Materials Today: Proceedings xxx (xxxx) xxx

Contents lists available at ScienceDirect

Materials Today: Proceedings

journal homepage: www.elsevier.com/locate/matpr

Steady solver for incompressible Navier-Stokes equation with automated adaptive mesh refinement using FEniCS

R. Varun Kumar [a], K.V. Nagaraja [b]

[a] Department of Mechanical Engineering, Amrita Vishwa Vidyapeetham, Amrita School of Engineering Bangalore (ASEB), 560035, India
[b] Department of Mathematics, ASEB, India

ARTICLE INFO

Article history:
Available online xxxx

Keywords:
Finite element method
Adaptive mesh refinement
Steady Navier-Stokes equation
Newton method
FEniCS

ABSTRACT

This study demonstrates steady incompressible flow over a cylinder section using the finite element method. An open-source framework was used to increase the accuracy of the results of the analysis. The Python-based open-source computing platform FEniCS was used for the problem formulation. The mesh generator used is Gmsh. FEniCS offers automated adaptive mesh refinement and higher-order elements of order three and higher, which produce more accurate results. These features are not available in most FEM software. The problem DFG benchmark 2D-1 in Featflow was analyzed using the open-source framework. The drag and lift coefficients and the pressure difference between the front and back of the cylinder were found and compared with the benchmark values. The results obtained prove that the proposed open-source framework is incredibly effective in solving fluid flow problems. The FEniCS code to perform this DFG 2D-1 benchmark analysis is given in this work. It shows the problem formulation, setting of boundary conditions, and the adaptive solver. Thus, the same open-source framework can be used to analyze any laminar incompressible flow by simply creating a different geometry and varying the flow characteristics and boundary conditions.

Selection and peer-review under responsibility of the scientific committee of the Fourth International Conference on Recent Advances in Materials and Manufacturing 2022.

1. Introduction

This paper solves the steady Navier-Stokes equation with adaptive mesh refinement using the finite element package FEniCS. The results are compared with a benchmark to show its effectiveness in solving problems.

1.1. Finite element method

The finite element method (FEM) is a numerical method that finds solutions to field problems that occur in the physical world and several engineering applications [1]. FEM can analyze various issues like fluid dynamics, heat transfer, etc. The solution obtained is the distribution of the dependent variables over the spatial domain in which the problem is analyzed. These field problems are described using partial differential equations(PDEs)[2,3]. PDEs govern most mechanical phenomena. For example, fluid flow is governed by the Navier-Stokes equation, and transient heat conduction is governed by the Fourier-Biot equation. These PDEs are solved by meshing the spatial domain into elements that are interconnected at their nodes and giving an approximate solution while trying to minimize the error using various numerical methods and different types of solvers [4,5].

1.2. Finite element method for fluid dynamics

Most computational fluid dynamics (CFD) software uses either a finite difference formulation or the finite volume (FV) technique. However, the FEM has many advantages over the other two methods. It allows fully unstructured and random domain subdivisions. The approximations provided by FEM are at least equal or superior to those produced by finite differences [6]. FV techniques are finite element forms with subdomain collocation. The FV approximation explicitly satisfies conservation equations for each FV. In FEM, the satisfaction of conservation equations is achieved implicitly. Since the general FEM gives superior results, explicit satisfaction of the conservation equations in the FV technique on each element is not beneficial.

E-mail addresses: varunkumarr2001@gmail.com (R. Varun Kumar), kv_nagaraja@bl.amrita.edu (K.V. Nagaraja)

https://doi.org/10.1016/j.matpr.2023.04.425
2214-7853/
Selection and peer-review under responsibility of the scientific committee of the Fourth International Conference on Recent Advances in Materials and Manufacturing 2022.

2. Navier-Stokes equation

2.1. Mathematical formulation

The Navier-Stokes equations are certain PDEs that explain the motion of viscous fluids. The Navier-Stokes equation is a model of fluid flow involving several inherent approximations. Hence it is not exact [7]. Despite its shortcomings, it is a splendid model and remains the foundation of modern CFD. Due to the nonlinear nature of the PDEs, the Navier-Stokes equation has no exact solutions. The nonlinearity is due to the presence of convective acceleration and makes the problem difficult or impossible to solve analytically [8]. There are many ways of representing the Navier-Stokes equations of viscous incompressible fluids [9]. One such formulation with unit density is [10,11]:

$$u \cdot \nabla u + \dot{u} - \nabla \cdot \sigma = f, \tag{1}$$

$$\nabla \cdot u = 0, \tag{2}$$

where u is velocity, v is kinematic viscosity, f is force, and σ is the Cauchy stress tensor for a Newtonian fluid. Eq.1 is more commonly known as the momentum equation, and Eq.2 is known as the continuity equation.

2.2. Steady and unsteady solvers

There are many numerical methods to give solutions for the Navier-Stokes equations. The methods can broadly be classified into 2 types, a steady solver for solutions that are not time-dependent and an unsteady solver for time-dependent and turbulent solutions. The decision of which solver to use is made based on whether the problem has turbulence. Turbulence complicates the numerical method of finding a solution to the Navier-Stokes equations. The nature of the flow is determined by calculating the Reynolds number (Re) [7]. Re is the ratio of inertial forces to viscous forces. The nature of the flow is usually considered laminar for Re less than 1000. When the flow is laminar, a steady solver can be used. In the DFG 2D-1 benchmark presented in this paper, Re is 20. So, a steady solver was used instead of an unsteady solver.

$$Re = \frac{Inertialforces}{Viscousforces} = \frac{\rho V_{avg}^2 L^2}{\mu V_{avg} L} = \frac{\rho V_{avg} L}{\mu} = \frac{V_{avg} L}{\nu} \tag{3}$$

The Newton method is a steady solver that solves the equation in a nonlinear form. Newton method attempts to solve for the velocity by adjusting the velocity. As the iterations progress, the solution begins to converge upon the accurate velocity value. It is a steady solver as it iterates towards a final solution and lacks time-dependent variables compared to unsteady solvers. Thus, steady solvers are poor at simulating turbulence. To derive the weak form of the Newton method, the momentum equation (Eq.1) is multiplied with the velocity test function $\boldsymbol{v}$ and is integrated by parts to give:

$$\int_\Omega \nu \nabla \boldsymbol{u} \cdot \nabla \boldsymbol{v}\, d\Omega + \int_\Omega ((\boldsymbol{u} \cdot \nabla)\boldsymbol{u}) \cdot \boldsymbol{v}\, d\Omega - \int_\Omega p \nabla \cdot \boldsymbol{v}\, d\Omega - \int_{\partial\Omega} \left(\nu \frac{\partial \boldsymbol{u}}{\partial \boldsymbol{n}} - p\boldsymbol{n} \right) \cdot \boldsymbol{v}\, d\sigma = 0 \tag{4}$$

The continuity equation (Eq.2) is multiplied with the pressure test function $\boldsymbol{q}$:

$$-\int_\Omega q \nabla \cdot \boldsymbol{u}\, d\Omega = 0 \tag{5}$$

After imposing the boundary conditions, the weak formulation becomes:

$$\int_\Omega \nu \nabla \boldsymbol{u} \cdot \nabla \boldsymbol{v}\, d\Omega + \int_\Omega ((\boldsymbol{u} \cdot \nabla)\boldsymbol{u}) \cdot \boldsymbol{v}\, d\Omega - \int_\Omega p \nabla \cdot \boldsymbol{v}\, d\Omega - \int_\Omega q \nabla \cdot \boldsymbol{u}\, d\Omega = \int_{\Gamma_N} g\boldsymbol{n} \cdot \boldsymbol{v}\, d\sigma \tag{6}$$

For the problem at hand, the RHS of the above equation is zero. Thus, Eq. (6) becomes:

$$\int_\Omega \nu \nabla \boldsymbol{u} : \nabla \boldsymbol{v} + ((\boldsymbol{u} \cdot \nabla)\boldsymbol{u}) \cdot \boldsymbol{v} - p \nabla \cdot \boldsymbol{v} - q \nabla \cdot \boldsymbol{u} = 0 \tag{7}$$

where $\boldsymbol{u}$ is the velocity trial function, $\boldsymbol{v}$ is the velocity test function, p is the pressure trial function and q is the pressure test function. The convective term $(\boldsymbol{u} \cdot \nabla)\boldsymbol{u}$ is nonlinear. So, Eq.7 is the nonlinear variational form of the Newton method to solve steady incompressible flow.

3. FEniCS to solve incompressible fluid flow

3.1. FEniCS

FEniCS is an open-source software that uses FEM to solve PDEs by turning scientific models into effectual finite element code. FEniCS is easy to get started with and also offers more advanced capabilities for more knowledgeable users [11,12,13]. FEniCS requires the PDE to be expressed in the variational or weak form [14,15]. The variational problem is entered into Python using the Unified Field Language (UFL). Fig. 1 shows the implementation of the Newton method in FEniCS. The geometry and mesh are created using Gmsh [16]. The mesh is imported into FEniCS, and the results are exported and viewed in ParaView [17,18].

3.2. Taylor-Hood elements

Solving the Navier-Stokes Equations using FEM usually involves the use of special pair of elements called Taylor-Hood elements [8,11]. The velocity function space uses continuous $P_q (q \geq 2)$ Lagrange element and the pressure function space uses continuous P_{q-1} Lagrange element. The order of the velocity function space is higher than the order of the pressure function space by 1. Taylor-Hood elements produce a stable solution. Using the wrong pair of elements causes spurious oscillations in the pressure solution. Fig. 2 shows Taylor-Hood elements for the case q = 3. The order of the velocity element is 3 (cubic order) and the order of the pressure element is one less than 3, i.e., 2 (quadratic order).

Fig. 3 shows the creation of a mixed-function space using Taylor-Hood elements. The velocity function space is made of vector elements **(V)** of order 2 and the pressure function space is made of finite elements **(Q)** of order 1. Taylor-Hood element **(TH)** is made by combining **V** and **Q**. Finally, a mixed-function space **W** is created

```
# Variational form of Newton method
F =   inner(grad(u)*u, v)*dx \
    + nu*inner(grad(u), grad(v))*dx \
    - div(v)*p*dx \
    - q*div(u)*dx

dw = TrialFunction(W)
dF = derivative(F, w, dw)
```

Fig. 1. Implementation of the variational form of Newton method (Eq.7) in FEniCS.

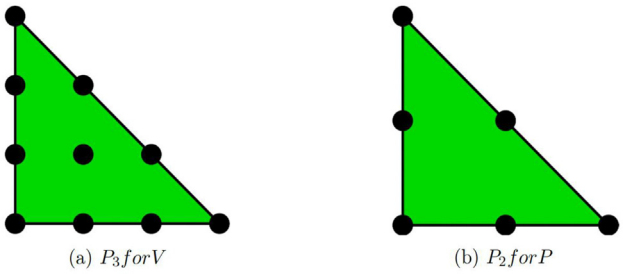

Fig. 2. (a)Velocity element of order 3 and (b)Pressure element of order 2 [14].

using **TH**. After the function space is defined, the trial and test functions for velocity and pressure are created.

3.3. Adaptive mesh refinement

Adaptivity of the mesh to local features of the problem can lead to more efficient simulations, which along with error control produce more reliable simulations [19]. While solving stationary variational problems, FEniCS offers automated goal-oriented error control. Any variational problem can be solved with an adaptive variational solver. Tolerance and the quantity of interest are to be entered as additional input. The adaptive variational solver starts with an initial mesh, after which it estimates the error and determines where to best refine and continues until the error tolerance is reached. The error estimates and refinement indicators are generated for each given variational problem and goal-functional and thus are directly tuned to the problem at hand. Fig. 4 shows flow in a bend solved using the stationary Navier–Stokes equations with the regular mesh. Fig. 5 shows the same flow with the adapted mesh. Fig. 6 shows the implementation of adaptive mesh refinement with the pressure as the quantity of interest and tolerance of 10^{-5}. (See Fig. 7).

4. FEniCS implementation

The DFG benchmark 2D-1 in Featflow simulates a flow in a pipe over a cylinder. This problem can be solved using either the Newton Method or stationary Navier-Stokes equations [8,11]. The drag and lift coefficients and pressure difference between two points are calculated to compare them with the reference results[10]. The reference values are:

$C_D = 5.57953523384$

$C_L = 0.010618948146$

$P_{diff} = 0.11752016697$

4.1. Geometry and boundary conditions

The flow domain is a rectangle minus a circle, $\Omega = [0, 2.2] \times [0, 0.41]$ - B_r (0.2,0.2) with r = 0.05. The circle represents the cross-section of the cylinder. The density of the fluid ρ is 1 and the kinematic viscosity ν is 0.001. The flow is laminar. Pressure difference is calculated between points (0.15, 0.2) and (0.25, 0.2).

4.1.1. Domain boundary conditions

No-Slip conditions are defined for the lower wall $\Gamma_1 = [0,2.2] \times 0$, upper wall $\Gamma_2 = [0,2.2] \times 0.41$ and for the boundary $S = \partial B_r(0.2, 0.2)$,

$$u_{|\Gamma_1} = u_{|\Gamma_2} = u_{|S} \tag{8}$$

A parabolic inflow profile is defined on the left edge $\Gamma_4 = 0 \times [0,0.41]$,

$$u(x,y) = \left(\frac{4Uy(0.41-y)}{0.41^2}, 0\right) \tag{9}$$

where U has a magnitude of 0.3 and is the maximum velocity at the center. The Reynolds number for these boundary conditions is 20. Thus, the flow is laminar. Fig. 8 shows the setting up of velocity and pressure boundary conditions.

4.1.2. Drag and lift coefficients

The formulae for drag and lift forces are as follows:

$$F_d = \int_S \left(\nu \frac{\partial u_\tau}{\partial_\eta} \eta_2 - p\eta_1\right) ds, F_L = -\int_S \left(\nu \frac{\partial u_\tau}{\partial_\eta} \eta_1 - p\eta_2\right) ds \tag{10}$$

The drag and lift coefficients can be derived from the forces:

$$C_D = \frac{2}{U_{mean}^2 L} F_D, C_L = \frac{2}{U_{mean}^2 L} F_L. \tag{11}$$

Fig. 9 shows the calculation of C_D, C_L, and the pressure difference between points (0.15, 0.2) and (0.25, 0.2).

4.2. Results and discussion

The velocity and pressure contours are shown in Figs. 10 and 11 respectively. The extreme values(maximum and minimum values) match that of the benchmark.

```
# Defining Taylor-Hood Elements
order = 2
V = VectorElement("Lagrange", mesh.ufl_cell(), order)
Q = FiniteElement("Lagrange", mesh.ufl_cell(), order-1)

# Creating a mixed space
TH = V * Q
W = FunctionSpace(mesh, TH)

# Defining trial and test functions for velocity and pressure
(v, q) = TestFunctions(W)
w = Function(W)
(u, p) = (as_vector((w[0], w[1])), w[2])
```

Fig. 3. Creation of Taylor-Hood elements and a mixed function space in FEniCS.

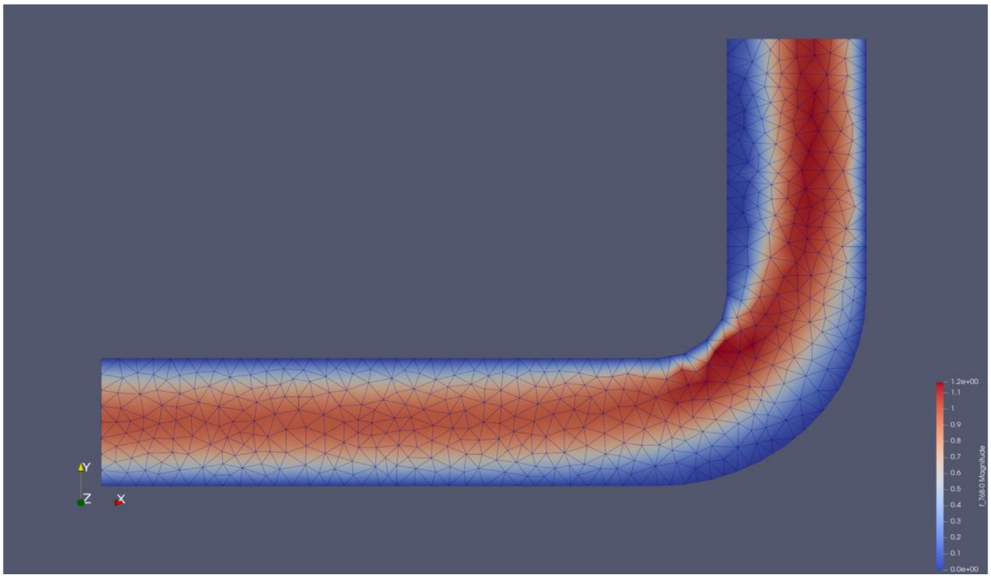

Fig. 4. Initial mesh of flow in a bend.

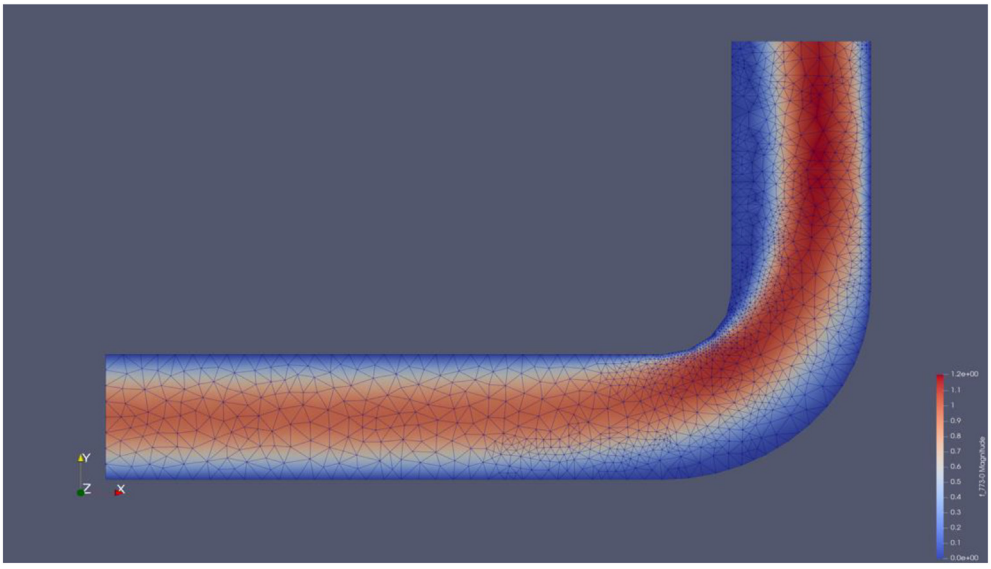

Fig. 5. Adapted mesh of flow in a bend.

```
# Defining pressure as goal functional (quantity of interest)
M = p*ds()

# Define error tolerance
tol = 1.e-5

# Solving the nonlinear variational problem
nsproblem = NonlinearVariationalProblem(F, w, bc, dF)
solver = AdaptiveNonlinearVariationalSolver(nsproblem , M)
solver.solve(tol)
```

Fig. 6. Implementation of adaptive mesh refinement with pressure as the quantity of interest.

The initial and final adapted meshes are shown in Figs. 12 and 13 respectively.

Table 1 lists the values of drag and lift coefficients and the pressure difference between points (0.15, 0.2) and (0.25, 0.2) obtained

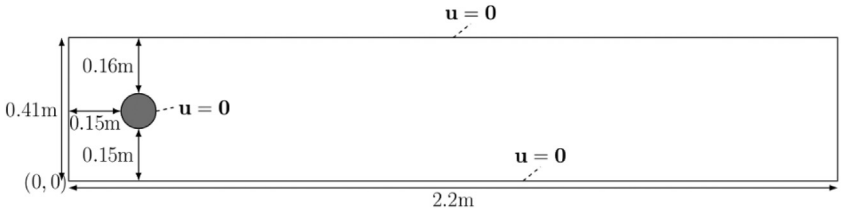

Fig. 7. Flow domain.

```
# Defining boundaries
inflow   = 'near(x[0], 0)'
outflow  = 'near(x[0], 2.2)'
walls    = 'near(x[1], 0) || near(x[1], 0.41)'
cylinder = 'on_boundary && x[0]>0.1 && x[0]<0.3 && x[1]>0.1 && x[1]<0.3'

# Defining inflow profile
inflow_profile = ('4.0*0.3*x[1]*(0.41 - x[1]) / pow(0.41, 2)', '0')

# Defining boundary conditions
bcu_inflow = DirichletBC(W.sub(0), Expression(inflow_profile, degree=2), inflow)
bcu_walls = DirichletBC(W.sub(0), Constant((0, 0)), walls)
bcu_cylinder = DirichletBC(W.sub(0), Constant((0, 0)), cylinder)
bcp_outflow = DirichletBC(W.sub(1), Constant(0), outflow)
bc = [bcu_inflow, bcu_walls, bcu_cylinder, bcp_outflow]
```

Fig. 8. Creating velocity and pressure boundary conditions.

from the coarse mesh and its refinement along with their respective error percentages. The quantity of interest is the pressure and the tolerance for first and second refinements are $10^{-5}$and 10^{-7} respectively. It can be seen from the table that the adaptive mesh refinement significantly brings down the error percentage of the lift coefficient from 52.72% for the initial mesh, to 7.85% and 2.5% for tolerances $10^{-5}$and 10^{-7} respectively. The error percentage of the drag coefficient comes down from 2.69% for the initial mesh, to 0.7% and 0.46% for tolerances $10^{-5}$and 10^{-7} respectively. The tolerance of $10^{-5}$was reached after 3 adaptive iterations. The tolerance of $10^{-7}$was reached after 12 adaptive iterations.

Table 2 lists the values of drag and lift coefficients and the pressure difference between points (0.15, 0.2) and (0.25, 0.2) along with their respective error percentages obtained from the fine mesh and its refinement. The error tolerance of the refinement is 10^{-7}. As expected, the results from the fine mesh are better than those from the coarse mesh. Adaptive mesh refinement brings down the error percentage of the lift coefficient from 8.399% to 0.544%. The error percentage of the drag coefficient comes down from 0.77% to 0.12%. The error percentage of the pressure difference comes down from 0.07% to 0.02%. Since the mesh is automatically refined the most near the cylinder surface and not in all other areas, the computational power is used quite effectively. The tolerance of 10^{-7} was reached after 11 adaptive iterations. The results obtained prove that the proposed open-source framework is incredibly effective in solving fluid flow problems.

Table 3 lists the results obtained from using higher-order elements in the fine mesh. The order of the velocity vector element is 3 and the order of the pressure finite element is 2 as shown in Fig. 2. The tolerance of $10^{-8}$was reached within 2 adaptive iterations. The results from the initial mesh itself are more accurate compared to the initial mesh results from Tables 1 and 2. This is because higher-order elements produce more accurate results at the cost of computational power.

5. Conclusion

As seen in the previous section, the mesh has automatically been refined significantly near the boundary of the cylinder. There is a significant reduction in the error of the lift coefficient due to adaptive mesh refinement. Thus, the results from adaptive mesh refinement and higher-order function spaces greatly improve the accuracy and the results agree with the values from the DFG 2D-1 benchmark. In engineering applications, the geometries used are incredibly complex. Adapting the discretization to the complex features of the geometries in problems like fluid flow will produce more accurate results. FEniCS makes it easy to run simulations and provides many automated features for anyone with a basic understanding of the mathematical basis of the FEM. Since FEM provides at least equal or superior results compared to the FV technique, running CFD analysis using FEM in FEniCS will produce results that are just as good or superior compared to those by other CFD software. The FEniCS code to perform this DFG 2D-1 benchmark analysis is given in this work. It shows the problem formulation, setting of boundary conditions, and the adaptive solver. Thus, the same open-source framework can be used to analyze any laminar incompressible flow by simply creating a different geometry and varying the flow characteristics and boundary conditions.

Data availability

No data was used for the research described in the article.

```
(u0, p0) = w.root_node().split()
(u1, p1) = w.leaf_node().split()
imesh = w.root_node().function_space().mesh()
fmesh = w.leaf_node().function_space().mesh()

class Cylinder(SubDomain):
    def inside(self, x, on_boundary):
        return on_boundary and (x[0]>0.1 and x[0]<0.3 and x[1]>0.1 and x[1]<0.3)

def results(meshr, ur, pr):

    domainBoundaries = MeshFunction("size_t", meshr ,mesh.topology().dim() - 1)
    domainBoundaries.set_all(0)
    Cylinder().mark(domainBoundaries,5)
    dsr = ds(subdomain_data = domainBoundaries)
    n = -FacetNormal(meshr)
    dObs = dsr(5)
    u_t = inner(as_vector((n[1], -n[0])), ur)
    cd = assemble(Form(500 * (nu * inner(grad(u_t), n) * n[1] - pr * n[0]) * dObs))
    cl = assemble(Form(-500 * (nu * inner(grad(u_t), n) * n[0] + pr * n[1]) * dObs))
    print('Drag coeff =',cd)
    print('Lift coeff =',cl)

    a_1 = Point(0.15, 0.2)
    a_2 = Point(0.25, 0.2)
    p_diff = pr(a_1) - pr(a_2)
    print("Pressure difference =",p_diff)

    refcd = 5.57953523384
    refcl = 0.010618948146
    refp = 0.11752016697
    print('Drag coeff Error % =',(abs(refcd-cd)*100/refcd))
    print('Lift coeff Error % =',(abs(refcl-cl)*100/refcl))
    print('Pressure difference Error % =',(abs(refp-p_diff)*100/refp))

print("Results from Original Mesh :")
results(imesh, u0, p0)
print("Results from Adapted Mesh :")
results(fmesh, u1, p1)
```

Fig. 9. Postprocessing to calculate C_D, C_L, and the pressure difference between points (0.15, 0.2) and (0.25, 0.2).

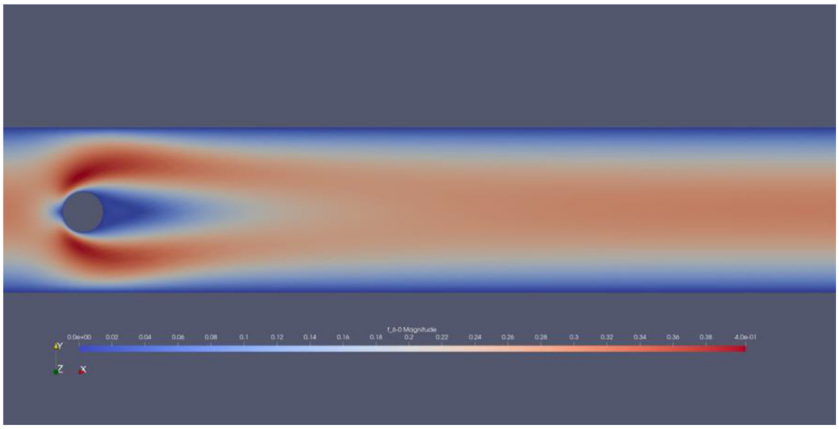

Fig. 10. Velocity contour.

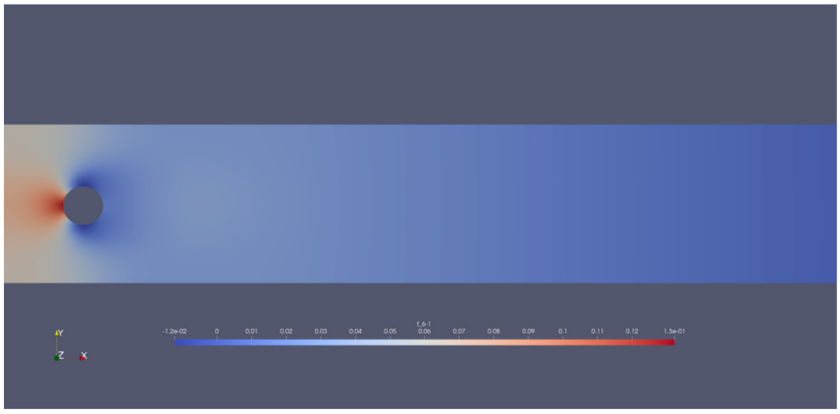

Fig. 11. Pressure contour.

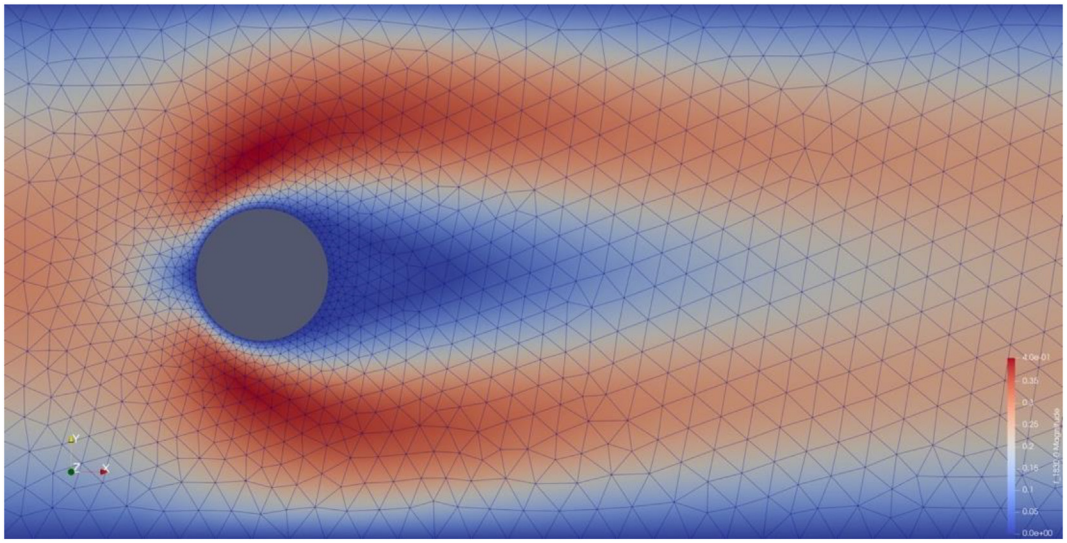

Fig. 12. Velocity contour with initial mesh.

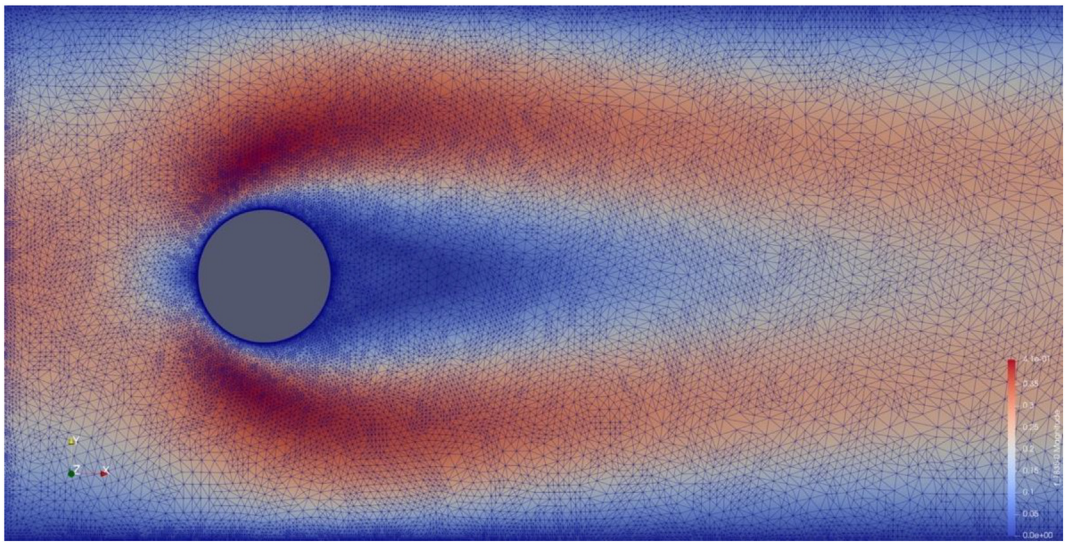

Fig. 13. Velocity contour with final adapted mesh.

Table 1
Result from coarse mesh with 591 nodes and 1058 triangular elements.

Mesh	C_D	C_L	Pressure difference	C_DError%	C_LError%	Pressure difference error %
Initial mesh	5.42937817	0.016217131	0.117173019	2.6912108876	52.718805833	0.29539420542
Adapted mesh-10^{-5}	5.54053379	0.009784911	0.116993041	0.6990088323	7.8542333809	0.44854108073
Adapted mesh-10^{-7}	5.5541028922	0.010353190	0.117712192	0.4558146959	2.5026753522	0.16339768889

Table 2
Result from fine mesh with 2108 nodes and 3968 elements with pressure tolerance 10^{-7}.

Mesh	C_D	C_L	Pressure difference	$C_D$error%	$C_L$error%	Pressure difference error %
Initial mesh	5.53649095	0.011510871	0.11743604612	0.77146719036	8.39935434917	0.0715799281
Adapted mesh	5.57307526	0.010561179	0.11754359631	0.11577969898	0.54402068999	0.01993644986

Table 3
Result from fine mesh with higher-order elements with pressure tolerance 10^{-8}.

Mesh	C_D	C_L	Pressure difference	$C_D$error%	$C_L$error%	Pressure difference error %
Initial mesh	5.557637461	0.010466692	0.1172789756	0.3924658967	1.4338111889	0.20523401277
Adapted mesh	5.564497649	0.010561156	0.1173420911	0.2695132003	0.5442354062	0.15152789996

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors are greatly indebted to Amrita Vishwa Vidyapeetham (ASEB), for their continuous support and encouragement in completing this work.

References

[1] Robert D. Cook, David S. Malkus, Michael E. Plesha, Robert J. Witt Concepts and Applications of Finite Element Analysis, 4th Edition.
[2] Susanne C. Brenner, L. Ridgway Scott; The Mathematical Theory of Finite Element Methods Springer New York, NY 2008 [https://doi.org/10.1007/978-0-387-75934-0].
[3] Kumar, Varun & Kallur, Nagaraja & Kovács, Endre & Shah, Nehad Ali & Chung, Jae & B.C, Prasannakumara. (2023). Accelerating finite element modeling of heat sinks with parallel processing using FEniCSx. Case Studies in Thermal Engineering. 44. 102865. 10.1016/j.csite.2023.102865
[4] M. G. Larson and F. Bengzon. The Finite Element Method: Theory, Implementation, and Applications. Texts in Computational Science and Engineering. Springer, 2013.
[5] Olek C Zienkiewicz, R. L. Taylor, J.Z. Zhu; The Finite Element Method: Its Basis and Fundamentals 6th Edition - April 18, 2005.
[6] Olek Zienkiewicz, Robert Taylor, P. Nithiarasu; The Finite Element Method for Fluid Dynamics 7th Edition -November 14, 2013.
[7] Yunus Cengel, John Cimbala Fluid Mechanics: Fundamentals and Applications 4th Edition.
[8] J. Donea, A. Huerta, Finite Element Methods for Flow Problems, Wiley Press, 2003.
[9] G.P. Galdi An Introduction to the Mathematical Theory of the Navier-Stokes Equations Steady-State Problems Springer New York, NY 2011 [https://doi.org/10.1007/978-0-387-09620-9].
[10] Grzegorz Łukaszewicz, Piotr Kalita Navier–Stokes Equations - An Introduction with Applications – 2016.
[11] A. Logg, K.-A. Mardal, G. N. Wells et al. Automated Solution of Differential Equations by the Finite Element Method, Springer(2012). [doi.org/10.1007/978-3-642-23099-8].
[12] Hans Petter Langtangen, Anders Logg Solving PDEs in Python-The FEniCS Tutorial I 2016.
[13] M.S. Alnaes, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M.E. Rognes, G.N. Wells, The FEniCS Project Version 1.5, Arch. Numer. Softw. 3 (2015), https://doi.org/10.11588/ans.2015.100.20553.
[14] H. P. Langtangen and K.-A. Mardal. Introduction to Numerical Methods for Variational Problems. 2016. http://hplgit.github.io/fem-book/doc/web/.
[15] V. Kumar, K. Chandan, K.V. Nagaraja, M.V. Reddy, Heat conduction with Krylov subspace method using FEniCSx, Energies 15 (21) (2022) 8077, https://doi.org/10.3390/en15218077.
[16] C. Geuzaine, J.-F. Remacle, Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities, Int. J. Numer. Meth. Eng. 79 (11) (2009) 1309–1331.
[17] Ahrens, James, Geveci, Berk, Law, Charles, ParaView: An End-User Tool for Large Data Visualization, Visualization Handbook, Elsevier, 2005, ISBN-13: 978-0123875822
[18] Ayachit, Utkarsh, The ParaView Guide: A Parallel Visualization Application, Kitware, 2015, ISBN 978-1930934306
[19] M. Rognes, A. Logg, Automated goal-oriented error control I: stationary variational problems, SIAM J. Sci. Comput. 35 (2012), https://doi.org/10.1137/10081962X.