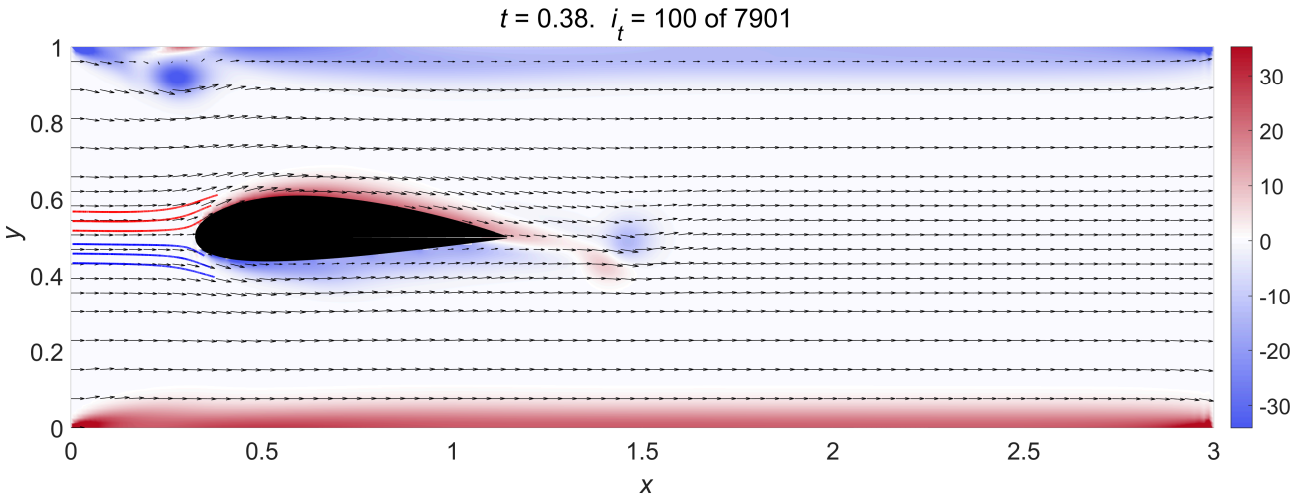

Figure 14 Wake flow past a 2418 NACA airfoil at angle of attack $\alpha = 0$ deg, $t = 0.38$. Heavy red and blue lines are *streaklines* and the color plot refers to ζ values over the domain.

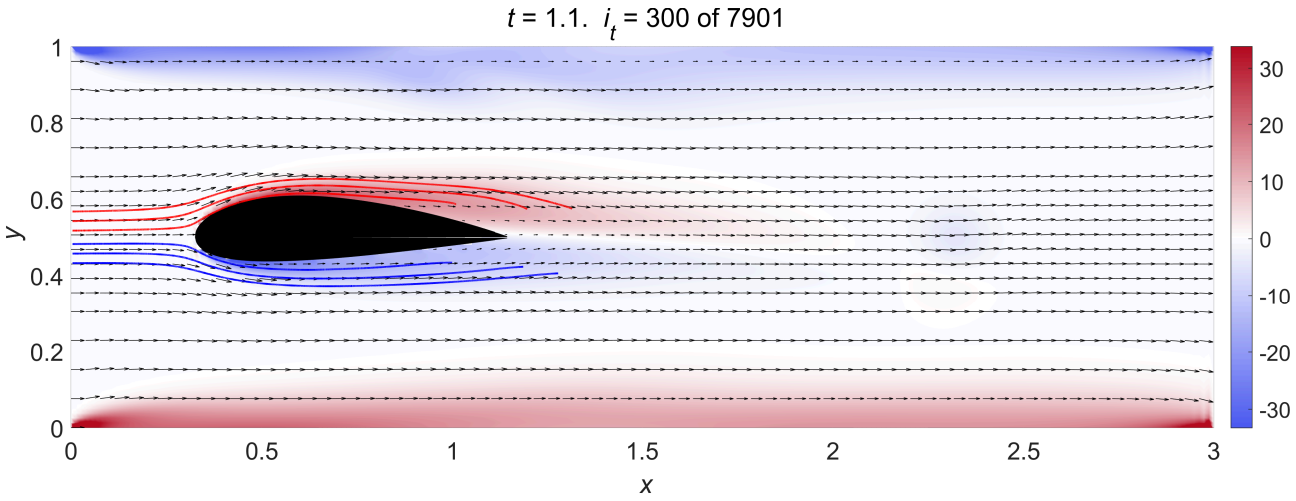

Figure 15 Wake flow past a 2418 NACA airfoil at angle of attack $\alpha = 0$ deg, $t = 1.1$. Heavy red and blue lines are *streaklines* and the color plot refers to ζ values over the domain.

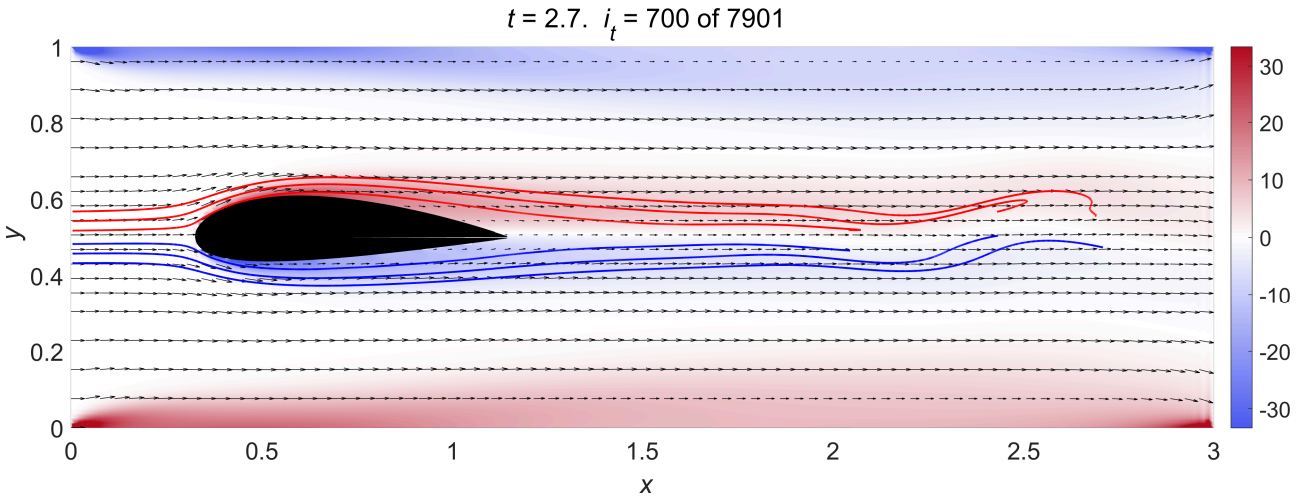

Figure 16 Wake flow past a 2418 NACA airfoil at angle of attack $\alpha = 0$ deg, $t = 2.7$. Heavy red and blue lines are *streaklines* and the color plot refers to ζ values over the domain.

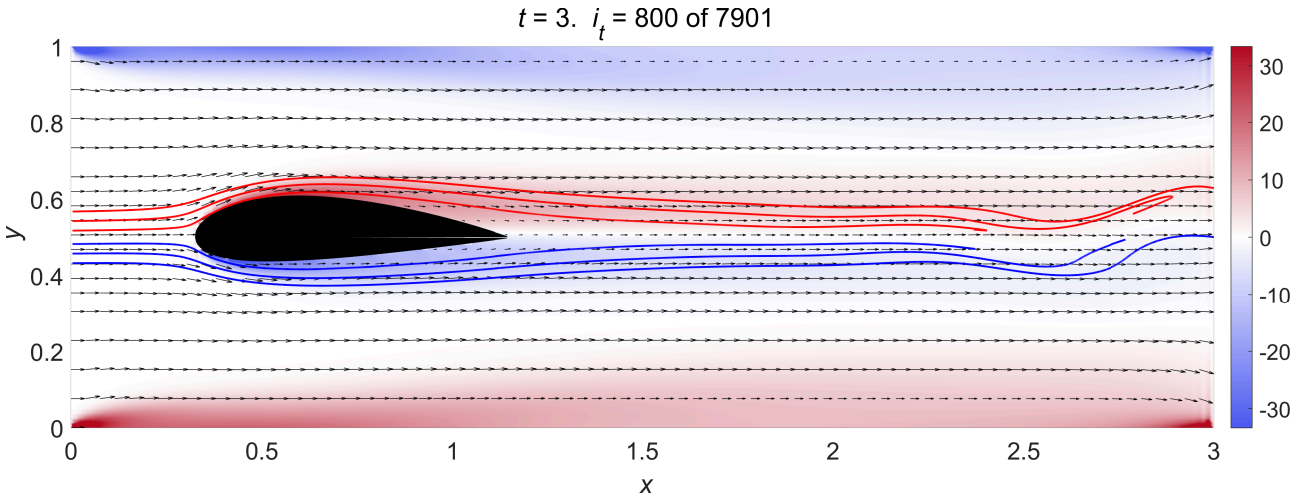

Figure 17 Wake flow past a 2418 NACA airfoil at angle of attack $\alpha = 0$ deg, $t = 3.0$. Heavy red and blue lines are *streaklines* and the color plot refers to ζ values over the domain.

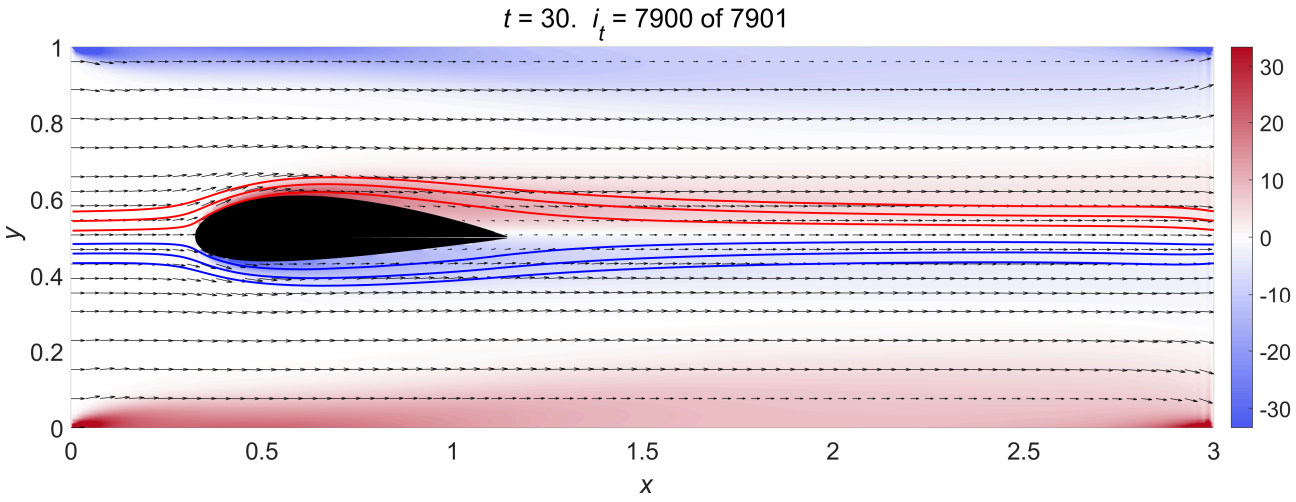

Figure 18 Wake flow past a 2418 NACA airfoil at angle of attack $\alpha = 0$ deg, $t = T = 30$. Heavy red and blue lines are *streaklines* and the color plot refers to ζ values over the domain.

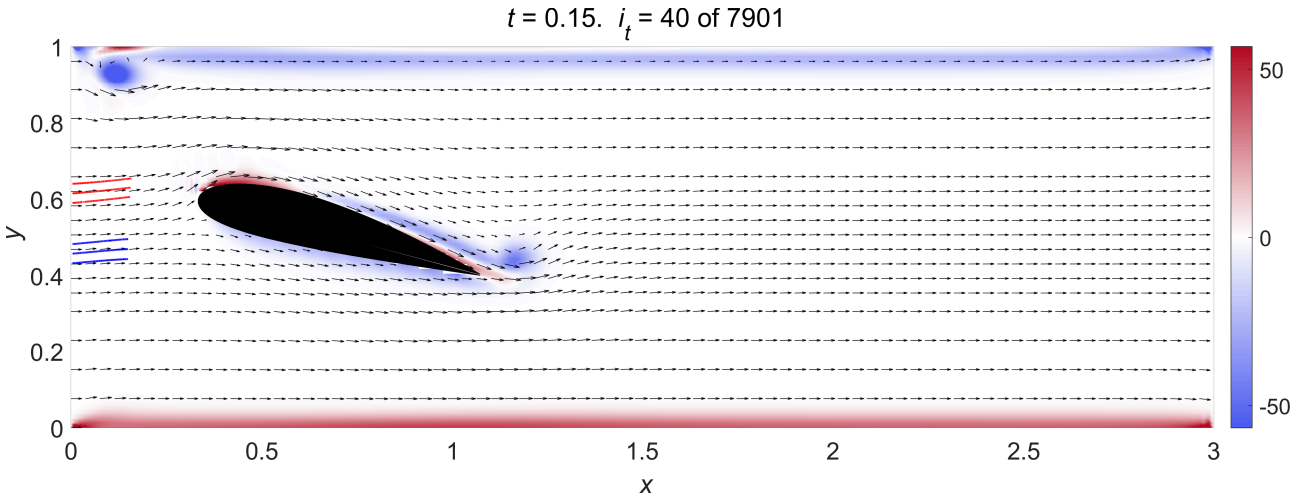

Figure 19 Wake flow past a 2418 NACA airfoil at angle of attack $\alpha = 15$ deg, $t = 0.15$. Heavy red and blue lines are *streaklines* and the color plot refers to ζ values over the domain.

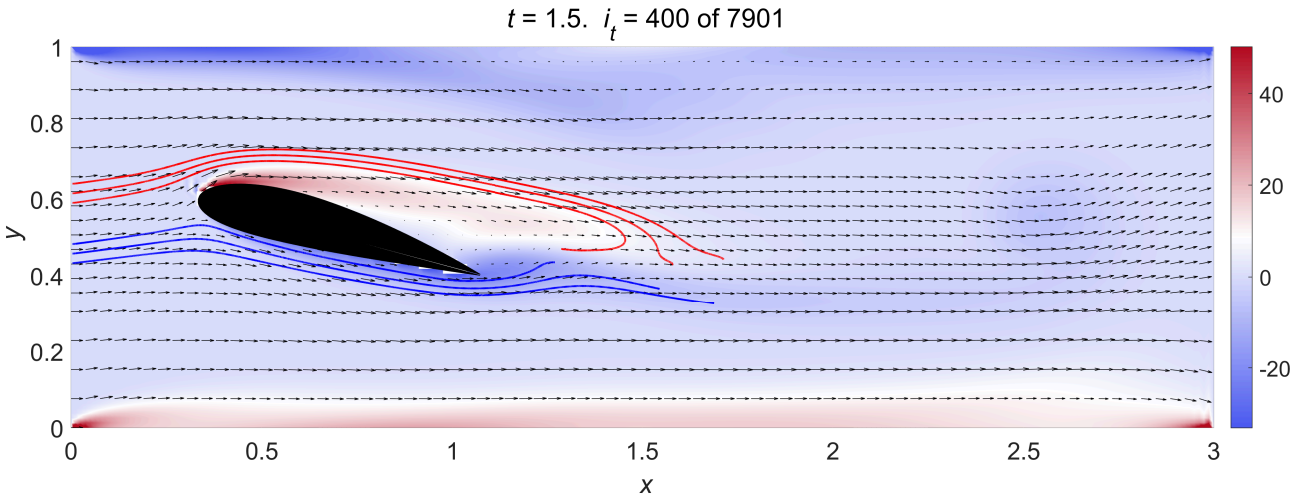

Figure 20 Wake flow past a 2418 NACA airfoil at angle of attack $\alpha = 15$ deg, $t = 1.5$. Heavy red and blue lines are *streaklines* and the color plot refers to ζ values over the domain.

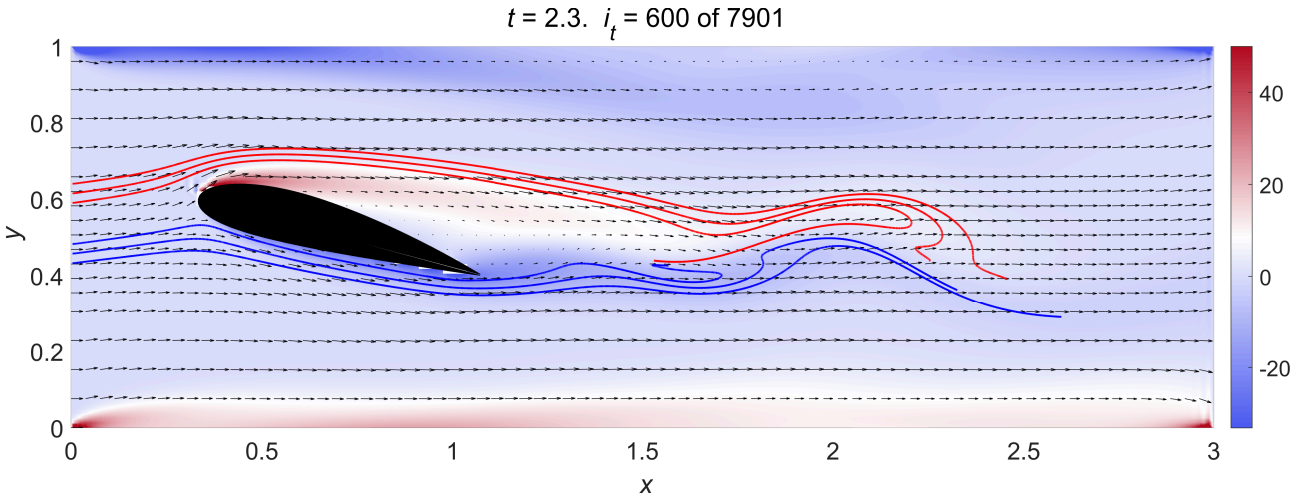

Figure 21 Wake flow past a 2418 NACA airfoil at angle of attack $\alpha = 15$ deg, $t = 2.3$. Heavy red and blue lines are *streaklines* and the color plot refers to ζ values over the domain.

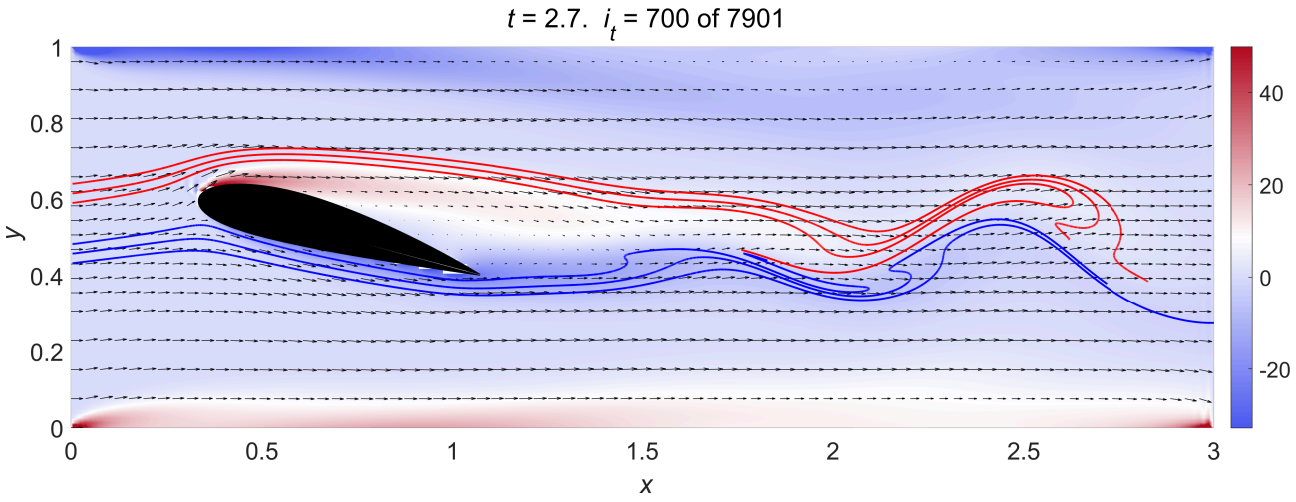

Figure 22 Wake flow past a 2418 NACA airfoil at angle of attack $\alpha = 15$ deg, $t = 2.7$. Heavy red and blue lines are *streaklines* and the color plot refers to ζ values over the domain.

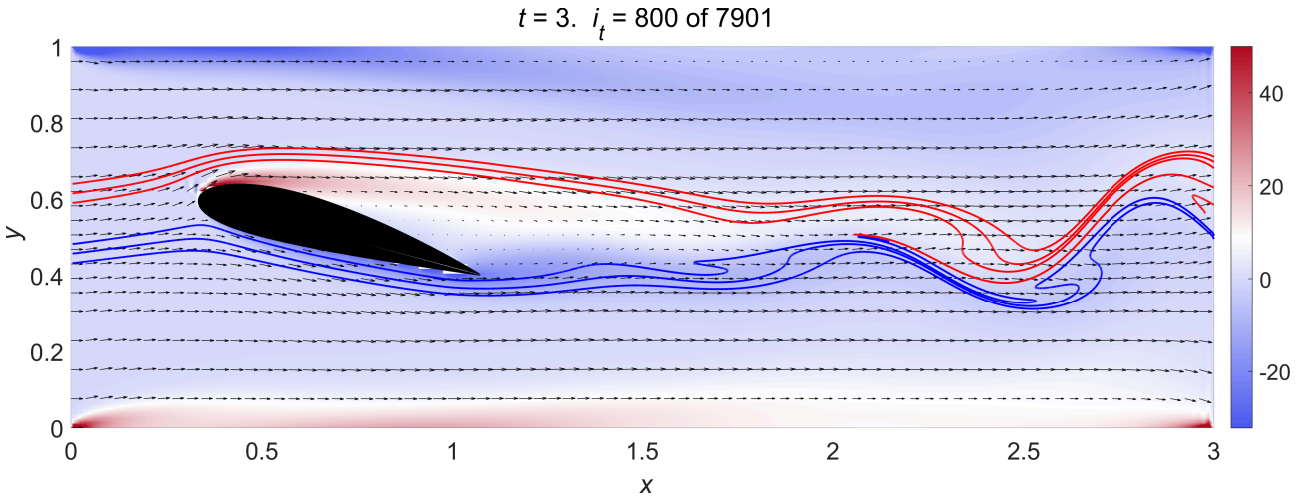

Figure 23 Wake flow past a 2418 NACA airfoil at angle of attack $\alpha = 15$ deg, $t = 3.0$. Heavy red and blue lines are *streaklines* and the color plot refers to ζ values over the domain.

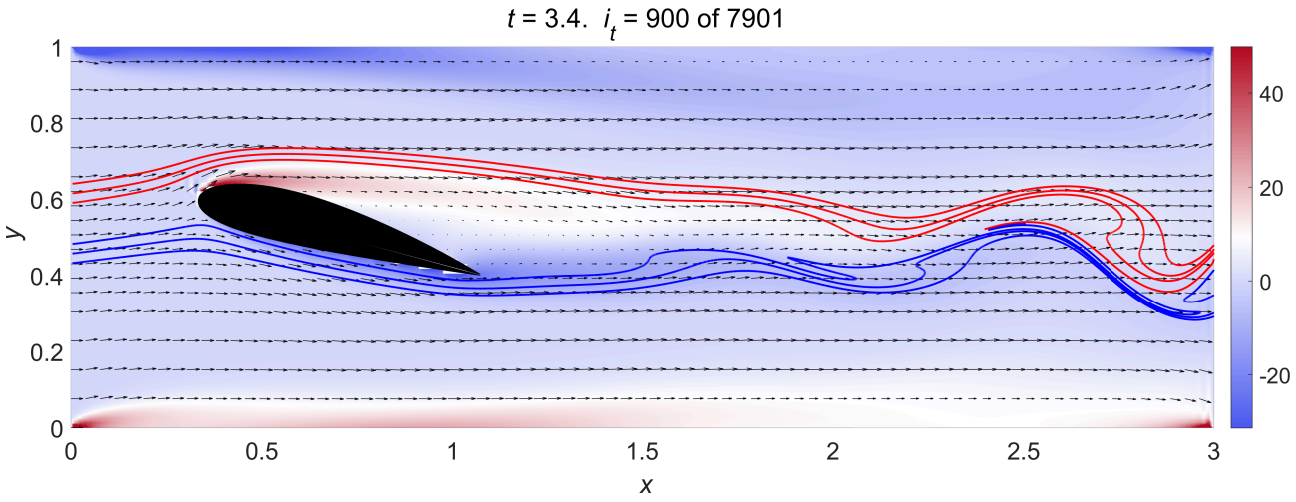

Figure 24 Wake flow past a 2418 NACA airfoil at angle of attack $\alpha = 15$ deg, $t = 3.4$. Heavy red and blue lines are *streaklines* and the color plot refers to ζ values over the domain.

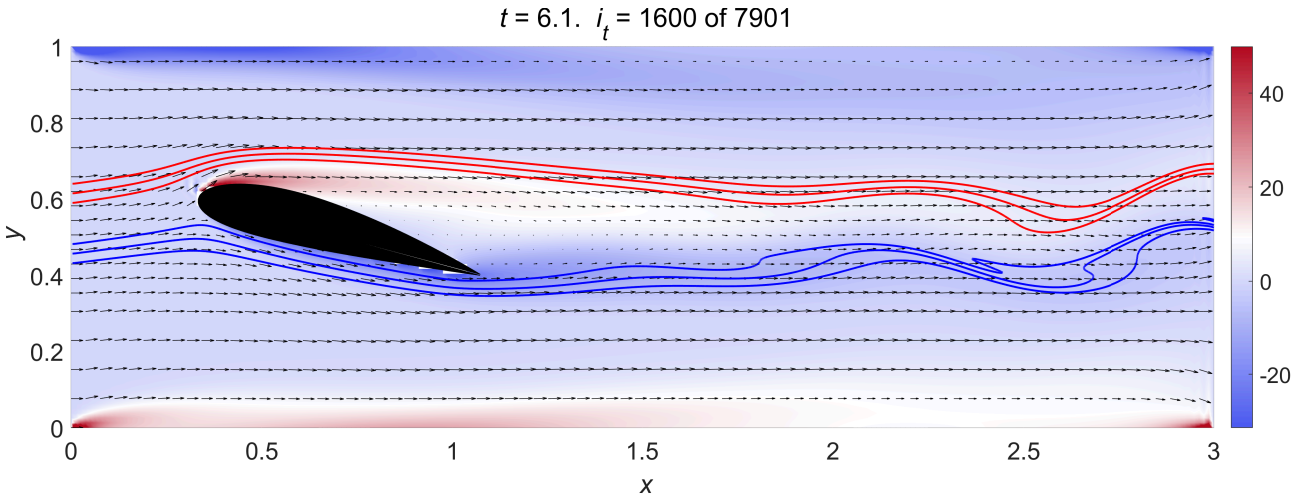

Figure 25 Wake flow past a 2418 NACA airfoil at angle of attack $\alpha = 15$ deg, $t = 6.1$. Heavy red and blue lines are *streaklines* and the color plot refers to ζ values over the domain.

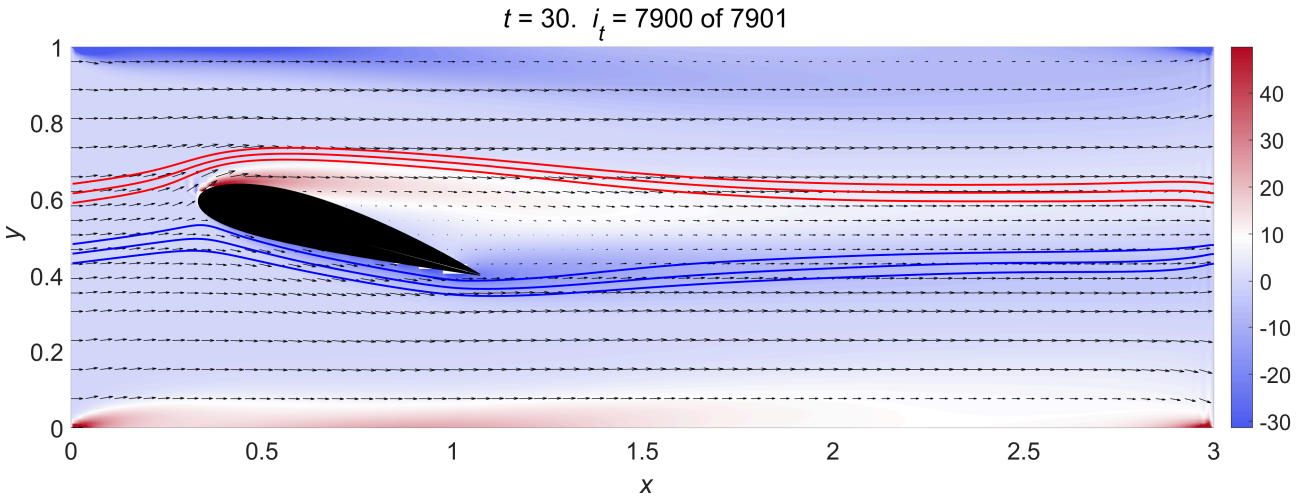

Figure 26 Wake flow past a 2418 NACA airfoil at angle of attack $\alpha = 15$ deg, $t = T = 30$. Heavy red and blue lines are *streaklines* and the color plot refers to ζ values over the domain.

References

[1] J. Anderson. *Fundamentals of aerodynamics*. McGraw-Hill, 2011.

[2] Milton Van Dyke. *An Album of Fluid Motion*. The Parabolic Press, 1988.

[3] Ira H. Abbott and Albert E. Von Doenhoff. *Theory of Wing Sections*. Dover Publications, 1949.

[4] Rodolfo Monti and Raffaele Savino. *Aerodinamica*. Liguori Editore, 1998.

[5] John C. Tannehill, Dale A. Anderson, and Richard H. Pletcher. *Computational Fluid Mechanics and Heat Transfer*. Taylor & Francis, 1984.

[6] Arieh Iserles. *A First Course in the Numerical Analysis of Differential Equations*. Cambridge University Press, 2009.