Numerical modeling and simulation of a flapping flexible simplified insect wing vein

Master Thesis

Submitted as a partial prerequisite for
MSc in Fluid Mechanics
and Nonlinear Physics

Institute:
Institut de Mathématiques de Marseille (I2M)

Supervisors:
Prof. Kai Schneider
Dinh HungTruong

Ranjan Dhakal

Aix-Marseille Université

Date: September 11, 2018

Numerical modeling and simulation of flapping flexible simplified insect wing vein

Abstract

Numerical simulation of bio-inspired locomotion, especially of insect flight, has evolved with a variety of rigid and non-rigid approximation models. To this end, mass spring models have been known for their versatility as an efficient method with a convenience in degrees of flexibility for solid modeling. This can be coupled with a flow solver using a renowned Fourier spectral discretization technique. The coupling is handled by the volume penalization method, which is able to simulate flexible and deforming obstacles.

Here, we study the fluid-structure interaction of a simplified insect wing i.e. a vein, which represents a key structural component in flight. First, we study the properties of our solid solver, particularly using the condition number of the Jacobian with respect to increasing stiffness and number of mass points. We also study the complexity of our solid solver and finally compare our mass-spring model solution with a nonlinear beam model. We validate our fluid solver by studying the aerodynamic forces and coefficients of a circular cylinder inside a channel. This is done using a classical Turek benchmark for flow past a circular cylinder. Finally, we validate and discuss the difficulties of our coupled solver, for a flexible beam attached with a circular cylinder, using the classical Turek benchmark for fluid-structure interaction.

Acknowledgments

I would like to express my heartly thanks to my supervisor Prof. Kai Schneider for his constant guidance and motivation throughout the past six months. He is really passionate about his work and does it with elegance and passion. Also, I would like to thank Dinh Hung Truong, my second supervisor, for his fruitful discussions throughout this internship. He spent much of his time to guide me through a lot of technical issues in the Matlab code that I was working.

I am really grateful towards Labex MEC Director Prof. Alain Pocheau and CNRS for providing me financial support for the entire academic year. Finally, I would not forget to express my special thanks to my course coordinator Dr. Thomas Leweke. Without him and also Prof. Alain Pocheau, I would not have been sitting here and doing my business.

Contents

Chapter 1

Introduction

Throughout history, animal locomotion has always been the subject of major interest for scientists and engineers, especially in case of flying and swimming animals. With the advancement in computing resources in the recent years, scientific communities are seriously involved in applying biomimetic solutions to real technical problems. They are always looking for more efficient and user-friendly with less noise and vibration as well as low power consuming optimum design. This is commonly seen in nature for e.g. in fish, birds and insects. However the complexity of actuating their motion as well as their physical and material properties are extremely challenging.

Current researchers have now understood that the mystery behind these extremely efficient flyer/swimmer comes from the fact that the fluid flow conditions are three dimensional and highly unsteady. The unsteadiness of the flow has been utilized to get a laminarized flow and with much lower drag coefficient. Therefore, the classical fixed-wing flight is replaced by a flexible wing subjected to undulatory motion to generate both the lift and thrust.

The aim of this thesis is to study the fluid structure interaction of a simplified insect wing vein. The main interest of our work is to understand the kinematics of a flapping flexible structure and to obtain the aerodynamic forces and coefficients using our coupled fluid and solid solver.

In chapter 2, we will discuss the dynamics of the insect wing-structure in a nutshell. We will study the properties of our solid solver, particularly, the condition number of the Jacobian with respect to increasing stiffness and number of mass points. Then we access the computational complexity of our solver and finally perform a validation test.

In chapter 3, we will discuss the spectral method and the volume penalization techniques, which is the building block of our flow solver. We will perform some flow simulation and make some comparison with the classical Turek benchmark for flow past a circular cylinder. Following this idea, we will particularly look at the mask function describing moving and flexible obstacles in chapter 4. Finally, we will perform some simulation for fluid-structure interaction with another Turek benchmark test for a flexible beam attached with a circular cylinder in a channel flow.

Chapter 2

Solid Solver

The insect wing structure is really intricate, which also involves motion with multiple degrees of freedom as well as flexible material properties. Keeping this in mind, we use an elastic rod as the flexible object to best represent our simplified insect wing model. This means, our model is able to illustrate almost any initial arbitrary shape in 3D (for example a curved line or a vein).

One of the key elements of our work is to apply a model and better understand the flexibility of our insect wing in 3D space. Here we have the luxury of taking account into two-dimensional flexibility. This means, we can extend our model and combine the rod (like a vein) with a membrane in order to describe the deformation within the overall planform area.

Due to the convenience in degrees of flexibility as well as efficient solver, we use a mass-spring model for an elastic rod which is based on Lagrangian formulation in order to describe the dynamics of a flexible object. This will enable us to characterize the prime effect of flapping motion in our insect wing.

In order to describe the rod flexibility, here we use a collection of segments, which comprise bending deformations in both y and z direction along with the extension deformation in x direction. This can be seen below in the figure 2.2.

Here, we define the coordinate of a lumped mass using x, y and z as the corresponding position at the end of each segment, while θ and ϕ denote the bending angles in the XY and XZ planes correspondingly. For simplicity, we have shown a 2D representation of the rod in 3D. Due to the symmetry with the Y axis and the angle θ, we have omitted the Z axis and the angle ϕ.

2.1 Governing Equations

Based on the global coordinate approach, Hung Truong developed a new model, which allows us to independently describe the position of any arbitrary segment l_i using two angles, i.e. θ and ϕ [8]. The advantage of this approach is that at every time step, there is no dependency on the neighbouring segment to resolve the deflecting angles. This means that there is no difficulty in describing the spatial angle.

Please note that here we use an independent projection for angular deformation along

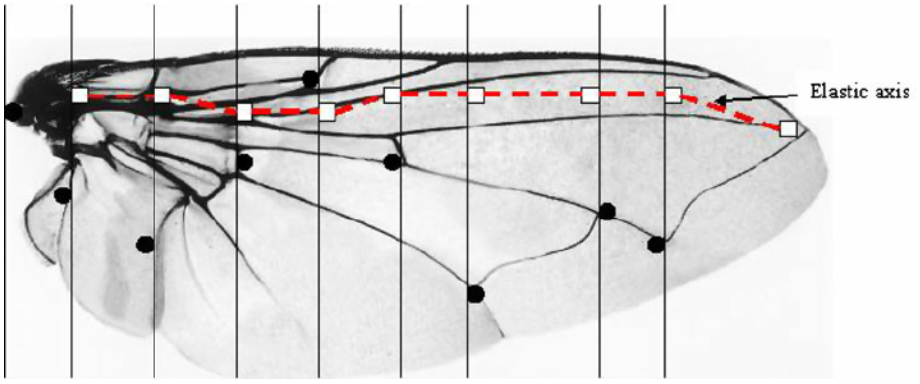


Figure 2.1: Wing of Calliphora with veins representing an elastic rod/beam [7]©

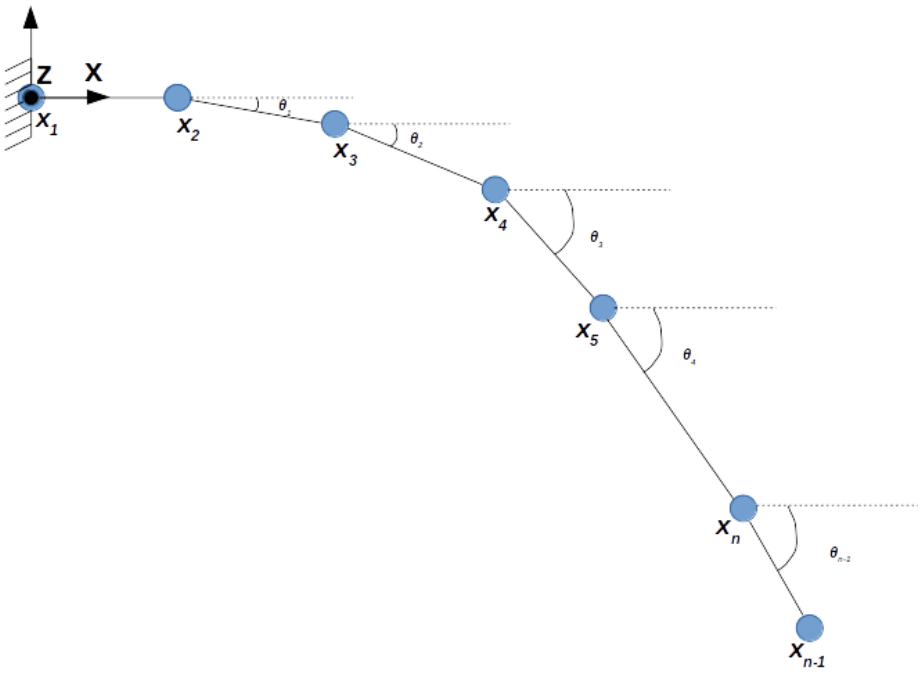


Figure 2.2: Rod in 3D illustrated by a 2D cantilever beam loaded by gravity. Note that the symmetrical OXY plane is coming out of the plane

OXY and OYZ plane rather than using the polar coordinate representation, due to the asymmetry in defining the polar angle of the free end trajectory in the OYZ plane for our rod. This can be seen below in the figure 4.5.

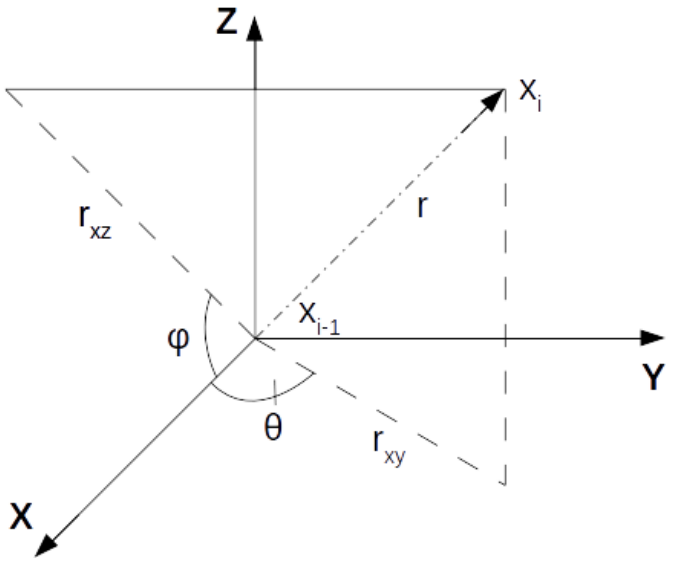


Figure 2.3: Projection of the bending deformation shown by angles in both y and z.

This ensures that the response to the external force are the same for both y and z directions, for example when loaded by gravity (i.e. $g_y = g_z = 0.7\sqrt{2}$). For further detail, please see [8].

However, the disadvantage of this model is that we have two singularity points, i.e. either when both x and y are zero, or when both x and z are zero. This is due to the fact that z is identically defined as y. Now, we can define our segment length l_i and the angles θ and ϕ[1] as follows,

$$\begin{aligned} l_i &= \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 + (z_i - z_{i-1})^2} \\ \theta_i &= arctan2(y_i - y_{i-1}, x_i - x_{i-1}) \\ \phi_i &= arctan2(z_i - z_{i-1}, x_i - x_{i-1}) \end{aligned} \tag{2.1}$$

The governing equations for describing the dynamics of our flexible body can be obtained using the Lagrangian formulation.

2.1.1 Lagrangian Formulation

Lagrangian formulation is a generalization of Newtonian mechanics, which deals with energy. Unlike the Newtonian formalism where calculating the forces exerted on every object

[1]Please note that the $arctan2$ is a matlab command for tan^{-1}. It is used instead of standard $arctan$ because of its advantage to include singular cases.

requires taking sines and cosines individually, this formulation is in fact a very straight forward method based on calculus of variation [2].

However, it is restricted to a system where the forces are purely conservative. Such forces are derived from a potential energy function using differentiation. As a result, it is lot more simpler and quicker to get the same equations of motion. Without going into further details, the Lagrange equations (a.k.a. Euler-Lagrange equations) can be determined using,

$$\frac{d}{dt}\Big(\frac{\partial \mathcal{L}}{\partial \dot{q_j}}\Big) - \frac{\partial \mathcal{L}}{\partial q_j} = Q_j \tag{2.2}$$

where $q = [x_3, ..., x_{n+1}, y_3, ..., y_{n+1}, z_3, ..., z_{n+1}]^T$ with $j = 1, ..., 3n-3$ and Q_j is the total energy of the system. Here, the aim of our study is also to use such a simple but conservative approach to model the dynamics of our elastic rod, where $\mathcal{L}$ is the difference between Kinetic and Potential Energy.

$$\begin{aligned}\mathcal{L} = \sum_{i=1}^{n-1} \frac{1}{2} m_i v_i^2 - \sum_{i=1}^{n-1} \frac{1}{2} k_{(e)i}(l_i - l_{0,i})^2 \\ - \frac{1}{2} k_{(b)1}(\theta_1 - \theta_{0,1})^2 - \sum_{i=2}^{n-1} \frac{1}{2} k_{(b)i}(\theta_i - \theta_{i-1} - \theta_{0,i} + \theta_{0,i-1})^2 \\ - \frac{1}{2} k_{(b)i}(\phi_1 - \phi_{0,1})^2 - \sum_{i=2}^{n-1} \frac{1}{2} k_{(b)i}(\phi_i - \phi_{i-1} - \phi_{0,i} + \phi_{0,i+1})^2\end{aligned} \tag{2.3}$$

and, $v_i^2 = \dot{x}_{i+2}^2 + \dot{y}_{i+2}^2 + \dot{z}_{i+2}^2$. Since the points x_1 and x_2 are held as clamped boundary condition, they do not undergo bending deformation. Therefore for $i = 1, ..., n-1$, we can redefine our segment l_i as,

$$\begin{aligned} l_i &= \sqrt{(x_{i+2} - x_{i+1})^2 + (y_{i+2} - y_{i+1})^2 + (z_{i+2} - z_{i+1})^2} \\ \theta_i &= arctan2(y_{i+2} - y_{i+1}, x_{i+2} - x_{i+1}) \\ \phi_i &= arctan2(z_{i+2} - z_{i+1}, x_{i+2} - x_{i+1}) \end{aligned} \tag{2.4}$$

Therefore, after taking the derivatives of the Lagrangian $\mathcal{L}$ from the equation (2.3), the Lagrangian formulation (2.2) can be decomposed and rewritten for each variable inside the vectors q and $\dot{q}$.

2.2 Numerics

Combining all the terms of equations for kinetic and potential energy (Detailed in appendix (A.1), (A.2), (A.3) and (A.4)) together, we obtain a system of $3(n-1)$ nonlinear ordinary differential equations, written in a compact form as,

$$\begin{cases} m_i \ddot{x}_{i+2} - F_{(x)i,int} - F_{(x)i,ext} - F_{(x)i,damp} = 0 \\ m_i \ddot{y}_{i+2} - F_{(y)i,int} - F_{(y)i,ext} - F_{(y)i,damp} = 0 \qquad i = 1, ..., n-1 \\ m_i \ddot{z}_{i+2} - F_{(z)i,int} - F_{(z)i,ext} - F_{(z)i,damp} = 0 \end{cases} \tag{2.5}$$

where $F_{(x)i,int} = \dfrac{\partial \mathcal{L}}{\partial x_{i+2}}$, $F_{(y)i,int} = \dfrac{\partial \mathcal{L}}{\partial y_{i+2}}$ and $F_{(z)i,int} = \dfrac{\partial \mathcal{L}}{\partial z_{i+2}}$.

The internal force $F_{i,int}$ depends on its location, 2 points before and 2 points after.

2.2.1 System of ODEs

In order to reduce the derivatives of second order to first order, we introduce additional velocity variables, defined by, $v_{(x)i} = \ddot{x},\ v_{(y)i} = \ddot{y},\ v_{(z)i} = \ddot{z}$.

This means, now, we have a system of $6n-6$ equations, given as follows,

$$\begin{cases} v_{(x)i} = \dot{x}_{i+2} \\ v_{(y)i} = \dot{y}_{i+2} \\ v_{(z)i} = \dot{z}_{i+2} \\ m_i\dot{v}_{(x)i} - F_{(x)i,int} - F_{(x)i,ext} - F_{(x)i,damp} = 0 \\ m_i\dot{v}_{(y)i} - F_{(y)i,int} - F_{(y)i,ext} - F_{(y)i,damp} = 0 \quad for\ i = 1, ..., n-1 \\ m_i\dot{v}_{(z)i} - F_{(z)i,int} - F_{(z)i,ext} - F_{(z)i,damp} = 0 \end{cases} \tag{2.6}$$

2.2.2 Temporal Discretization

For the purpose of numerical stability, we use implicit method. Keeping this in mind, as well as in order to reduce the truncation error, we use the trapezoidal rule for the time stepping of our numerical simulation. Therefore, the previous equation (2.6) can be discretized as,

$$\begin{cases} x_{i+2}^{k+1} - x_{i+2}^{k} - (u_i^{k+1} + u_i^k)\frac{dt}{2} = 0 \\ y_{i+2}^{k+1} - y_{i+2}^{k} - (v_i^{k+1} + v_i^k)\frac{dt}{2} = 0 \\ z_{i+2}^{k+1} - z_{i+2}^{k} - (w_i^{k+1} + w_i^k)\frac{dt}{2} = 0 \\ m_i(v_{(x),i}^{k+1} + u_{(x),i}^{k}) - \frac{dt}{2}(F_{(x)i,int}^{k+1} + F_{(x)i,ext}^{k+1} + F_{(x)i,damp}^{k+1} + F_{(x)i,int}^{k} + F_{(x)i,ext}^{k} + F_{(x)i,damp}^{k}) = 0 \\ m_i(u_{(y),i}^{k+1} + u_{(y),i}^{k}) - \frac{dt}{2}(F_{(y)i,int}^{k+1} + F_{(y)i,ext}^{k+1} + F_{(y)i,damp}^{k+1} + F_{(y)i,int}^{k} + F_{(y)i,ext}^{k} + F_{(y)i,damp}^{k}) = 0 \\ m_i(u_{(z),i}^{k+1} + u_{(z),i}^{k}) - \frac{dt}{2}(F_{(z)i,int}^{k+1} + F_{(z)i,ext}^{k+1} + F_{(z)i,damp}^{k+1} + F_{(z)i,int}^{k} + F_{(z)i,ext}^{k} + F_{(z)i,damp}^{k}) = 0 \end{cases} \tag{2.7}$$

for $i = 1, ..., n-1$. Here dt corresponds to the time step and the index k denotes the time level, with $t^k = kdt$ Therefore, we have $(6n-6)$ nonlinear equations to be solved.

2.2.3 Jacobian Matrix

For the purpose of illustration, we define the vector-valued function **f(u)** to denote the system of equations (2.7). The Jacobian matrix is the matrix that consists of all the derivatives of **f** with respect to each corresponding components of the phase vector $\vec{\boldsymbol{u}}$. The Jacobian matrix is a block matrix, where most of the off diagonal entries are zero. The pattern of the matrix is illustrated in figure 2.4.

Here, we can notice that the matrix can be further simplified into four blocks. Beside the block at the bottom left, the entries in the three blocks are all diagonal. These 3 blocks

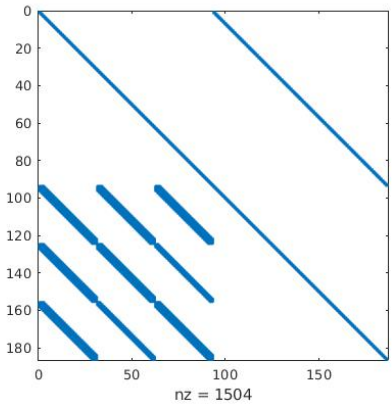


Figure 2.4: Pattern of the Jacobian matrix illustrating sparsity and block structure

have identity matrix entries which is evident if (considering its derivatives,) we look at the equations (2.7). Only the block at the bottom left consist of pentadiagonal structure. This is due to the evaluated derivatives of F_{int}, which not only depend on the corresponding location or its nearby points but also on the other axis in the plane of rotation.

2.3 Properties of the solver

Before going any further into validation of our model, we will first look at the plausibility of our solution. This is done by checking the condition number of the Jacobian to ensure that there is no likelihood of singularity. Then we will examine the computational efficiency of our solver for an increasing number of mass points.

Here, we use Matlab for our computation. This allows us to utilize the elegant feature of rapid prototyping programming techniques along with its inbuilt function resources. We also have the luxury of using the graphical tool for visualization, which helps us to comfortably analyze the data.

Please note that, in case of multiple numbers of operation, Matlab will have a considerable drawback regarding the memory consumption, consequently, slow execution time compared to the performance of other programming languages, such as, fortran. Nevertheless, we can use it as a building block to briefly investigate the behaviour of our mass-spring system.

2.3.1 Conditioning of the Jacobian

While performing matrix computation, we should be very careful if the matrix is stiff, i.e. ill-conditioned. Although, the matrix can be numerically non-invertible, we might still get an answer in Matlab. This is when the condition number comes into play. The condition number κ can be defined as the ratio of largest eigenvalue to the smallest eigenvalue of

a matrix. Since, the solution of our linear system of equations requires the computation of the Jacobian matrix, it is inescapable to make some analysis regarding its condition number.

No of Mass Points	Size of Jacobian Matrix	Condition number for $k_e = 50,000$	Condition number for $k_e = 100,000$	Condition number for $k_e = 200,000$
32	186×186	527,168.64	705,920.26	1,752,600.66
64	378×378	6,301,043.05	6,389,968.51	6,571,857.76
128	762×762	60,404,887.40	60,256,883.66	60,968,369.21
256	1530×1530	530,612,724.29	532,383,835.99	499,121,585.93
512	3066×3066	4,606,965,436.28	3,850,664,611.25	4,375,584,326.29

Table 2.1: Condition Number of the Jacobian for $k_e = 50,000$, $k_e = 100,000$ and $k_e = 200,000$ for different mass points

As mentioned above in table 2.1, we choose 3 different values of axial stiffness k_e. We can see in logarithmically scaled plot of the condition numbers for these 3 different stiffness values of our Jacobian matrices in the figure 2.5 below.

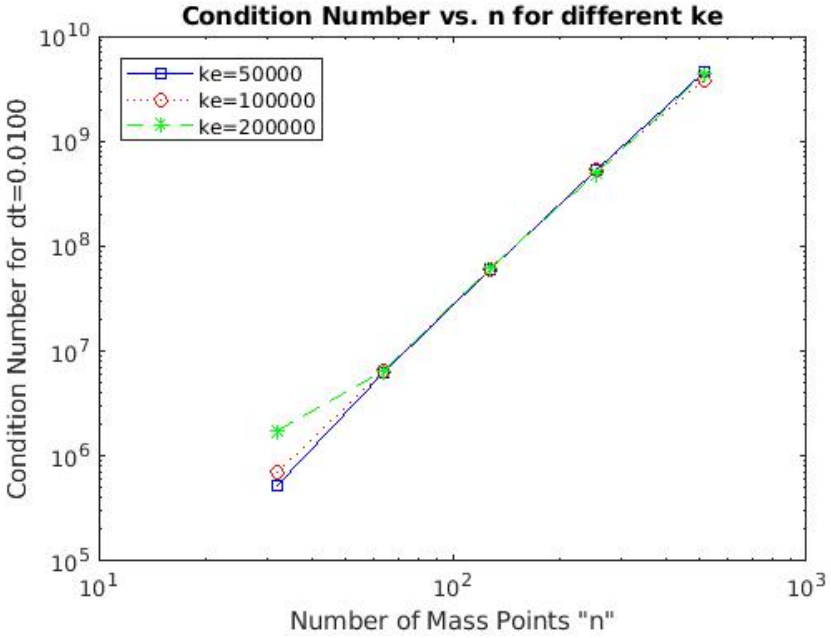


Figure 2.5: Condition number of the Jacobian matrix as a function of varying mass points for 3 different axial stiffness values.

When the condition number equals infinity, then the matrix is not invertible or singular.

Since, infinity is not meaningful in computing, therefore, in matlab, it is generally assumed that whenever the condition number κ is greater than or approaching 10^{15}, then the matrix is problematic or ill-conditioned.

As shown in the figure 2.5, we can observe that, for smaller number of mass points, the condition number increases for increasing stiffness values. However, for larger number of mass points, the condition numbers tends to be approximately similar (i.e. of the same order). This is because as one increases the number of mass points, the mass-spring segments tend to be more stiff. This can be verified by the relation $k = \dfrac{EA}{L_0}$ in case of linear elasticity.

Using linear fitting, we are able to find the power law relation i.e. $\kappa \propto Cn^{3.1}$ for axial stiffness $k_e = 10^5$ and a time step of $dt = 10^{-3}$. Note that for $n = 512$, we are already approaching the condition number of the order 10^9. Inevitably, we can extrapolate, for example, that we get a condition number $\kappa = 2.0141 \times 10^{13}$ as we increase the mass points to $n = 8192$.

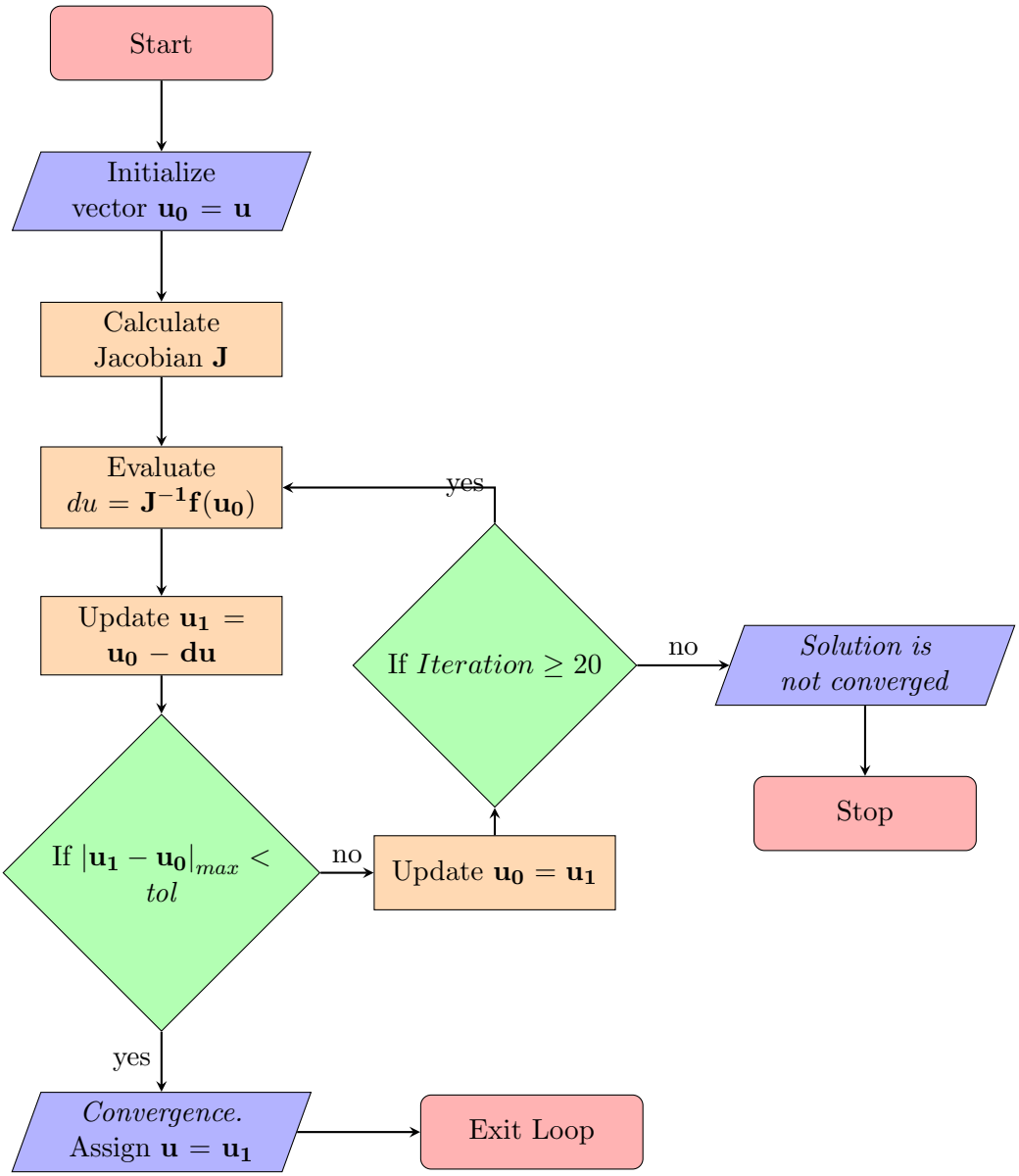
Using larger time steps, for example, $dt = 10^{-2}$, while using the same number of mass points implies much larger condition numbers. This is illustrated in the table 2.2

No of Mass Points	Size of Jacobian Matrix	Condition number for $k_e = 100,000$
32	186×186	1,156,286.06
64	378×378	9,035,572.09
128	762×762	70,029,681.81
256	1530×1530	714,462,652.81
512	3066×3066	14,672,886,774.30

Table 2.2: Condition number of the Jacobian for $k_e = 100,000$ and $dt = 0.01$ for a different number of mass points

For same number of mass points $n = 8192$, by doing extrapolation, we get the condition number κ to be 1.6507×10^{14}. Hence, it can be suggested for the user not to use more mass points beyond this value, especially, we have to be very careful when reducing the time steps. Also, the reason is that the block at the bottom left (see figure 2.4), has higher value than the whole Jacobian and it is always growing rapidly. Nevertheless, for the entire insect wing geometry, we just need the number of mass points within the range of 1024, for our mass-spring model.

2.3.2 Iterative Algorithm



Here we define the phase vector $\vec{u} = [\vec{x}\ \vec{y}\ \vec{z}\ \vec{u}\ \vec{v}\ \vec{w}]^T$ to include all the 6 unknowns with each having $n-1$ (unclamped) mass points. In order to solve the above $6n-6$ nonlinear system of equations (2.7) symbolized in the form of $\mathbf{f(u^{k+1})} = \mathbf{0}$, we use the Newton-Rhapson algorithm.

Finally we get the positions and velocities of each mass points at the new time steps t^{k+1}. The algorithm is illustrated by the flow chart above.

2.3.3 Computational Efficiency

In order to investigate the computational efficiency for our mass-spring model solver, we will briefly look at the solver time specified by the profile summary of our program. Then, we will also analyze the computational complexity using the CPU time evaluated as a function of the number of mass points in logarithmic scale.

In our mass-spring solver, we use the routine *CN_Implicit_straight_beam.m* as our main program. We can notice that the function *constructFja.m*, contributes to the most amount of execution time, because it calculates the Jacobian terms of internal forces $F_{(x)i,int}$, $F_{(y)i,int}$ and $F_{(z)i,int}$ or in other words, the derivatives of the equations (A.2), (A.3) and (A.4). However, this is not the case for larger number of mass points, which can be seen in the 2.3 listed below. This is because, as we increase the number of mass points, the main routine *CN_Implicit_straight_beam.m*, which contains the variable for convergence criteria, uses the inversion of the Jacobian matrix. Hence, this will consume the major amount of operation time as we increase the number of points.

We will now look at the computational complexity i.e. the CPU time for different cases of mass points. We will also compare the CPU time for a sparsed-solver and a non-sparsed solver, (i.e. when we skip the sparse storage organization) for our *Jacobian* matrix. This is shown in the table 2.3 below.

Computational Complexity		
Number of Mass Points	CPU Time [s] (sparsed solver)	CPU Time [s] (non-sparsed solver)
8	32.514	30.042
16	37.006	36.388
32	46.888	50.297
64	81.209	104.936
128	216.240	359.969
256	1142.829	2014.8
512	3511.771	10166.194
1024	16337.062	53284.245

Table 2.3: CPU time for number of mass points of the mass-spring model.

It is evident that these complexity plots figure 2.6 show a nonlinear behavior. Please note that these plots in figures 2.6 are in logarithmic scale.

Like we mentioned in the table 2.1, we can also visually notice the contrast of CPU time in figure 2.6, once we remove the sparse storage for our Jacobian. The complexity is of higher order polynomial for the non-sparsed matrix case. This result verifies the advantage of using the *Sparse function* for efficient storage and solving linear systems in Matlab.

It is clearly noticeable that these nonlinear logarithmic plots of the computational complexity show an approximately linear behaviour for $n = 128$ or above. This is because at smaller number of mass points, the function calls in our program also contribute to the investment of our CPU time. This also means, the CPU time for larger number of mass

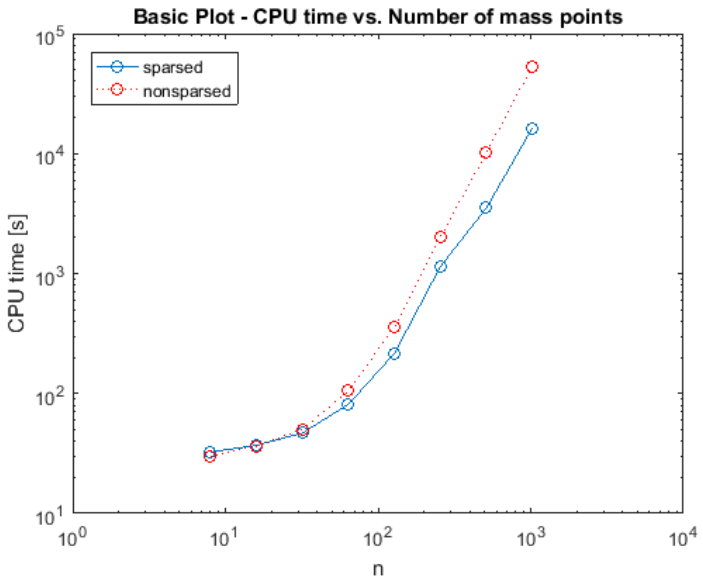


Figure 2.6: Complexity comparison for sparsed and non-sparsed Jacobian matrix

points, is mostly dominated by the computation of system of equations.

If we extrapolate using the linear fitting or the power law relation, (i.e. the Complexity $\propto Cn^2$) we can estimate, for example, the CPU time for $n = 4096$ to be approximately 205990 seconds or 57.2204 hours (i.e. 2 days 9 hours and 15 minutes). For the non-sparsed solver, we get the CPU time to be approximately 156390 seconds or 434.41 hours (i.e. 18 days 2 hours and 30 minutes).

Consequently, this means our model is not computationally efficient. We have to be really aware of CPU time while coupling this solid solver with the fluid solver. Nevertheless, less number of mass points are sufficient to be able to represent the overall insect wing.

2.4 Validation

It is essential to perform a validation of our mass-spring model. We will do this by analyzing the solution for our model with the collection of data from the nonlinear beam model mentioned in [5]. We will, then check the sensitivity of our mass-spring model with respect to the number of grid points. This is done by making some comparison with an increasing number of mass points.

2.4.1 Convergence

Since the nonlinear beam is only chord-wise flexible (i.e. 1D flexibilty), we set the gravity load in z-direction to be $g_z = 0$, for our mass spring solver. For the given material properties of mass $m = 0.0571L/n$, flexural stiffness $EI = 0.0259$ and axial stiffness $k_e = 10^3$, we

compare the deflection of our mass-spring model with the nonlinear beam solution from Thomas Engels [5]. We found that the absolute relative error is 0.01% in the y direction at the free end of the beam.

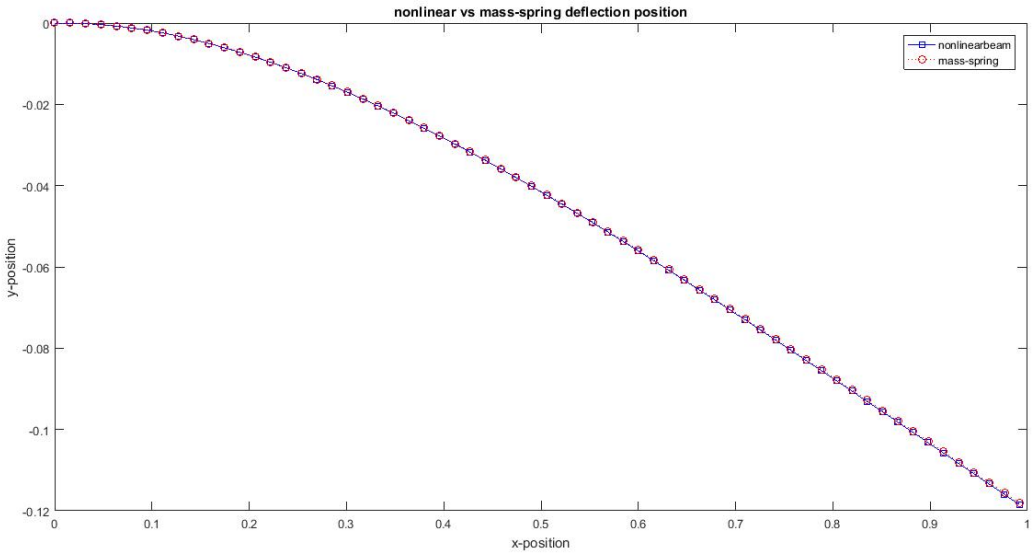


Figure 2.7: Comparison of the deflection between the mass-spring model and the nonlinear beam model

Chapter 3

Fluid Solver

The governing equations to describe a fluid flow is given by the Navier-Stokes equations. Here we discuss the techniques that are used to solve the incompressible Navier-Stokes equation numerically. The difficulty lies in satisfying the no-slip boundary condition on a complex geometry with moving solid obstacle and also to ensure the divergence-free condition.

Thus we use volume penalization method, which uses the immersed boundary approach, which includes the interior of the solid obstacles as computational domain. Therefore, the model equation are given as,

$$\partial_t u + \omega \times u + \nabla \Pi - \nu \nabla^2 u = 0 \tag{3.1}$$

$$\nabla \cdot u = 0 \tag{3.2}$$

where, u is the velocity, $\omega = \nabla \times u$ is the vorticity and $\Pi = p + \frac{1}{2} u \cdot u$ is the total pressure. The system is completed with initial conditions $u(x, 0) = u_0$ (in the fluid domain Ω_f). and $u = u_S$ (at the boundary $\partial\Omega$), where u_S is the solid velocity.

3.1 Volume Penalization Method

The idea of Volume Penalization Method is to replace the solid obstacle by a porous medium with small permeability. The domain definition of the problem is demonstrated using the simplified figure 3.1.

The model equation can be, thus, expressed as Penalized Navier-Stokes Equation,

$$\partial_t u + \omega \times u = -\nabla \Pi + \nu \nabla^2 u - \frac{\chi}{C_\eta}(u - u_S) \tag{3.3}$$

having an additional forcing term. With an initial condition,

$$u(x, t = 0) = u_0 \tag{3.4}$$

The geometrical information is stored by the mask function, where,

$$\chi(x_i, t) = \begin{cases} 0, & \text{if } x \in \Omega_f. \\ 1, & \text{if } x \in \Omega_s. \end{cases} \tag{3.5}$$

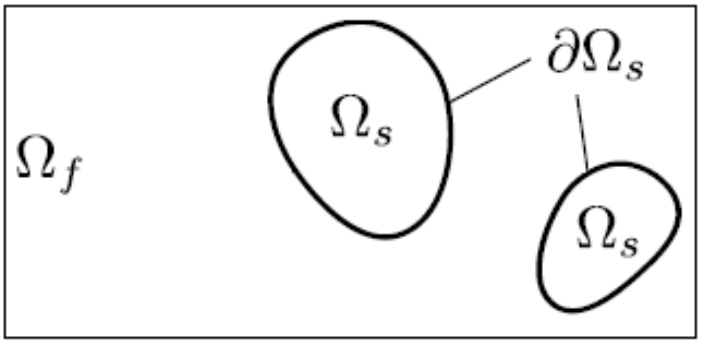


Figure 3.1: The computational domain $\Omega = \Omega_f \cup \Omega_s$

For $C_\eta \to 0$ the solution of the penalized Navier–Stokes equations (3.3)-(3.5) approach towards the exact solution of the Navier–Stokes equation [1], [4]. This means we model the above equations (3.3)-(3.5) using the penalization term (in the RHS of the equation (3.3)) in order to satisfy the no-slip boundary conditions.

To fulfill the incompressibility constraint equation (3.2), we take the divergence of equation (3.3). Thus, we obtain a Poisson equation for the pressure as,

$$\nabla^2 \Pi = \nabla \cdot \left(-\omega \times u - \frac{\chi}{C_\eta}(u - u_S) \right) \tag{3.6}$$

If we disregard the pressure, viscous and the nonlinear terms C_η can be directly interpreted as the relaxation time [6]. The reason for chosing this formulation has to do with its conservation properties, particularly for momentum and energy (for more, see [11],pp. 210).

3.2 Spectral method

Since, we are dealing with relatively low Reynolds number flow for our insect wing, we use direct numerical simulation (i.e. without Turbulence Modeling). So, the obvious choice for domain discretization is the Spectral Method.

The idea of Spectral methods is to represent the solution and the spatial derivatives using a sum of certain trial or basis functions, for example, a generalized Fourier Series or sometimes Chebyshev polynomials, Legendre polynomials etc and then to determine the coefficients in the sum to satisfy the differential equations as much as possible.

For example, a field variable can be expressed in terms of the Fourier approximation,

$$u(\vec{x}, t) = \sum_k \hat{u}_k(\vec{k}, t) e^{ikx} \tag{3.7}$$

where, $\hat{u}_k$ is the Fourier coefficient,

$$\hat{u}_k(\vec{k},t) = \frac{1}{2\pi}\int_0^{2\pi} u(\vec{x},t)e^{-ikx}dx \tag{3.8}$$

Since, this is in continuous setting, we have trouble representing in computer. Thus, we take a finite number of nodes for truncation. Therefore, we use the discrete Fourier series and then compute our coefficients,

$$\hat{u}_k = \frac{1}{N}\sum_{n=0}^{N-1} u\left(\frac{n2\pi}{N}\right)e^{-ik2\pi kn/N} \tag{3.9}$$

With $\vec{k} = (k_x, k_y)$. The discretization is uniform in space and is truncated at $k_x = -N_x/2$, $k_x = N_x/2 + 1$ and $k_y = -N_y/2$, $k_y = N_y/2 + 1$. Therefore the discrete Fourier Series representation for the equation (3.7) is,

$$u_N(x) = \sum_{k=-N/2}^{N/2-1} \hat{u}_k e^{2\pi ikx} \tag{3.10}$$

The advantage of this representation is that, due to its trigonometric or exponential form, the basis function remains the same (or simply alternates in trigonometric form). The gradient of u is computed by multiplying the $\hat{u}$ by $i\vec{k}$.

Therefore, this will end up in a simple algebraic multiplication. Using Fast Fourier Tranform we can move between both domains. To sum up, Spectral Methods solve differential equations (like the Navier Stokes Equations) as a Fourier polynomial approximation with the help of the Fast Fourier Transform (FFT).

To solve the penalized Navier–Stokes equations we employ the classical Fourier pseudospectral method, which is very precise for flows with periodic boundary condition [Canuto et al.,1988]

3.2.1 Fourier Pseudo-Spectral Discretization

The pseudo-spectral method is special technique of discretizing the nonlinear terms in the spectral form of the Navier–Stokes momentum equation (i.e. $\hat{N}(\omega)$), where,

$$N(\omega) = -\omega \times u - \frac{\chi}{C_\eta}(u - u_S) \tag{3.11}$$

This term is computed by the pseudo-spectral method, which is primarily done by not evaluating the convolution directly. And the velocity induced by vorticity can, thus, be represented as,

$$u(x,t) = -\sum_k \frac{ik^\perp}{k^2}\widehat{\omega}_k(t)e^{ikx} + u_\infty \tag{3.12}$$

Here, the convection term and the penalization term, which contain products are computed using collocation in physical space using inverse FFT operation and then, subjected to FFT for calculation in Fourier space. This is done in order to reduce the computational time taken by the convolution operation in Fourier space.

The product of the Fourier transforms in the nonlinear expression will yield non-resolved terms. This will produce errors, which are called aliasing errors. They are removed by de-aliasing, which corresponds to the truncation of the corresponding wave numbers. For this, 2/3 rule is applied by discarding all Fourier coefficients at each time step for which,

$$k > \frac{2}{3} k_{max} \tag{3.13}$$

The 2/3 rule for vorticity can be summerized as,

$$\hat{\omega}(\vec{k}) = \begin{cases} \hat{\omega}(\vec{k}) & \text{for } \left(\frac{3k_x}{2N_x}\right)^2 + \left(\frac{3k_y}{2N_y}\right)^2 < 1. \\ 0 & \text{for } \left(\frac{3k_x}{2N_x}\right)^2 + \left(\frac{3k_y}{2N_y}\right)^2 > 1. \end{cases} \tag{3.14}$$

For details we refer to the book of Canuto et al. [3].

3.2.2 Time Discretization

For our model, here, we use the classical 2nd order Runge-Kutta time stepping scheme, which can be given by,

$$u^{n+1} = u^n + (a_1 k_1 + a_2 k_2)\Delta t \tag{3.15}$$

where, the coefficients $a_1 = 1/2$ and $a_2 = 1/2$ yield,

$$u^{n+1} = u^n + (\frac{1}{2} k_1 + \frac{1}{2} k_2)\Delta t \tag{3.16}$$

$$with, \qquad k_1 = f(t^n, u^n) \tag{3.17}$$

$$k_2 = f(t^n + \Delta t, u^n + \Delta t) \tag{3.18}$$

Note this is an exclusively explicit scheme. Other semi-implicit schemes like the Adams-Bashforths method [13] have also been used.

3.3 Hydrodynamic forces

For the volume penalization method, an explicit way to determine the forces F (i.e. lift and drag) on the body immersed in a fluid is used. It is proven in [1] that, instead of computing the surface integral of the stress tensor, (for a fixed obstacles, i.e. $u_S = 0$,) we have,

$$F = \int_{\partial\Omega_S} \sigma \mathbf{n} ds = -\lim_{C_\eta \to 0} \frac{1}{C_\eta} \int_{\Omega_S} \chi \mathbf{u} dV \tag{3.19}$$

where, σ is the fluid stress tensor, $\mathbf{n}$ is the outward unit normal vector of the surface $\partial\Omega_S$. Please note that the penalized velocity for moving obstacle includes the solid velocity, i.e. $u_S \neq 0$ and the unsteady correction term [9]. Therefore,

$$F = -\lim_{C_\eta \to 0} \frac{1}{C_\eta} \int_{\Omega_S} \chi(\mathbf{u} - \mathbf{u_S}) dV + \frac{d}{dt} \int_{\Omega_S} \mathbf{u_S} dV \tag{3.20}$$

for the moving obstacle. This formula will be used in chapter 4.

3.4 Validation study: Flow past circular cylinder

In order to compare the solution of the incompressible Navier–Stokes equations, we use the Turek Benchmark computations for 2D laminar flow around a cylinder. For detailed evaluation and comparison with a reference experiment the results are given in [12].

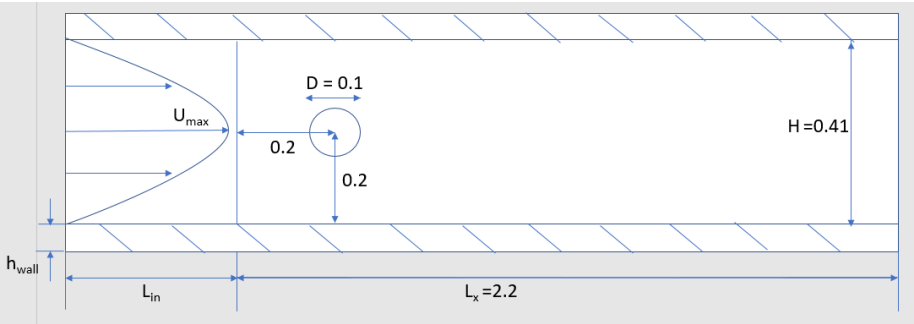


Figure 3.2: Geometrical setup for flow past a circular cylinder

To avoid the periodicity, we have extended the computational domain so that we can insert the so-called feeding inflow zone. Thus we can impose a Poiseuille velocity profile, which is necessary for the benchmark.

To begin, we investigate the forces and coefficients for a circular cylinder with 3 different grid resolutions and compare the velocity profile at the inlet. In the end, we compare the drag coefficients for some more grid resolutions.

3.4.1 Results

The input parameters and the dimensions are specified below in the table 3.1, where, Level $1 = 900 \times 180$, Level $2 = 1000 \times 200$ and Level $3 = 1100 \times 220$. Note that the feeding inflow length is $L_{in} = 0.7520$, which is added to a comparable domain length in x direction, i.e., exactly as the domain length in x direction of the reference Turek benchmark.

Resolution	Parameters	Values
Level 1	n_x	900
	n_y	180
Level 2	n_x	1000
	n_y	200
Level 3	n_x	1100
	n_y	220

Table 3.1: Number of grids for three different resolution level

Parameters	Symbols	Units	Values
Inlet velocity	U_∞	$\frac{m}{s}$	0.694444
Mean flow velocity	$\bar{U}$	$\frac{m}{s}$	1.0
Maximum time level	T_{max}	s	15.0
Domain length in 'x'	L_x	m	2.9520
Domain length in 'y'	L_y	m	0.5904
Fluid kinematic viscosity	ν	$\frac{m^2}{s}$	0.001
Fluid Density	ρ^f	$\frac{kg}{m^3}$	1.0
Reynolds number	Re	-	100
CFL number	CFL	-	0.25
Permeability	C_η	-	0.001
Thickness of the wall	h_{wall}	m	0.0902
Diameter of the cylinder	d_{cyl}	m	0.1
'x' position of the cylinder	x_{0c}	m	0.9520
'y' position of the cylinder	y_{0c}	m	0.2902

Table 3.2: Input parameters for the flow past a circular cylinder

From table 3.3, we can clearly notice that the lower and upper bound of the maximum lift and drag coefficients are within the range found in the benchmark reference (See [12] table 4), i.e. 2.8920 to 4.7330 for maximum drag coefficients and 0.5658 to 2.0600 for maximum lift coefficients. Please note that we have used the same scale magnitude for forces and coefficients to see the variation. The vorticity field for 3 level of grid resolutions, shown in figure(3.3)-(3.5), are similar as expected. We can also observe that the velocity profile is parabolic with the maximum value approaching $U_{max} = 1.5$ for each 3 cases (see figure (3.5)).

In table 3.4, we can, particularly, observe that, as we increase the grid resolution from 400×80 to 1100×220, the drag coefficients are sinusoidally damping to a value of $C_d = 3.27$, which is near the frequently occurring value of drag coefficient in the benchmark, i.e. $C_d = 3.22$. The C_d value depends on the drag force F_d, which inturn depends on mask function χ. Since the mask function χ is discontinuous, we need a smoothening layer to get the resulting drag coefficient value refined. Also, we should note that if we increase the grid resolution further, we should be aware of the relation with the permeability value, i.e. $C_\eta \propto (\Delta x^2)$.

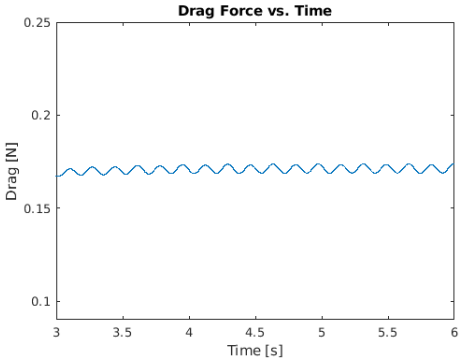
Output	Level 1	Level 2	Level 3
U_{max} $[\frac{m}{s}]$	1.4997	1.4994	1.5
$\bar{F}_d$ $[N]$	0.169184	0.162821	0.163524
F_{dmax} $[N]$	0.173800	0.166831	0.167472
F_{dmin} $[N]$	0.154820	0.150131	0.151064
$\bar{C}_d$	3.383686	3.256428	3.270474
C_{dmax}	3.475995	3.336625	3.349443
C_{dmin}	3.096412	3.002624	3.021291
$\bar{F}_l$ $[N]$	-0.000698	-0.000379	-0.000299
F_{lmax} $[N]$	0.058586	0.052441	0.052298
F_{lmin} $[N]$	-0.059995	-0.053593	-0.053362
$\bar{C}_l$	-0.013952	-0.007580	-0.005982
C_{lmax}	1.171720	1.048816	1.045971
C_{lmin}	-1.199907	-1.071858	-1.067240

Table 3.3: Results for the flow past circular cylinder with Level 1 = 900×180, Level 2 = 1000×200 and Level 3 = 1100×220

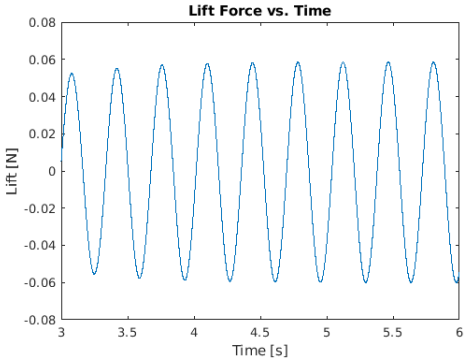
nx	400	600	800	900	1000	1100
ny	80	120	160	180	200	220
$\bar{F}_d$ $[N]$	0.168080	0.173429	0.166651	0.169184	0.162821	0.163524
$\bar{C}_d$	3.361607	3.468580	3.333011	3.383686	3.256428	3.270474

Table 3.4: Mean drag and mean drag coefficient for the flow past a circular cylinder with increasing grid resolution

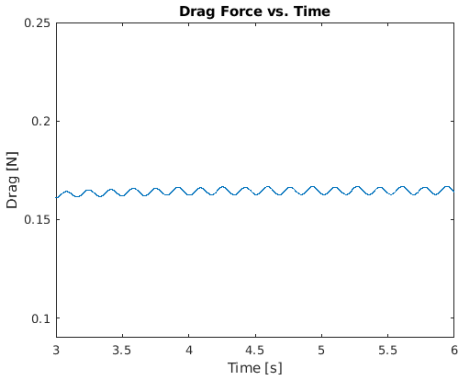
Please note that the drag parameters are averaged over the time interval between $t = 0.2s$ and $t = 15s$, so that we exclude the impulsive oscillations at the beginning.



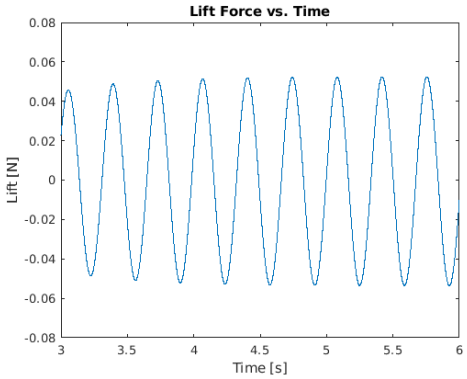
Level 1: Drag as a function of time



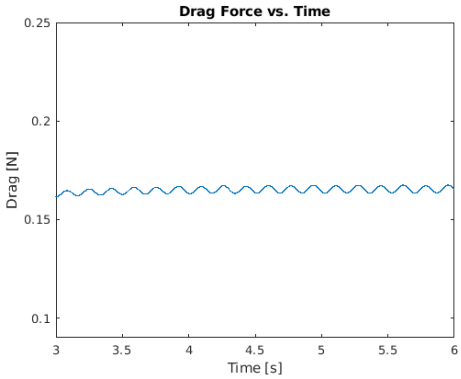
Level 1: Lift as a function of time



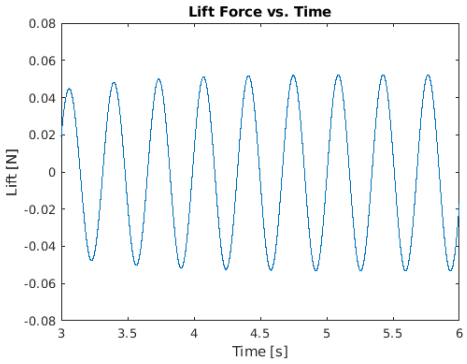
Level 2: Drag as a function of time



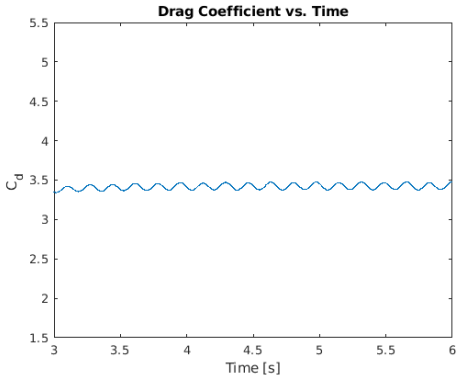
Level 2: Lift as a function of time



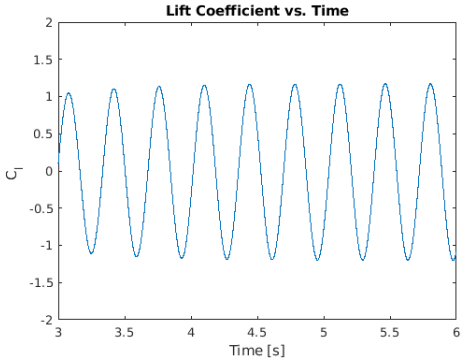
Level 3: Drag as a function of time



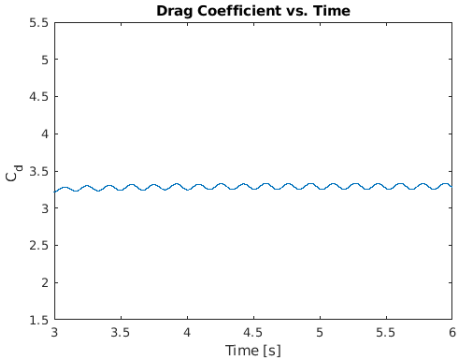
Level 3: Lift as a function of time



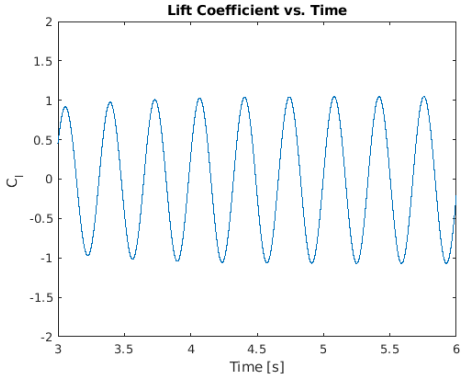
Level 1: C_d as a function of time



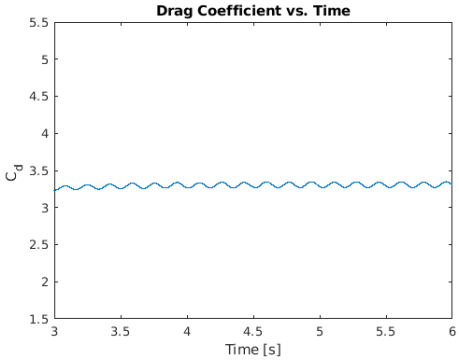
Level 1: C_l as a function of time



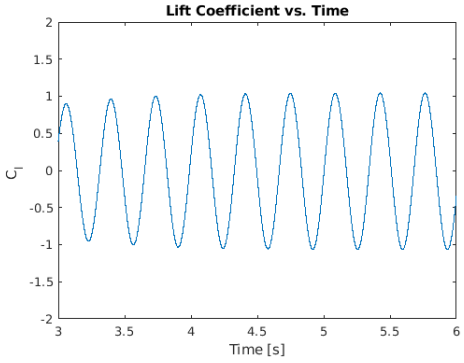
Level 2: C_d as a function of time



Level 2: C_l as a function of time



Level 3: C_d as a function of time



Level 3: C_l as a function of time

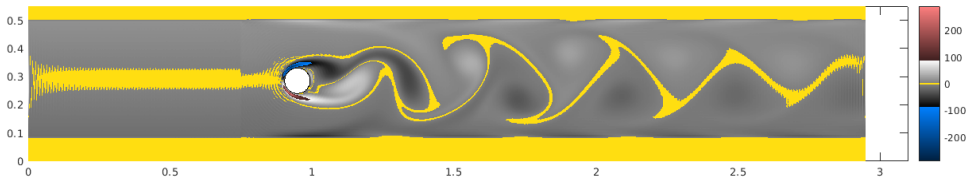


Figure 3.3: Level 1: Vorticity field after 15 seconds

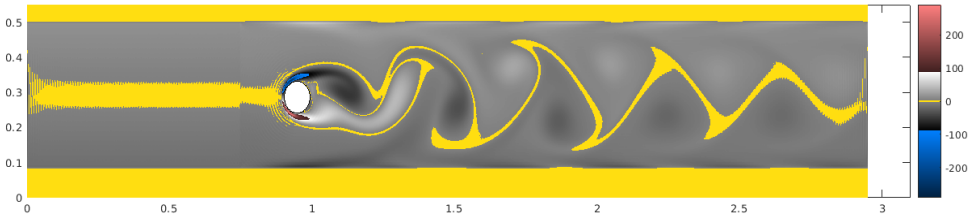


Figure 3.4: Level 2: Vorticity field after 15 seconds

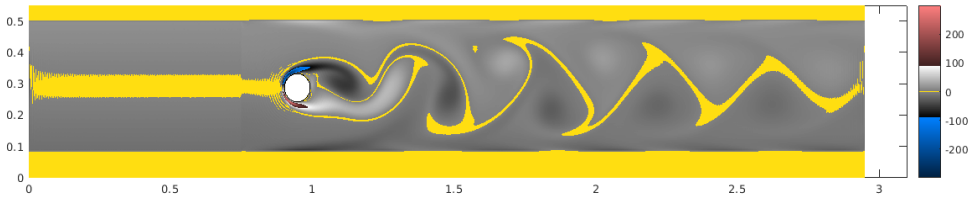


Figure 3.5: Level 3: Vorticity field after 15 seconds

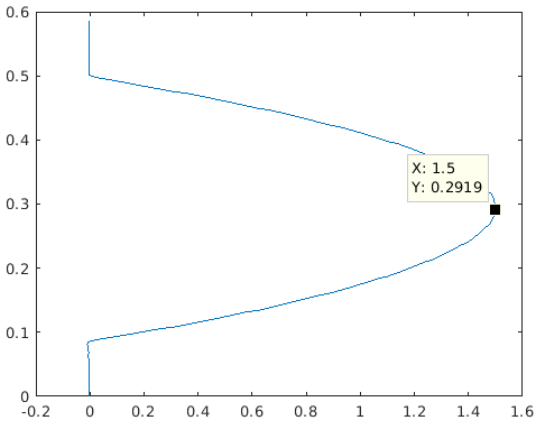


Figure 3.6: Level 1: Velocity profile at the inlet

Using the lift coefficient curve, we can check the period of vortex shedding, and thus the frequency f. Using this, we can calculate the Strouhal number by the relation,

$$St = \frac{fD}{\overline{u}} \tag{3.21}$$

We found that the Strouhal number to be $St = 0.2703$ for 900×180, which is within the range of 5.8% near the reference value of $St = 0.287$ in [12]. This means that we can now reliably enhance our work and move to fluid structure interaction.

Chapter 4

Fluid-Structure Interaction

Insect wings are shape morphing and compliant by nature. Its complexity is also characterized by a cross-section with corrugated configuration. Moreover, flapping motion and deforming phenomena are omnipresent in insect flight.

In this chapter we will look at how the fluid solver and the solid solver are coupled to obtain a more realistic and oscillatory reponse of a simplified vein of an insect wing. We use the Turek benchmark for Fluid-Structure Interaction (FSI) [14] to validate our coupled solver.

4.0.1 Mask function for moving and flexible solid obstacles

In order to describe a solid obstacle inside a fluid domain, we directly use the mask function according to the equation (3.5). However, large oscillations can be found in hydrodynamic forces for moving obstacles using a Fourier pseudo-spectral method [9]. A smoothened mask function is instead used, which relies on the signed distance function in order to capture the translation of the obstacle [5].

$$\chi(x,t) = \begin{cases} 0, & \delta \leq -h. \\ \frac{1}{2}(1+\cos(\pi\frac{\delta+h}{h})), & -h \leq \delta \leq +h. \\ 1, & \delta \geq +h. \end{cases} \tag{4.1}$$

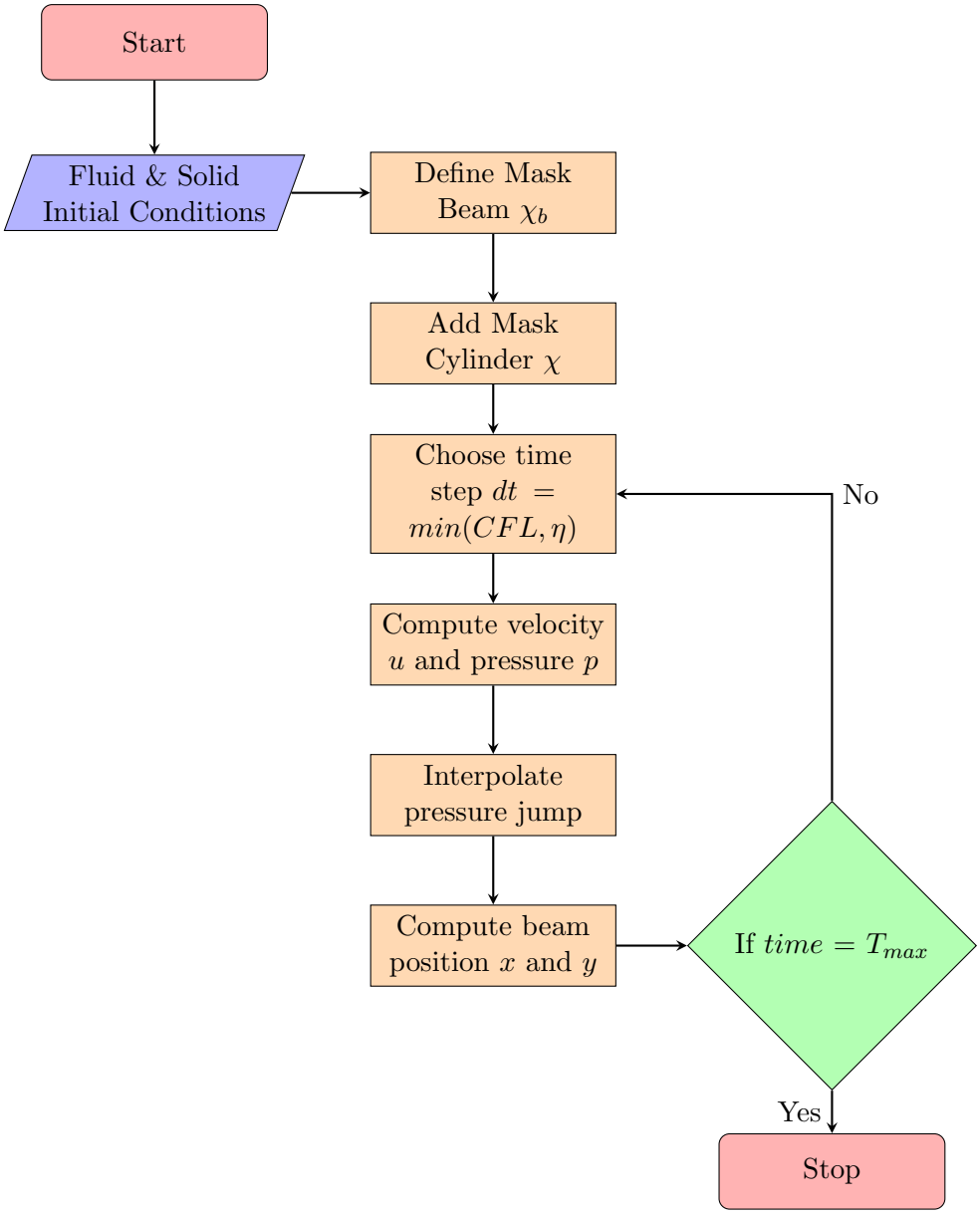
where, $\delta(x,t) = \|x - x_0(t)\| - R_0$, R_0 is the radius centered around x_0 and h is the half thickness of the smoothing layer. The external or the fluid forces are one of the forces that influence the solid deformation. Since, the division of the segments of solid obstacles is independent of the entire domain discretization, we use pressure interpolation to evaluate the pressure jump across the beam using,

$$[p]^{\pm} = \int_{r=b_{bot}(s)}^{r=b_{top}(s)} (p_{top}(s) - p_{bottom}(s)) \tag{4.2}$$

where, s is the chordwise position of the deformed mean. The function b_{bot} and b_{top} takes into account of non-rectangular shapes. For more details, see [5] [10].

4.1 Coupling Algorithm

The coupling between the fluid and structure is attained by the construction of the mask function χ and solid velocity field u_S, and the interpolation of the pressure on the solid surface. This is described by a simple flow chart below.



4.2 Turek Benchmark

We are now in a position to study the Turek benchmark for fluid-structure interaction. The domain dimension in x-direction, is slightly increased as we have seen for the circular cylinder in figure (3.2). A flexible beam of length $L_b = 0.35$m has been added with a thickness of $t_b = 0.02$m. Note that the feeding inflow length here is $L_{in} = 0.1771$m.

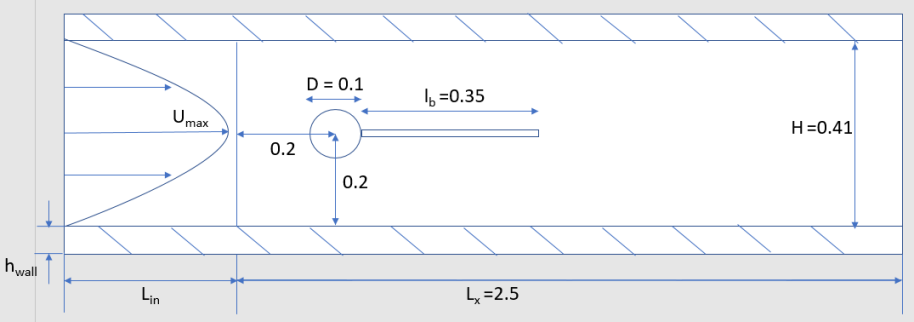


Figure 4.1: Geometrical setup for a circular cylinder with a beam attached for the FSI simulation

4.2.1 Results

The input parameters for a flow past a circular cylinder with a flexible beam at $Re = 200$, i.e. based on Turek's benchmark for FSI3 [15], can be summarized in the below table 4.1.

Parameters	Symbols	Units	Values
Inlet velocity	U_∞	$\frac{m}{s}$	0.694444
Mean flow velocity	$\bar{U}$	$\frac{m}{s}$	2.0
Maximum time level	T_{max}	s	20.0
Domain length in 'x'	L_x	m	2.6771
Domain length in 'y'	L_y	m	0.5904
Fluid kinematic viscosity	ν	$\frac{m^2}{s}$	0.001
Fluid Density	ρ^f	$\frac{kg}{m^3}$	1000
Reynolds number	Re	-	200
CFL number	CFL	-	0.1
Permeability	C_η	-	0.001

Thickness of the wall	h_{wall}	m	0.0902
Diameter of the cylinder	d_{cyl}	m	0.1
'x' position of the cylinder center	x_{0c}	m	0.3771
'y' position of the cylinder center	y_{0c}	m	0.2902
Position of the beam in 'x'	x_0	m	0.4271
Length of the beam	L_b	m	0.35
Thickness of the beam	t_b	m	0.02

Table 4.1: Input parameters for the flow past a circular cylinder attached with a flexible beam

From table (4.2), we can clearly see that the upper and lower bound of both the minimum and maximum drag and lift forces for level 2 are more reasonable than level 1 with respect to the range specified in the benchmark. When the time level is above 3.5 seconds (see figure (4.2) and (4.3)), level 2 shows more accuracy, when we compared to the benchmark [14], i.e. for example the maximum drag is 487.81 and the minimum drag is 432.79. The minimum and maximum displacement in the benchmark computations is between -0.03345 and 0.03637, which seems plausible even with such low resolution, though minimum displacement for level 1 does not agree for the entire time interval we considered. (Note that the time interval that we choose to calculate the maximum value is between $t = 1.1$ seconds to $t = 4.375$ seconds, i.e. the t_{end}. Also the grid resolution in the benchmark is finer compared to our test case).

The frequency of oscillation of the beam is calculated graphically using the displacement curve, using the relation with period T, i.e. $f = 1/T$. We can notice that for level 2, the frequency is 5.555 Herz which is almost similar compared with the benchmark, i.e. 5.47 Herz.

Output	Level 1	Level 2
F_{dmax} $[N]$	1107.2131	718.9604
F_{dmin} $[N]$	429.7072	356.4436
F_{lmax} $[N]$	505.6752	276.0013
F_{lmin} $[N]$	-542.5533	-329.7791
y_{dmax} $[m]$	0.032246	0.039913
y_{dmin} $[m]$	-0.014062	-0.034417

Table 4.2: Results for the flow past circular cylinder attached with a flexible beam with Level 1 $= 325 \times 72$ with $n_s = 42$ and Level 2 $= 650 \times 144$ with $n_s = 85$

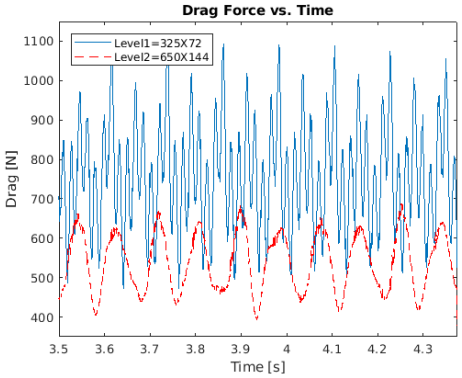


Figure 4.2: Drag as a function of time

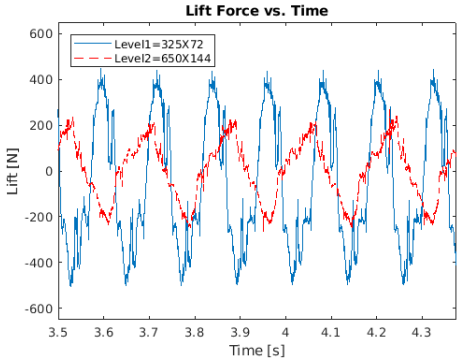


Figure 4.3: Lift as a function of time

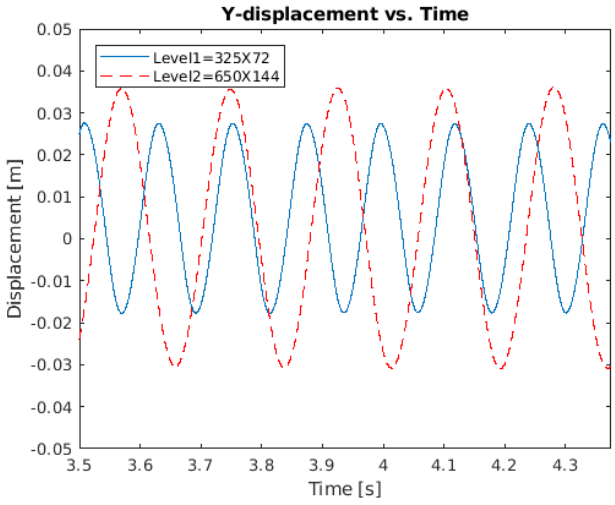


Figure 4.4: Displacement as a function of time FSI simulation

Figure 4.5: Vorticity field of FSI simulation for level 2 at 4.37 seconds

Note that the color table for figure (4.5) is the same as in figure (3.3).

Chapter 5

Conclusion

Numerical simulation of a flexible beam representing a simplified wing was the core of this work. The beam is immersed in a viscous fluid and is subjected to external forces exerted by the fluid, while the flow field additionally experiences feedback from the solid at each instance in time. The subject is studied in three different chapters with an aim of understanding the fluid-structure interaction.

Followed by an introduction and motivation in the first chapter, we studied the properties of a mass-spring model, particularly, the condition number of the Jacobian matrix in Chapter 2. We found that the condition number κ increases for increasing number of mass points. Interestingly, κ increases if the axial stiffness value is at the smaller number of mass points, while it tends to be approximately similar for larger number of mass points. We also studied the computational complexity of the mass-spring model and realized the benefit of using the sparse solver in Matlab. Although the complexity C is found to be nonlinear with $C \propto n^2$, fortunately, it is sufficient to model the insect wing with limited number of mass points to obtain reasonable accuracy. Using the power law relation for the CPU time, we extrapolated the CPU time for larger number of mass points. We also found the deflection at the free end to be in good accuracy compared to nonlinear beam model.

In Chapter 3, we used the volume penalization method with spectral method's discretization for flow simulation. We performed a benchmark test for a flow past a circular cylinder with channel walls. We obtained a reasonable accuracy for lift and drag coefficients. The results of drag coefficients, are sinusoidally converging to a mean value of $C_d = 3.27$, as we increased the number of grid points. We also looked at the Strouhal number and found it to be within the range of 5.8 % with respect to reference computations. Finally, in Chapter 4, we studied the fluid-structure interaction and validated the Turek benchmark for FSI3. We found that the maximum drag and lift for resolution level 2, i.e. 650×144 to be near the values provided in the benchmark. The frequency of oscillation for level 2 is almost similar to the benchmark. And the displacements value for both levels 1 and 2 are near the range found in the benchmark for fluid structure interaction.

In this fascinating field of bio-inspired propulsion, fluid-structure interaction of the flexible wing is really a challenging and interdisciplinary area. There are many open questions. It will be interesting to study the two dimensional flexibility in a three dimensional fluid domain and the future work is associated with computations with better resolution.

Appendix A

Lagrangian Formulation

A.0.1 Kinetic Energy

We get the term for **Kinetic Energy** as:

$$\frac{d}{dt}\Big(\frac{\partial \mathcal{L}}{\partial \dot{x}_{i+2}}\Big) = m_i \ddot{x}_{i+2}; \frac{d}{dt}\Big(\frac{\partial \mathcal{L}}{\partial \dot{y}_{i+2}}\Big) = m_i \ddot{y}_{i+2}; \frac{d}{dt}\Big(\frac{\partial \mathcal{L}}{\partial \dot{z}_{i+2}}\Big) = m_i \ddot{z}_{i+2}; \tag{A.1}$$

A.0.2 Potential Energy

And the term for **Potential Energy** is given as:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial x_{i+2}} =& - k_{(e)i}(l_i - l_{0,i})\frac{x_{i+2} - x_{i+1}}{l_i} + k_{(e)i+1}(l_{i+1} - l_{0,i+1})\frac{x_{i+3} - x_{i+2}}{l_i + 1} \\
&+ k_{(b)i}(\theta_i - \theta_{i-1} - \theta_{0,i} + \theta_{0,i-1})\frac{y_{i+2} - y_{i+1}}{(x_{i+2} - x_{i+1})^2 + (y_{i+2} - y_{i+1})^2} \\
&- k_{(b)i+1}(\theta_{i+1} - \theta_i - \theta_{0,i+1} + \theta_{0,i})\Bigg(\frac{y_{i+3} - y_{i+2}}{(x_{i+3} - x_{i+2})^2 + (y_{i+3} - y_{i+2})^2} \\
&+ \frac{y_{i+2} - y_{i+1}}{(x_{i+2} - x_{i+1})^2 + (y_{i+2} - y_{i+1})^2}\Bigg) \\
&+ k_{(b)i+2}(\theta_{i+2} - \theta_{i+1} - \theta_{0,i+2} + \theta_{0,i+1})\frac{y_{i+3} - y_{i+2}}{(x_{i+3} - x_{i+2})^2 + (y_{i+3} - y_{i+2})^2} \\
&+ k_{(b)i}(\phi_i - \phi_{i-1} - \phi_{0,i} + \phi_{0,i-1})\frac{z_{i+2} - z_{i+1}}{(x_{i+2} - x_{i+1})^2 + (z_{i+2} - z_{i+1})^2} \\
&- k_{(b)i+1}(\phi_{i+1} - \phi_i - \phi_{0,i+1} + \phi_{0,i})\Bigg(\frac{z_{i+3} - z_{i+2}}{(x_{i+3} - x_{i+2})^2 + (z_{i+3} - z_{i+2})^2} \\
&+ \frac{z_{i+2} - z_{i+1}}{(x_{i+2} - x_{i+1})^2 + (z_{i+2} - z_{i+1})^2}\Bigg) \\
&+ k_{(b)i+2}(\phi_{i+2} - \phi_{i+1} - \phi_{0,i+2} + \phi_{0,i+1})\frac{z_{i+3} - z_{i+2}}{(x_{i+3} - x_{i+2})^2 + (z_{i+3} - z_{i+2})^2}
\end{aligned} \tag{A.2}$$

Please note that we have 8 terms for arbitrary $x_k (= x_{i+2})$. However, at the boundary we have no $k_{(e)n}$ and also $k_{(b)n} = 0$, which means that

- $\dfrac{\partial \mathcal{L}}{\partial x_n}$ has 6 terms, because 2 terms associated with θ and ϕ, will disappear.
- $\dfrac{\partial \mathcal{L}}{\partial x_{n+1}}$ has 3 terms, because 1 term associated with l and 4 terms associated with θ and ϕ, will disappear.

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial y_{i+2}} =& - k_{(e)i}(l_i - l_{0,i})\frac{y_{i+2} - y_{i+1}}{l_i} + k_{(e)i+1}(l_{i+1} - l_{0,i+1})\frac{y_{i+3} - y_{i+2}}{l_i + 1} \\
&+ k_{(b)i}(\theta_i - \theta_{i-1} - \theta_{0,i} + \theta_{0,i-1})\frac{x_{i+2} - x_{i+1}}{(x_{i+2} - x_{i+1})^2 + (y_{i+2} - y_{i+1})^2} \\
&- k_{(b)i+1}(\theta_{i+1} - \theta_i - \theta_{0,i+1} + \theta_{0,i})\Bigg(\frac{x_{i+3} - x_{i+2}}{(x_{i+3} - x_{i+2})^2 + (y_{i+3} - y_{i+2})^2} \\
&+ \frac{x_{i+2} - x_{i+1}}{(x_{i+2} - x_{i+1})^2 + (y_{i+2} - y_{i+1})^2}\Bigg) \\
&+ k_{(b)i+2}(\theta_{i+2} - \theta_{i+1} - \theta_{0,i+2} + \theta_{0,i+1})\frac{x_{i+3} - x_{i+2}}{(x_{i+3} - x_{i+2})^2 + (y_{i+3} - y_{i+2})^2}
\end{aligned} \tag{A.3}
$$

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial z_{i+2}} =& - k_{(e)i}(l_i - l_{0,i})\frac{z_{i+2} - z_{i+1}}{l_i} + k_{(e)i+1}(l_{i+1} - l_{0,i+1})\frac{z_{i+3} - z_{i+2}}{l_i + 1} \\
&+ k_{(b)i}(\phi_i - \phi_{i-1} - \phi_{0,i} + \phi_{0,i-1})\frac{x_{i+2} - x_{i+1}}{(x_{i+2} - x_{i+1})^2 + (z_{i+2} - z_{i+1})^2} \\
&- k_{(b)i+1}(\phi_{i+1} - \phi_i - \phi_{0,i+1} + \phi_{0,i})\Bigg(\frac{x_{i+3} - x_{i+2}}{(x_{i+3} - x_{i+2})^2 + (z_{i+3} - z_{i+2})^2} \\
&+ \frac{x_{i+2} - x_{i+1}}{(x_{i+2} - x_{i+1})^2 + (z_{i+2} - z_{i+1})^2}\Bigg) \\
&+ k_{(b)i+2}(\phi_{i+2} - \phi_{i+1} - \phi_{0,i+2} + \phi_{0,i+1})\frac{x_{i+3} - x_{i+2}}{(x_{i+3} - x_{i+2})^2 + (z_{i+3} - z_{i+2})^2}
\end{aligned} \tag{A.4}
$$

Since, $\dfrac{\partial \mathcal{L}}{\partial y_{i+2}} \neq f(\phi)$ and $\dfrac{\partial \mathcal{L}}{\partial z_{i+2}} \neq f(\theta)$, the 3 associated terms vanish compared to the equation (A.2). This means we should have 5 terms as shown above in equation (A.3) and equation (A.4). Please note that

- $\dfrac{\partial \mathcal{L}}{\partial y_n}$ has 4 terms, because 1 term associated with θ, will disappear.
- $\dfrac{\partial \mathcal{L}}{\partial z_n}$ has 4 terms, because 1 term associated with ϕ, will disappear.
- $\dfrac{\partial \mathcal{L}}{\partial y_{n+1}}$has 2 terms, because 1 term associated with l and 2 terms associated with θ will disappear.
- $\dfrac{\partial \mathcal{L}}{\partial z_{n+1}}$has 2 terms, because 1 term associated with l and 2 terms associated with ϕ will disappear.

Bibliography

[1] Philippe Angot, Charles-Henri Bruneau, and Pierre Fabrie. A penalization method to take into account obstacles in incompressible viscous flows. *Numerische Mathematik*, 81(4):497–520, 1999.

[2] Alain J Brizard. *An introduction to Lagrangian mechanics*. World Scientific Publishing Company, 2014.

[3] Claudio Canuto, M Yousuff Hussaini, Alfio Quarteroni, A Thomas Jr, et al. *Spectral methods in fluid dynamics*. Springer Science & Business Media, 2012.

[4] Gilles Carbou, Pierre Fabrie, et al. Boundary layer for a penalization method for viscous incompressible flow. *Advances in Differential equations*, 8(12):1453–1480, 2003.

[5] Thomas Engels. *Numerical modeling of fluid-structure interaction in bio-inspired propulsion*. PhD thesis, Aix-Marseille, 2015.

[6] Thomas Engels, Dmitry Kolomenskiy, Kai Schneider, and Jorn Sesterhenn. Flusi: A novel parallel simulation tool for flapping insect flight using a fourier method with volume penalization. *SIAM Journal on Scientific Computing*, 38(5):S3–S24, 2016.

[7] Ranjan Ganguli, S Gorb, F-O Lehmann, and S Mukherjee. An experimental and numerical study of calliphora wing structure. *Experimental Mechanics*, 50(8):1183–1197, 2010.

[8] Truong Dinh Hung. Torsional spring-mass system for modelling an elastic rod. *Unpublished*.

[9] Dmitry Kolomenskiy and Kai Schneider. A fourier spectral method for the navier–stokes equations with volume penalization for moving solid obstacles. *Journal of Computational Physics*, 228(16):5687–5709, 2009.

[10] Stanley Osher and Ronald Fedkiw. *Level set methods and dynamic implicit surfaces*. Springer, 2003.

[11] Roger Peyret. *Spectral methods for incompressible viscous flow*. Springer Berlin / Heidelberg, 2002.

[12] Michael Schäfer, Stefan Turek, Franz Durst, Egon Krause, and Rolf Rannacher. Benchmark computations of laminar flow around a cylinder. In *Flow simulation with high-performance computers II*, pages 547–566. Springer, 1996.

[13] Kai Schneider. Numerical simulation of the transient flow behaviour in chemical reactors using a penalisation method. *Computers & Fluids*, 34(10):1223–1238, 2005.

[14] Stefan Turek and Jaroslav Hron. Proposal for numerical benchmarking of fluid-structure interaction between an elastic object and laminar incompressible flow. In *Fluid-structure interaction*, pages 371–385. Springer, 2006.

[15] Stefan Turek, Jaroslav Hron, Mudassar Razzaq, Hilmar Wobker, and Michael Schäfer. Numerical benchmarking of fluid-structure interaction: A comparison of different discretization and solution approaches. In *Fluid Structure Interaction II*, pages 413–424. Springer, 2011.