POLITECNICO DI MILANO

Facoltà di Ingegneria Industriale

Corso di Laurea Specialistica in
Ingegneria Aeronautica

Coupled Fluid-Structure Simulation of Flapping Wings

Relatore: Prof. Marco MORANDINI

Tesi di Laurea di:
Mattia ALIOLI Matr. 735677

Anno Accademico 2010-2011

Θάλαττα, θάλαττα

Abstract

In this thesis, we adopted an adaptive finite element method to compute incompressible viscous flows. In particular, we first tested the method, implemented in FEniCS software library, for 2D turbulent flow problems, then we computed 3D, fully coupled, time-dependent fluid–structure interaction problems. We also presented the employment of a non linear 4-node shell element within the free multibody solver MBDyn, using a boundary interfacing approach based on MLS with RBFs, to support the modeling and the analysis of flexible wing oscillating in plunge.
The method we used to compute approximate solutions to the Navier-Stokes equation is G2 method, in form of cG(1)cG(1), with friction boundary conditions. The cG(1)cG(1) method is a stabilized Galerkin finite element method using continues, piecewise linear trial functions in time and space with continues piecewise linear test functions in space and piecewise constant test functions in time. Instead of resolving the turbulent boundary layer, the method uses a friction boundary condition as a wall model. The adaptive cG(1)cG(1) method is based on "a posteriori" error estimate, where space discretization is adaptively modified based on the solution of an auxiliary linearized dual problem to control the error in a given goal functional of interest. This because understanding the mechanics of turbulent fluid flow is of key importance for industry and society and the massive computational cost for resolving all turbulent scales in a realistic problem makes direct numerical simulation of the underlying Navier-Stokes equations impossible.

Keywords: Adaptive finite element method; fluid-structure interaction; flapping wings.

Sommario

Durante questo lavoro di tesi è stato implementato un metodo ad elementi finiti adattivo per il calcolo di flussi incomprimibili e viscosi. In particolare tale metodo, sviluppato grazie all'ambiente di programmazione fornito dal progetto FEniCS, è stato dapprima applicato a problemi di flusso turbolento bidimensionale, per poi procedere al calcolo dell'interazione fluido-struttura per problemi tridimensionali nonstazionari. Per l'analisi e la modellazione di ali flessibili, si è ricorso all'utizzo di un elemento di piastra a 4 nodi non lineare all'interno del solutore libero multicorpo MBDyn, usando un strategia per esplicitare il problema di interfaccia basata su di uno schema di interpolazione ai minimi quadrati mobili, con funzioni a supporto radiale compatto.
Il metodo utlizzato per risolvere le equazioni di N.S. è il cosidetto metodo G2, nella forma di cG(1)cG(1), con condizioni al contorno di attrito. Il metodo cG(1)cG(1) è un metodo agli elementi finiti di Galerkin stabilizzato che utilizza funzioni *trial* continue e lineari a tratti sia in spazio che in tempo, mentre utilizza funzioni *test* continue e lineari a tratti in spazio e costanti a tratti in tempo. In tale metodo è stata implementata una condizione al contorno di attrito come modello di parete. Il metodo adattivo cG(1)cG(1) è basato su di una stima "a posteriori" dell'errore, in cui la discretizzazione spaziale è adattivamente modificata basandosi sulla risoluzione di un problema linearizzato duale per controllare l'errore rispetto ad una funzione obiettivo scelta. Questo perchè comprendere bene i meccanismi che governano un flusso turbolento è un aspetto fondamentale per l'industria e la risoluzione di tutte le scale turbolente presenti in un problema realistico rende impossibile una simulazione numerica diretta anche per la potenza computazionale disponibile al giorno d'oggi.

Keywords: Metodo ad elementi finiti adattivo; interazione fluido-struttura; ali oscillanti.

Acknowledgements

This work could not have been accomplished without the support, encouragement and enthusiasm of many people surrounding me.
First of all, I would like to express my sincere gratitude to my supervisor, professor Marco Morandini, for his valuable assistance, motivation and encouragement in my work. He helped me prepare numerical simulations and I never hesitated to ask him any question.
I am also very grateful to professor Pierangelo Masarati for his suggestions in understanding the multibody MBDyn code used throughout this thesis.
After the academic credits, I would like to thank my present and former colleagues: it is not possible to thank everyone individually, I would, therefore, like to express my gratefulness to everybody who was there for me. Their friendship and support helped me a lot in completing this work.
Finally, I wish to extend my deepest thanks to my parents, brother and relatives, for their love, support, encouragement and understanding.

Alioli Mattia
Politecnico di Milano, April 2012

Summary in Italian

Questo lavoro di tesi ha avuto come obiettivo quello di analizzare l'interazione fluido-struttura per ali flessibili flappeggianti. Per raggiungere tale scopo, ci si è concentrati dapprima sullo sviluppo di un metodo ad elementi finiti per la risoluzione del flusso incomprimibile attorno ad un profilo bidimensionale rigido oscillante, poi si è passato alla risoluzione del flusso turbolento attorno ad un'ala rigida di apertura finita oscillante ed, infine, a quello attorno ad un'ala flessibile.

Le motivazioni che sono alla base di tale lavoro sono da ricercarsi nel grande interesse, sia in ambito civile che militare, che si è sviluppato attorno a piccoli aerei radio-controllati, noti come *Micro Air Vehicles* (MAVs). Per sfruttare su questi micro-aerei le prestazioni e l'efficienza del volo tipico di uccelli e insetti, i quali hanno sviluppato l'arte del volo in milioni di anni di evoluzione, si può ricorrere ad un meccanismo che permette all'ala di flappeggiare. Per queste ragioni è necessario comprendere bene l'aerodinamica che si sviluppa attorno a profili oscillanti e sfruttare l'accoppiamento tra l'oscillazione dell'ala e il suo grado di flessibilità.

Alla base di numerosi fenomeni aerodinamici ci sono le equazioni di Navier-Stokes, che sono un'estensione delle equazioni di Eulero includendo gli effetti della viscosità, e per una corrente incomprimibile sono derivate dalle equazioni di conservazione della massa e della quantità di moto. Considerando il moto di un fluido con densità costante ρ in un dominio $\Omega \subset \mathbb{R}^d$ (con $d = 2, 3$), queste equazioni si esprimono come:

$$\begin{cases} \dfrac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} - 2\nu\nabla \cdot \epsilon(\boldsymbol{u}) + \nabla p = \boldsymbol{f}, & \boldsymbol{x} \in \Omega, t > 0, \\ \nabla \cdot \boldsymbol{u} = 0, & \boldsymbol{x} \in \Omega, t > 0. \end{cases} \tag{1}$$

dove p è la pressione scalata per la densità, $\nu = \frac{\mu}{\rho}$ è la densità cinematica, $\epsilon(\boldsymbol{u}) = \frac{1}{2}(\nabla(\boldsymbol{u}) + \nabla^\top(\boldsymbol{u}))$ il tensore di velocità di deformazione. Per risolvere il sistema (1) occorre imporre delle opportune condizioni al contorno e delle condizioni iniziali, come mostrato in seguito, indicando con Γ_D e

Γ_{N} delle porzioni complementari di $\partial\Omega$, contorno di Ω:

$$\begin{cases} \dfrac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u}\cdot\nabla)\boldsymbol{u} - 2\nu\nabla\cdot\epsilon(\boldsymbol{u}) + \nabla p = \boldsymbol{f}, & \boldsymbol{x}\in\Omega, t>0,\\ \nabla\cdot\boldsymbol{u} = 0, & \boldsymbol{x}\in\Omega, t>0,\\ \boldsymbol{u}(\boldsymbol{x},t) = \boldsymbol{g}(\boldsymbol{x},t), & \boldsymbol{x}\in\Gamma_{\mathrm{D}}, t>0,\\ \boldsymbol{n}\cdot\sigma = -p\boldsymbol{n} + 2\nu\boldsymbol{n}\cdot\epsilon(\boldsymbol{u}) = \boldsymbol{t}, & \boldsymbol{x}\in\Gamma_{\mathrm{N}}, t>0,\\ \boldsymbol{u}(\boldsymbol{x},0) = \boldsymbol{u}(\boldsymbol{x}), & \boldsymbol{x}\in\Omega. \end{cases} \tag{2}$$

dove σ è il tensore di Cauchy "dinamico" o "scalato" (diviso per la densità), i.d., $\sigma = -p\mathbb{I} + 2\nu\epsilon(\boldsymbol{u})$. Utilizzando un semplice modello di parete, come quello presentato in §Hoffman and Johnson [16], possiamo implementare delle condizioni al contorno di strisciamento con attrito e penetrazione con resistenza, per una porzione Γ_{sfpr} del contorno con versore normale $\boldsymbol{n}$ e due vettori tangenti ortogonali $\boldsymbol{\tau}_1$ e $\boldsymbol{\tau}_2$, semplicemente variando i valori di α e β in:

$$\begin{aligned} \boldsymbol{u}\cdot\boldsymbol{n} + \alpha\,\boldsymbol{n}^\top\sigma\boldsymbol{n} &= 0,\\ \boldsymbol{u}\cdot\boldsymbol{\tau}_k + \beta^{-1}\,\boldsymbol{n}^\top\sigma\boldsymbol{\tau}_k &= 0, \quad k=1,2. \end{aligned} \tag{3}$$

La formulazione debole di (2) si ottiene nel consueto modo, cioè moltiplicando per una coppia di funzione test $(\boldsymbol{v},q)\in(V_0,Q)$ e integrando per parti l'equazione di conservazione della quantità di moto, definendo inoltre i seguenti spazi funzionali:

$$V_g := \left\{ \boldsymbol{v}\in(H^1(\Omega))^d\colon \boldsymbol{v}\big|_{\Gamma_{\mathrm{D}}} = \boldsymbol{g} \right\},$$

$$V_0 := \left\{ \boldsymbol{v}\in(H^1(\Omega))^d\colon \boldsymbol{v}\big|_{\Gamma_{\mathrm{D}}} = \boldsymbol{0} \right\},$$

$$Q := L^2(\Omega) = \left\{ f\colon\Omega\mapsto\mathbb{R}\colon \int_\Omega f(\boldsymbol{x})^2\,\mathrm{d}\Omega < +\infty \right\}.$$

dove $H^1(\Omega)$ è il classico spazio di Hilbert di funzioni che sono quadrato integrabili insieme con la loro derivata prima. Scomponendo inoltre la funzione test $\boldsymbol{v}$ su Γ_{sfpr} in d componenti ortonormali, come suggerito in §John [17], e indicando con $(\cdot,\cdot)$ il prodotto interno in $(L^2(\Omega))^d$, $d = 1,2,3$, la formulazione variazionale risulta: trovare $(\boldsymbol{u},p)\in(V_g,Q)$ tali che per ogni $(\boldsymbol{v}, q) \in (V_0, Q)$

$$\left\{\begin{aligned} &(\boldsymbol{u}_t, \boldsymbol{v}) + 2\nu(\epsilon(\boldsymbol{u}), \epsilon(\boldsymbol{v})) + ((\boldsymbol{u}\cdot\nabla)\boldsymbol{u}, \boldsymbol{v}) - (p, \nabla\cdot\boldsymbol{v}) + \\ &\qquad + \int_{\Gamma_{\text{sfpr}}} \alpha^{-1}(\boldsymbol{u}\cdot\boldsymbol{n})(\boldsymbol{v}\cdot\boldsymbol{n})\,\mathrm{d}S + \\ &\qquad + \int_{\Gamma_{\text{sfpr}}} \sum_{k=1}^{d-1} \beta(\boldsymbol{u}\cdot\boldsymbol{\tau}_k)(\boldsymbol{v}\cdot\boldsymbol{\tau}_k)\,\mathrm{d}S = (\boldsymbol{f}, \boldsymbol{v}), \\ &(\nabla\cdot\boldsymbol{u}, q) = 0 \end{aligned}\right. \tag{4}$$

avendo usato $\boldsymbol{u}_t$ per indicare $\frac{\partial \boldsymbol{u}}{\partial t}$.

Per trovare delle funzioni che soddisfino le equazioni di N.S. solo in modo approssimato in forma debole, in questa tesi, viene utilizzato un metodo agli elementi finiti alla Galerkin stabilizzato, qui indicato come metodo G2, che può essere visto come una combinazione di un metodo di Galerkin, che assicura che il residuo sia piccolo "in media", ed una stabilizzazione ai minimi quadrati pesati, che corrisponde ad un certo controllo "forte" del residuo. In particolare, il metodo utilizzato è il cG(1)cG(1), che è un tipo di metodo G2, e che usa funzioni *trial* continue lineari a tratti sia in tempo che in spazio e funzioni *test* continue lineari a tratti in spazio ma costanti a tratti in tempo. Sia $0 = t_0 < t_1 < \cdots < t_N = T$ una sequenza discreta di istanti di tempo, con intervalli temporali associati $I_n = (t_{n-1}, t_n]$ di dimensione $k_n = t_n - t_{n-1}$, e sia $W^n \subset H^1(\Omega)$ uno spazio di elementi finiti composto da funzioni continue lineari a tratti su di una griglia $\mathcal{T}_n = \{K\}$ di dimensione $h_n(\boldsymbol{x})$ con W_0^n le funzioni in W^n che soddisfano delle condizioni di Dirichlet omogenee su di una porzione Γ_D del contorno di Ω. Allora il metodo cG(1)cG(1) per le equazioni di N.S. (4) risulta: per $n = 1, ..., N$, trovare $(\boldsymbol{U}^n, P^n) = (\boldsymbol{U}(t_n), P(t_n))$ con $\boldsymbol{U}^n \in V_0^n \equiv [W_0^n]^3$ e $P^n \in W^n$, tali che

$$\begin{aligned} &((\boldsymbol{U}^n - \boldsymbol{U}^{n-1})k_n^{-1} + \bar{\boldsymbol{U}}^n\cdot\nabla\bar{\boldsymbol{U}}^n, v) + (2\nu\epsilon(\bar{\boldsymbol{U}}^n), \epsilon(\boldsymbol{v})) - (P^n, \nabla\cdot\boldsymbol{v}) + \\ &+ (\nabla\cdot\bar{\boldsymbol{U}}^n, q) + \alpha^{-1}\langle(\boldsymbol{U}\cdot\boldsymbol{n}), (\boldsymbol{v}\cdot\boldsymbol{n})\rangle_{\Gamma_{\text{sfpr}}} + \beta\langle\boldsymbol{U}\otimes\boldsymbol{v}, \mathbb{I} - \boldsymbol{n}\otimes\boldsymbol{n}\rangle_{\Gamma_{\text{sfpr}}} + \\ &+ SD_\delta(\bar{\boldsymbol{U}}^n, P^n; \boldsymbol{v}, q) = (\boldsymbol{f}, \boldsymbol{v}) \quad \forall\,\hat{\boldsymbol{v}} = (\boldsymbol{v}, q) \in V_o^n \times W^n \end{aligned} \tag{5}$$

dove $\bar{\boldsymbol{U}}^n = \frac{1}{2}(\boldsymbol{U}^n + \boldsymbol{U}^{n-1})$ e P^n sono costanti a tratti in tempo su I_n, e con il termine di stabilizzazione dato da:

$$\begin{aligned} SD_\delta(\bar{\boldsymbol{U}}^n, P^n; \boldsymbol{v}, q) = {} & (\delta_1(\bar{\boldsymbol{U}}^n\cdot\nabla\bar{\boldsymbol{U}}^n + \nabla P^n - \boldsymbol{f}), \bar{\boldsymbol{U}}^n\cdot\nabla\boldsymbol{v} + \nabla q) + \\ & + (\delta_2\nabla\cdot\bar{\boldsymbol{U}}^n, \nabla\cdot\boldsymbol{v}) \end{aligned} \tag{6}$$

dove si è sfruttata la seguente proprietà per tre vettori unitari ortonormali, per esprimere la condizione al contorno di strisciamento con attrito:

$$\boldsymbol{\tau}_1 \otimes \boldsymbol{\tau}_1 + \boldsymbol{\tau}_2 \otimes \boldsymbol{\tau}_2 + \boldsymbol{n} \otimes \boldsymbol{n} = \mathbb{I}$$

I parametri di stabilizzazione sono espressi come: $\delta_1 = \kappa_1(k_n^{-2} + |\boldsymbol{U}|^2 h_n^{-2})^{-\frac{1}{2}}$ e $\delta_2 = \kappa_2 h_n |\boldsymbol{U}|$ nel caso di convezione dominante; cioè se $\nu < |\boldsymbol{U}|h_n$. Nel caso di diffusione dominante, i parametri sono dati da $\delta_1 = \kappa_1 h_n^2/(|\boldsymbol{U}|L)$ e $\delta_2 = \kappa_2 h_n^2 |\boldsymbol{U}|/L$. Qui, κ_1 e κ_2 sono costanti positive di dimensione unitaria ed L la lunghezza di riferimento. Inoltre:

$$\begin{aligned}
(\boldsymbol{v}, \boldsymbol{w}) &= \sum_{K \in \mathcal{T}_n} \int_K \boldsymbol{v} \cdot \boldsymbol{w} \, \mathrm{d}\boldsymbol{x}, \\
(\epsilon(\boldsymbol{v}), \epsilon(\boldsymbol{w})) &= \sum_{i,j=1}^{3} (\epsilon_{ij}(\boldsymbol{v}), \epsilon_{ij}(\boldsymbol{w})), \qquad (7)\\
\langle \boldsymbol{v}, \boldsymbol{w} \rangle_{\Gamma_{\text{sfpr}}} &= \int_{\Gamma_{\text{sfpr}}} (\boldsymbol{v} : \boldsymbol{w}) \, \mathrm{d}S.
\end{aligned}$$

Il metodo cG(1)cG(1) può essere utilizzato per calcolare adattivamente soluzioni approssimate con l'obiettivo di soddisfare una certa tolleranza rispetto ad un specifico *output* di interesse (solitamente la resistenza o la portanza). Rispetto a questo quantità, il metodo adatta automaticamente la griglia basandosi su di una stima "a posteriori" dell'errore e risolve le caratteristiche del flusso che hanno grande influenza sull'*output* scelto, mentre le altre scale rimangono irrisolte con la stabilizzazione che agisce come un modello numerico di turbolenza. Per flussi turbolenti questo metodo può essere visto come una DNS/LES adattiva, dove parte del flusso viene risolta come in una DNS, mentre un'altra parte rimane irrisolta come in una LES. L'algoritmo adattivo utilizzato è basato sul metodo presentato in §Rognes and Logg [24] per cercare di ottenere un controllo automatizzato *goal-oriented* dell'errore, basato sulla risoluzione automatizzata di un problema ausiliario (duale), sulla derivazione e sulla valutazione automatizzata di un stima "a posteriori" dell'errore, e sul raffinamento automatico adattivo della griglia per controllare che l'errore in un funzionale scelto sia al di sotto di una certa tolleranza. Ciò rende possibile la costruzione un algoritmo per il raffinamento adattivo della griglia rispetto ad una certa quantità di interesse.

In un problema di iterazione fluido-struttura le forze che agiscono sulla struttura deformabile causano la modifica delle condizioni al contorno per il problema aerodinamico, allora l'approccio più semplice e corretto, ma anche computazionalmente più oneroso, consiste nel deformare la griglia di calcolo aerodinamica e riformulare il problema aerodinamico secondo una strategia *Arbitrary Lagrangian-Eulerian* (ALE). Considerando quindi la velocità $\hat{\boldsymbol{v}}$ della griglia, le equazioni di N.S. (2) nella descrizione ALE diventano:

$$\begin{cases} \boldsymbol{u}_t + ((\boldsymbol{u} - \hat{\boldsymbol{v}}) \cdot \nabla)\boldsymbol{u} - 2\nu\nabla \cdot \epsilon(\boldsymbol{u}) + \nabla p = \boldsymbol{f}, & \boldsymbol{x} \in \Omega, t > 0, \\ \nabla \cdot \boldsymbol{u} = 0, & \boldsymbol{x} \in \Omega, t > 0, \\ \boldsymbol{u}(\boldsymbol{x}, t) = \boldsymbol{g}(\boldsymbol{x}, t), & \boldsymbol{x} \in \Gamma_{\mathrm{D}}, t > 0, \\ \boldsymbol{n} \cdot \sigma = -p\boldsymbol{n} + 2\nu\boldsymbol{n} \cdot \epsilon(\boldsymbol{u}) = \boldsymbol{t}, & \boldsymbol{x} \in \Gamma_{\mathrm{N}}, t > 0, \\ \boldsymbol{u}(\boldsymbol{x}, 0) = \boldsymbol{u}_0(\boldsymbol{x}), & \boldsymbol{x} \in \Omega, \\ (\boldsymbol{u} - \hat{\boldsymbol{v}}) \cdot \boldsymbol{n} + \alpha\, \boldsymbol{n}^\top \sigma \boldsymbol{n} = 0, & \boldsymbol{x} \in \Gamma_{\mathrm{sfpr}}, t > 0, \\ (\boldsymbol{u} - \hat{\boldsymbol{v}}) \cdot \boldsymbol{\tau}_k + \beta^{-1}\, \boldsymbol{n}^\top \sigma \boldsymbol{\tau}_k = 0, \quad k = 1, 2 & \boldsymbol{x} \in \Gamma_{\mathrm{sfpr}}, t > 0. \end{cases} \tag{8}$$

Per determinare la velocità $\hat{\boldsymbol{v}}$ dei nodi della griglia aerodinamica, noti gli spostamenti e le velocità dei nodi aerodinamici appartenenti all'interfaccia fluido-struttura, in questa tesi, si è ricorso alla soluzione di un problema elastico "fittizio", dove il modulo elastico di ogni elemento è inversamente proporzionale al proprio volume (o area). Se lo spostamento dei nodi appartenenti al contorno del corpo provoca la formazione di alcuni elementi a volume negativo, a causa dell'eccessiva distorsione subita, allora il modulo elastico di tali elementi viene incrementato e il problema elastico viene riformulato.

Il compito di interpolare gli spostamenti e le velocità strutturali sui nodi aerodinamici appartenenti al contorno del corpo è lasciato ad un opportuno schema di interfaccia, il cui obbiettivo è quello di realizzare la connessione ad anello chiuso tra il sistema strutturale e quello aerodinamico mediante un'opportuna procedura di interpolazione e garantire che lo scambio di informazione avvenga nel modo più accurato ed efficiente possibile. Ad esempio, è necessario sapere tradurre gli spostamenti e le velocità strutturali in variazioni delle condizioni al contorno del sistema aerodinamico ed analogamente le forze aerodinamiche in una condizione di carico agente sul sistema strutturale. Quindi occorre cercare un operatore lineare che, noto il vettore degli spostamenti strutturali, restituisca il vettore degli spostamenti interpolati sui nodi aerodinamici appartenenti al contorno del corpo nel seguente modo:

$$\boldsymbol{u}_a = H\boldsymbol{u}_s$$

dove H è la matrice di interfaccia incognita. Per garantire la conservazione del lavoro (virtuale) svolto nel sistema strutturale ed nel sistema aerodinamico, l'operatore lineare che consente di riportare i carichi aerodinamici sui nodi strutturali corrisponde alla matrice di interfaccia incognita trasposta:

$$\boldsymbol{F}_s^a = H^\top \boldsymbol{F}_a^a$$

La strategia qui adoperata per esplicitare la matrice H, secondo un'approccio agli spostamenti, è quella proposta in §Quaranta [22], e consiste nell'utilizzare uno schema di interpolazione ai minimi quadrati mobili o *Moving Least Squares* (MLS), che consente di costruire un'approssimazione sufficientemente regolare ed accurata del campo degli spostamenti e delle velocità strutturali in corrispondenza dei nodi aerodinamici appartenenti al contorno del corpo conoscendo la soluzione solo all'interno di una nuvola di nodi strutturali in generale irregolarmente distribuiti. Il campo degli spostamenti e delle velocità strutturali è quindi approssimato in corrispondenza dei nodi aerodinamici appartenenti al contorno del corpo mediante uno seguente sviluppo polinomiale ad m termini del tipo:

$$\hat{\boldsymbol{u}}(\boldsymbol{x}) = \sum_{i=1}^{m} p_i(\boldsymbol{x}) a_i(\boldsymbol{x}) = \boldsymbol{p}^\top \boldsymbol{a} \tag{9}$$

dove $p_i(\boldsymbol{x})$ è la i-esima funzione di di base polinomiale di ordine opportuno, generalmente lineare o quadratica per evitare problemi di instabilità numerica, ed $a_i(\boldsymbol{x})$ è l'i-esimo coefficiente moltiplicativo incognito. Questi ultimi possono essere determinati minimizzando un'opportuna norma dell'errore locale commesso utilizzando la funzione interpolante $\hat{\boldsymbol{u}}(\boldsymbol{x})$ mediante il metodo dei minimi quadrati pesati, ovvero minimizzando il seguente funzionale scritto in forma variazionale:

$$J(\boldsymbol{x}) = \int_\Omega \Phi(\boldsymbol{x}) \|\boldsymbol{\varepsilon}(\boldsymbol{x})\|^2 \,\mathrm{d}\Omega \tag{10}$$

dove $\Phi(\boldsymbol{x})$ è un'opportuna funzione peso e $\boldsymbol{\varepsilon}(\boldsymbol{x})$ l'errore locale.
Considerando ad esempio un supporto di N_j nodi strutturali $\boldsymbol{x}_{s,j}$, in generale irregolarmente distribuiti ed in corrispondenza dei quali siano noti gli spostamenti strutturali $\boldsymbol{u}_{s,j}$, i coefficienti moltiplicativi incogniti $a_i(\boldsymbol{x})$ possono essere determinati minimizzando il seguente funzionale scritto in forma discretizzata:

$$J(\boldsymbol{x}) = \sum_{j=1}^{N_j} \Phi(\boldsymbol{x} - \boldsymbol{x}_{s,j}) \left\| \sum_{i=1}^{m} p_i(\boldsymbol{x}_{s,j}) a_i(\boldsymbol{x}) - \boldsymbol{u}_{s,j} \right\|^2 .$$

Come funzioni peso sono utilizzate le *Radial Basis Functions* (RBFs) a supporto compatto e di grado minimo espresse nella forma $\Phi(r/\delta)$, dove $r = \|\boldsymbol{x} - \boldsymbol{x}_{s,j}\|$ e δ è un fattore di scala che consente di modificare localmente le dimensioni N_j del supporto dei nodi strutturali $\boldsymbol{x}_{s,j}$, considerandone solo un numero minimo sufficiente di relativamente vicini e scartando viceversa quelli più distanti in modo tale da contenere il costo computazionale.

Avendo come obiettivo ultimo quello di analizzare l'interazione fluido-struttura per ali flessibili flappeggianti, in questa tesi, è stata utilizzata una formulazione di elemento finito di piastra a 4 nodi, non lineare, sviluppata e validata in §Quaranta et al. [23], a cui si rimanda per ulteriori chiarimenti, all'interno dell'ambiente multicorpo fornito dal solutore (strutturale) libero multicorpo MBDyn, sviluppato al Dipartimento di Ingegneria Aerospaziale del Politecnico di Milano, mentre per risolvere il problema aerodinamico ci si è affidati all'uso di un ambiente di programmazione piuttosto recente (FEniCS/DOLFIN) ed a Python, la cui combinazione permette la scrittura di codice compatto di "alto" livello, molto vicino alla notazione matematica delle equazioni che devono essere risolte, e da cui codice di "basso" livello è automaticamente generato.

FEniCS è una collezione di *software* liberi, in combinazione con un'estesa lista di servizi, che permette la creazione e la risoluzione di problemi retti da equazioni differenziali usando il metodo degli elementi finiti. Alla base di FEniCS si trova DOLFIN, che oltre ad esserne un componente fondamentale, svolge il ruolo di interfaccia-utente. In FEniCS, la scrittura di codice è combinata con il simbolismo matematico attraverso l'adozione di un linguaggio specifico (UFL) per la dichiarazione della formulazione variazionale delle equazioni differenziali parziali o ordinarie, incorporato nel linguaggio di programmazione Python. Le equazioni differenziali alle derivate parziali possono essere scritte in una notazione molto simile a quella matematica (così come i problemi variazionali ad elementi finiti) e risolte automaticamente.

Come esperimenti numerici, per verificare la "bontà" del metodo sviluppato, sono stati presi in considerazione i seguenti problemi:

- Flusso instazionario attorno ad un profilo NACA 0012, posto ad un' incidenza di 34°, con $Re = 1000$, confrontando i risultati ottenuti con quelli da §Guermond and Quartapelle [7].
- Flusso attorno ad un profilo NACA 0012 oscillante sinusoidalmente in direzione verticale, a $Re = 20000$, confrontando i risultati ottenuti con quelli da §Heathcote and Gursul [8]; Heathcote et al. [9]; Young and Lai [26].
- Effetto della flessibilità in apertura sulle caratteristiche di trazione per un'ala rettangolare oscillante in direzione verticale ad un'estremità, confrontando i risultati ottenuti con quelli da §Heathcote et al. [9].

Contents

List of Figures

List of Tables

Part I

Introductory Chapters

1 Introduction

Computer simulation is an important tool in many disciplines of science and engineering since many of the problems that are being simulated are computationally expensive and computer resources must therefore be used wisely. One of these problems, e.g., is fluid–structure interaction (FSI), which require the solution of the Navier-Stokes equations in a time-dependent flow domain. In order to accurately represent the solution of these problems, the numerical method requires the use of moving and deforming meshes. Several numerical techniques can deal with deforming meshes: in this thesis we focused on finite element methods since they provide an excellent framework for solving the Navier-Stokes equations and make it possible to efficiently deal with complicated geometry and to adapt the computational mesh to accurately capture flow phenomena such as boundary layers, vortical structures, etc. In particular, arbitrary Lagrangian-Eulerian finite element techniques have been successfully used to deal with time-dependent fluid flow problems with changing spatial configurations. A tremendous amount of computing power has been expended to solve N.S. equations, often for simple geometries, far from engineering applications: for simulating turbulent flow within a reasonable amount of time one can resort to adaptive finite element methods, which are based on the idea that we want to compute the solution with good accuracy to a minimal computational cost, or, alternatively, compute a solution with as good accuracy as possible to a given computational cost. The main part of this work thus concerned the efficient simulation of fluid flow using adaptive methods based on General Galerkin (G2) finite element methods. When applied to advection dominated problems, the Galerkin method lacks stability: a least-squares operator is so added to the basic Galerkin formulation in order to overcome these difficulties. When properly defined, these operators guarantee stability without compromising accuracy. The Galerkin least-squares method, in combination with a suitable stabilization operator, is an excellent method to accurately compute the periodic shedding of vortices and, in combination with the space-time formulation, it is well suited for moving and deforming meshes.

In this thesis, analysis struments that are freely available on Internet were adopted, where it was possible, not only for the solutions of the structural and aerodynamic problems, but also for the pre/post-processing phases. Referring to Figure 1.1, we adopted as structural solver the Open Source multibody-multidisciplinary code MBDyn, in which an interface scheme that provides to exchanges information between the structural and the aerodynamic models has been implemented; we utilized some tools of the FEniCS project (a recent C++/Python framework, where systems of PDEs and corresponding discretization and iteration strategies can be defined in terms of a few high-level Python statements which inherit the mathematical structure of the problem and from which low level code is generated) to solve the aerodynamic problem. The choice of utilizing only freely available software extended also on the elaboration and graphic presentation of the numeric results by Octave, Gnuplot, and the post-processor ParaView. The only commercial software we utilized is Abaqus to generate a 3D computational grid for the computation in Chapter 11, while in 2D cases we exploited the functionality of the Open Source TriTetMesh interface to Triangle/Tetgen packages to create meshes.

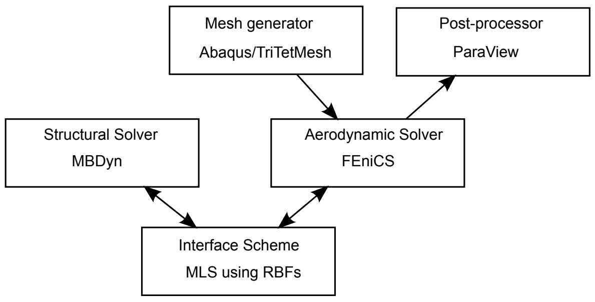


Figure 1.1: Scheme of the structure of the toolbox for the FSI analysis.

1.1 Motivation

There is great interest in small radio-controlled aircraft known as Micro Air Vehicles (MAVs): these crafts, whose size is less than 15 cm in length, width, and height, represent an interesting means to provide reconnaissance and surveillance capabilities in dangerous environments, for both military and civil tasks. Many applications, which require great maneuverability and demand the ability to hover, have been suggested for a MAV carrying a miniature video camera or other sensing device.

It is thought that it may be possible to approach the agility and endurance of birds and insects, which have perfected the art of flying through millions of years of evolution, by adopting a flapping wing mechanism, and for this reason there is a need to understand the aerodynamics of oscillating airfoils. The flight of insects have stimulated a great deal of interest as their flight seems improbable based on the conventional theories of aerodynamics. To support the body weight and maneuver, the wings of an insect must be able to produce two to three times more lift than a conventional fixed wing flight: in most of the insects, the wing stall during the flight has been observed to produce a leading edge vortex which helps in the production of the required lift. This has resulted in studies to find a way of harnessing the efficiency of the insect flight and apply it to the development of MAVs.

Because the length scale of birds, insects, and MAVs is of the order of the centimeter, the Reynolds number is very low, typically $Re = 10^3 \div 10^5$, and an understanding of the nature and relative importance of inviscid and viscous phenomena in this regime is sought. Minimum wing area for ease of packing and launch handling are also require: fixed wings become less efficient as the size and speed of the vehicle decreases. At the low Reynolds numbers of birds, insects, fish, and MAVs, flow separation tends to occur at the leading edge, especially at high angles of attack: these effects can't be predicted by inviscid methods and, for this reason, Navier–Stokes codes have been developed. These methods are able to predict leading-edge flow separation, vortex formation and shedding, and the consequent merger of the leading-edge vortices into the trailing-edge vortex system.

Birds and insects also exploit the coupling between the flapping wing and the wing deformation to enhance their aerodynamic performance. Flexibility is an interesting subject in the design of MAVs, because in addition to any aerodynamic benefits it is noted that flexible wings are inherently light. Despite the knowledge that flexibility is of importance in bird and insect flight, and essential in fish swimming, the effect of wing stiffness, in either the chordwise or spanwise direction, is relatively unexplored: the majority of experimental studies present in literature have focused on rigid airfoils. One of the first studies about the effect of chordwise flexibility for an airfoil in heave at low Reynolds numbers was made by §Heathcote and Gursul [8]. The progression to a study of spanwise flexibility follows in §Heathcote et al. [9]: the phase of the flexing motion relative to the heave was found to be a key parameter in determining the thrust and efficiency characteristics of the wing/fin (in-phase motions yielded a benefit in efficiency and a significant increase in thrust, out of phase motions were found to be detrimental).

1.2 Brief history of vehicles with flapping airfoils

Studies trying to mimic the flight of birds and insects dates back to early days: everyone knows the ancient Greek legend of Daedalus that was imprisoned, with his son Icarus, by King Minos but, mading wings of wax and feathers, he could successfully fly away, instead of Icarus that fell to his death in the ocean because he flew too high and too near the sun causing "his" wing to melt down. Around 1490, Leonardo da Vinci began to study the flight of birds. He grasped that humans are too heavy, and not strong enough, to fly using wings simply attached to their arms: he made several drawings that illustrated his theories on flight and designed a human-powered device, which, although this was not capable of flight, showed a great deal of careful thought and engineering. The first ornithopters (from Greek ornithos "bird" and pteron "wing") capable of flight were constructed in France in the 1870s for a demonstration for the French Academy of Science: its wings were flapped by gunpowder charges. Around 1890, several ornithopters powered by steam or compressed air was built. In the 1930s, the piston internal combustion engine was harnessed by some researchers in Germany. Human-powered birdlike ornithopters that were towed into the air, and then released to perform powered glides, were also built and tested in early 1900. In 1960s Percival Spencer, of the United States, developed a remarkable series of engine-powered free-flight ornithopter models, with various sizes and different engine sizes. In 1990s the FAI recognized a Harris/DeLaurier engine-powered model as the first successful engine-powered remotely-piloted ornithopter. On this model is based the Project Ornithopter engine-powered piloted aircraft that in 1999 self accelerated (flapping alone) on level pavement to lift-off speed.

1.3 Outline of the Thesis

The structure of the present work is described in the following. Starting in Chapter 2 by introducing the physical explanation of the model, we present the Navier-Stokes equations for an incompressible flow of a fluid that we shall assume Newtonian. In Chapter 3 we give a summary of the history of turbulent flow and computational methods as well as an explanation about the mathematical formulation we used to solve the problem. We present the cG(1)cG(1) method for simulation of turbulent fluid flow, and discuss its implementation to derive an adaptive mesh refinement algorithm based on "a posteriori" error estimate. In Chapter 4 the relation between the arbitrary Lagrangian-Eulerian (ALE) and space-time weak formulation of the incompressible N.S. equations is shown. After presenting in Chapter 5 the Open Source multibody code MBDyn and the basic elements of the multibody formulation here adopted for the representation of the structural behavior, we describe in Chapter 6 the problem of boundary conditions exchange between the structural and the aerodynamic model and present a methodology to solve this aspect in a very general and topology independent manner. In Chapter 7 we describe our implementation in the Open Source project FEniCS, which is also briefly explained, focusing on the parts that are used for the present work. The features of this project that provide the production of unstructured 2D or 3D meshes are presented in Chapter 8.

Chapter 9 reports the numeric results for a test problem in two dimension: the unsteady incompressible viscous flow past NACA 0012 at incidence.

In Chapter 10 a study of a rigid airfoil heaving with constant amplitude has been carried out.

Finally, in Chapter 11, a study of effect of spanwise flexibility on the thrust and lift of a rectangular wing oscillating in pure wave has been performed.

2 Incompressible Navier-Stokes equations

In this thesis, for our computations, we are going to use the Navier Stokes (N.S.) equations, which were formulated 1825-45. These equations are widely used in Computational Fluid Dynamics (CFD) for computing both laminar and turbulent flow.

2.1 Stress tensor in Newtonian fluid

When a fluid is at rest, only normal stresses are present and the stress tensor has the isotropic form:

$$\sigma_{ij} = -p\delta_{ij} \tag{2.1}$$

where p is the static fluid pressure and δ_{ij} the Kronecker delta. In a fluid in motion, tangential stresses are non-zero and the normal component of the stress acting across a surface element depends on the direction of the normal to the element. So, in order to define the pressure at a point in a moving fluid, the quantity $-\frac{1}{3}\sigma_{ii}$ (index summation convection) is used, which is invariant under rotation of the reference axes and reduces to the static fluid pressure if fluid is at rest:

$$p = -\frac{1}{3}\sigma_{ii}. \tag{2.2}$$

This is a purely mechanical definition of pressure, which is thus not connected to the definition of pressure in thermodynamics. It's convenient to decompose the Cauchy stress tensor σ_{ij} into the sum of an isotropic part and a remaining non-isotropic part s_{ij}, the deviatoric stress tensor.

$$\sigma_{ij} = -p\delta_{ij} + s_{ij}. \tag{2.3}$$

The velocity gradient is a second order tensor: it may be decomposed into its symmetric and skew-symmetric parts according to:

$$\nabla \boldsymbol{u} = \nabla^{\mathrm{S}}\boldsymbol{u} + \nabla^{\mathrm{W}}\boldsymbol{u}, \quad \text{where} \begin{cases} \nabla^{\mathrm{S}} := \frac{1}{2}(\nabla + \nabla^{\top}), \\ \nabla^{\mathrm{W}} := \frac{1}{2}(\nabla - \nabla^{\top}). \end{cases} \tag{2.4}$$

The symmetric tensor ∇^{S} is called the *strain rate tensor*, also indicated with $\epsilon(\boldsymbol{u})$, while the skew-symmetric tensor ∇^{W} is called the *spin tensor*.

For a Newtonian fluid, the stress tensor and the strain rate tensor are linearly related, and the stress-strain rate relationship is given by:

$$\sigma_{ij} = -p\delta_{ij} + s_{ij} = -p\delta_{ij} + \mu\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right) + \lambda\frac{\partial v_k}{\partial x_k}\delta_{ij}$$

where μ is the fluid dynamic viscosity and λ the so-called second coefficient of viscosity. For an incompressible flow $\nabla\cdot\boldsymbol{u} = 0$ (see 2.2.1), and consequently the previous relationship reduces to Stokes' law:

$$\sigma_{ij} = -p\delta_{ij} + \mu\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right) = -p\delta_{ij} + 2\mu v_{(i,j)}$$

which in a more compact form results:

$$\sigma(\boldsymbol{u}, p) = -p\mathbb{I} + 2\mu\epsilon(\boldsymbol{u}) \tag{2.5}$$

2.2 The incompressible Navier-Stokes equations

A number of important phenomena in fluid dynamics are described by the Navier-Stokes equations. These equations, which are the extension of the Euler equations to include viscous forces (and heat flow by conduction), are a statement of the dynamical effect of the externally applied forces and the internal forces of a fluid that we shall assume Newtonian. The Navier Stokes equations are derived from the conservation laws of mass and momentum, as shown as follow. We consider the motion of a fluid with density ρ in a domain $\Omega\subset\mathbb{R}^d$ (with $d = 2, 3$).

2.2.1 Conservation of mass

If $\rho(\boldsymbol{x}, t)$ is the density of a fluid at time t, then the mass m of the fluid is given by:

$$m = \int_{\Omega_t}\rho(\boldsymbol{x}, t)\,\mathrm{d}\boldsymbol{x} \tag{2.6}$$

Starting at time $t = 0$, we have some amount of fluid occupying the domain Ω_0. As time goes by, the same amount of fluid will occupy the domain Ω_t. Hence:

$$\int_{\Omega_0}\rho(\boldsymbol{x}, 0)\,\mathrm{d}\boldsymbol{x} = \int_{\Omega_t}\rho(\boldsymbol{x}, t)\,\mathrm{d}\boldsymbol{x} \quad \forall t\geq 0. \tag{2.7}$$

Since mass is constant in time, its derivative with respect to time must vanish:

$$\frac{d}{dt}\int_{\Omega_t} \rho(\boldsymbol{x},t)\,\mathrm{d}\boldsymbol{x} = 0. \tag{2.8}$$

Applying the transport theorem we have:

$$\int_{\Omega_t} \left(\frac{\partial \rho}{\partial t} + \nabla\cdot(\rho\boldsymbol{u})\right)(\boldsymbol{x},t)\,\mathrm{d}\boldsymbol{x} = 0 \quad \forall\,\boldsymbol{x}\in\Omega_t, \quad t\geq 0. \tag{2.9}$$

Since this is valid for arbitrary regions Ω_t, the integrand vanishes. This yields:

$$\frac{\partial \rho}{\partial t} + \nabla\cdot(\rho\boldsymbol{u}) = 0. \tag{2.10}$$

If the fluid is incompressible (its density is constant), the above equation results in:

$$\nabla\cdot\boldsymbol{u} = 0 \tag{2.11}$$

which is the continuity equation for incompressible fluids.

2.2.2 Conservation of momentum

The momentum $\boldsymbol{p}$ of a solid body is given by the product of its mass m and its velocity $\boldsymbol{u}$.

$$\boldsymbol{p}(t) = \int_{\Omega_t} \rho(\boldsymbol{x},t)\boldsymbol{u}(\boldsymbol{x},t)\,\mathrm{d}\boldsymbol{x}. \tag{2.12}$$

According to Newton's second law, the time derivative of momentum equals the total force applied on the body:

$$\frac{D}{Dt}\boldsymbol{p}(t) = \sum_i \boldsymbol{F_i} \tag{2.13}$$

where $\frac{D}{Dt}$ is the material derivative, defined by:

$$\frac{D}{Dt}(\clubsuit) = \frac{\partial}{\partial t}(\clubsuit) + (\boldsymbol{u}\cdot\nabla)(\clubsuit). \tag{2.14}$$

The forces acting on the fluid are body forces and surface forces. The first, e.g. gravity, can be expressed as:

$$\int_{\Omega_t} \rho(\boldsymbol{x},t)\boldsymbol{f}(\boldsymbol{x},t)\,\mathrm{d}\boldsymbol{x} \tag{2.15}$$

where $\boldsymbol{f}$ is a given force-density for unit volume. The latter, e.g. pressure and internal friction, can be represented by:

$$\int_{\partial\Omega_t} \sigma(\boldsymbol{x},t)\boldsymbol{n}\,\mathrm{d}S \tag{2.16}$$

where σ is a stress tensor which can be expressed as (2.17),

$$\begin{aligned}\sigma &= -p\mathbb{I} + 2\mu\epsilon \\ \epsilon &= \frac{1}{2}\left(\nabla\boldsymbol{u} + \nabla^\top\boldsymbol{u}\right)\end{aligned} \tag{2.17}$$

μ is the dynamic viscosity, $\boldsymbol{n}$ is the outward pointing unit normal vector and ϵ is the strain rate tensor.

Using the divergence theorem, the expression for the surface forces can be reformulated accordingly:

$$\int_{\partial\Omega_t} \sigma(\boldsymbol{x},t)\boldsymbol{n}\,\mathrm{d}S = \int_{\Omega_t} \nabla\cdot\sigma(\boldsymbol{x},t)\,\mathrm{d}\boldsymbol{x}. \tag{2.18}$$

Then, according to what we have just seen, the Newton's second law can be rewritten to:

$$\frac{D}{Dt}\int_{\Omega_t} \rho\boldsymbol{u}\,\mathrm{d}\boldsymbol{x} = \int_{\Omega_t} (\rho\boldsymbol{f} + \nabla\cdot\sigma)\,\mathrm{d}\boldsymbol{x}. \tag{2.19}$$

Using the formula for the material derivative results in:

$$\int_{\Omega_t} \left(\frac{\partial}{\partial t}(\rho\boldsymbol{u}) + (\boldsymbol{u}\cdot\nabla)(\rho\boldsymbol{u})\right)\mathrm{d}\boldsymbol{x} = \int_{\Omega_t} (\rho\boldsymbol{f} + \nabla\cdot\sigma)\,\mathrm{d}\boldsymbol{x}. \tag{2.20}$$

This applies for arbitrary Ω_t so we can remove the integrals, and considering incompressible fluids we obtain:

$$\frac{\partial\boldsymbol{u}}{\partial t} + (\boldsymbol{u}\cdot\nabla)\boldsymbol{u} - \frac{1}{\rho}(\nabla\cdot\sigma) = \boldsymbol{f} \tag{2.21}$$

which is the momentum equation for incompressible fluids.

2.2.3 N.S. Equations

The system (2.22) is the system of two equations governing incompressible flow of a viscous (Newtonian) fluid with constant kinematic viscosity $\nu > 0$, where the first equation is indicating the conservation of momentum while the latter the conservation of mass.

$$\begin{cases} \rho\left(\dfrac{\partial\boldsymbol{u}}{\partial t} + (\boldsymbol{u}\cdot\nabla)\boldsymbol{u}\right) = \nabla\cdot\sigma + \rho\boldsymbol{f}, & \boldsymbol{x}\in\Omega, t>0, \\ \nabla\cdot\boldsymbol{u} = 0, & \boldsymbol{x}\in\Omega, t>0. \end{cases} \tag{2.22}$$

Using Stokes' law (2.5), introducing the fluid kinematic viscosity $\nu = \frac{\mu}{\rho}$ and indicating hereafter with p the kinematic pressure, that is the pressure divided by density, and with σ the "dynamic" or "scaled" Cauchy stress tensor (normalized by density), i.d., $\sigma = -p\mathbb{I} + 2\nu\nabla^{\mathrm{S}}\boldsymbol{u}$, the first of the above equation can be rewritten as:

$$\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u}\cdot\nabla)\boldsymbol{u} - 2\nu\nabla\cdot\epsilon(\boldsymbol{u}) + \nabla p = \boldsymbol{f}. \tag{2.23}$$

We can observe that:

$$2\nu\nabla\cdot\epsilon(\boldsymbol{u}) = \nu\nabla\cdot(\nabla\boldsymbol{u} + \nabla^{\top}\boldsymbol{u}) = \nu\nabla^2\boldsymbol{u} + \underbrace{\nu\nabla(\nabla\cdot\boldsymbol{u})}_{=0}. \tag{2.24}$$

and the quantity representing the total fluid force can be expressed as

$$\nabla\cdot\sigma(\boldsymbol{u},p) = \nu\nabla^2\boldsymbol{u} - \nabla p$$

So, under the incompressibility condition the momentum equation can be transformed to:

$$\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u}\cdot\nabla)\boldsymbol{u} - \nu\nabla^2\boldsymbol{u} + \nabla p = \boldsymbol{f}. \tag{2.25}$$

The system (2.22) is composed by two equation (one is scalar and the other is vectorial) in two unknown functions $\boldsymbol{u}(\boldsymbol{x},t)$ and $p(\boldsymbol{x},t)$, while ρ is a given constant. So the system has many equations as unknowns and it can be resolved once the required initial and boundary conditions are given. Thus the problem must be completed with suitable initial and boundary conditions to form a well-posed initial boundary value problem. Typical boundary conditions consist of prescribing the value of the velocity on a portion Γ_{D} of the boundary:

$$\boldsymbol{u}(\boldsymbol{x},t) = \boldsymbol{g}(\boldsymbol{x},t), \quad \boldsymbol{x}\in\Gamma_{\mathrm{D}}, t>0 \tag{2.26}$$

and boundary traction $\boldsymbol{t}$ on the complementary portion Γ_{N}. The latter condition is a Neumann type boundary condition, and takes the form:

$$\boldsymbol{n}\cdot\sigma = -p\boldsymbol{n} + 2\nu\boldsymbol{n}\cdot\epsilon(\boldsymbol{u}) = \boldsymbol{t}, \quad \boldsymbol{x}\in\Gamma_{\mathrm{N}}, t>0. \tag{2.27}$$

where $\boldsymbol{n}$ is the unit outward normal to the boundary.

In the case of a time-dependent problem, the value of velocity field at the initial time $t=0$ must be given in Ω:

$$\boldsymbol{u}(\boldsymbol{x},0) = \boldsymbol{u}_0(\boldsymbol{x}), \quad \boldsymbol{x}\in\Omega. \tag{2.28}$$

Moreover, the initial velocity field must be divergence free.

The problem (2.22) can so be rewritten as:

$$\begin{cases} \dfrac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} - \nu \nabla^2 \boldsymbol{u} + \nabla p = \boldsymbol{f}, & \boldsymbol{x} \in \Omega, t > 0, \\ \nabla \cdot \boldsymbol{u} = 0, & \boldsymbol{x} \in \Omega, t > 0, \\ \boldsymbol{u}(\boldsymbol{x}, t) = \boldsymbol{g}(\boldsymbol{x}, t), & \boldsymbol{x} \in \Gamma_{\mathrm{D}}, t > 0, \\ \boldsymbol{n} \cdot \sigma = -p\boldsymbol{n} + 2\nu \boldsymbol{n} \cdot \epsilon(\boldsymbol{u}) = \boldsymbol{t}, & \boldsymbol{x} \in \Gamma_{\mathrm{N}}, t > 0, \\ \boldsymbol{u}(\boldsymbol{x}, 0) = \boldsymbol{u}_0(\boldsymbol{x}), & \boldsymbol{x} \in \Omega. \end{cases} \tag{2.29}$$

No initial condition must be specified for the fluid pressure, in fact no time derivative of pressure appears in the governing equations. When Dirichlet conditions are imposed everywhere (i.d., $\Gamma_{\mathrm{N}} = \emptyset$), pressure is determined only up to an arbitrary constant. In this case it is usual to impose the pressure average or its value at one point to uniquely define the pressure filed. We can observe that in the particular case of incompressible flows the two equations of mass and momentum conservation are enough to formulate a complete math problem, even if there are terms which represent viscous forces. Instead, for compressible flow, we have to add the equation of energy conservation.

The incompressibility condition may be a possible source of numerical difficulty. It consists of a constrain on the velocity field, which must be divergence free. Then, the pressure has to be considered as a variable not related to any constitutive equation. Its presence in the momentum equation has the purpose of introducing additional degree of freedom needed to satisfy the incompressibility constrain. The role of pressure variable is thus to adjust itself instantaneously in order to satisfy the condition of divergence-free velocity. This fact implies that $p(\boldsymbol{x}, t)$ acts as a *Lagrangian multiplier* of the incompressibility constrain $\nabla \cdot \boldsymbol{u} = 0$, which must be satisfied by the velocity $\boldsymbol{u}(\boldsymbol{x}, t)$ in any position $\boldsymbol{x}$ and in any time $t > 0$, and thus there is a coupling between the velocity and pressure unknowns. Various formulations have been proposed in literature to deal with incompressible flow problems. The two main approaches which address these issues are finite elements which satisfy the *LBB* or *inf-sup condition* or the use of stabilized finite element formulations. The first approach results in successful finite element methods. The second approach uses stabilized finite element formulations and provides more flexibility in the construction of finite element discretizations. This technique requires, however, the design of a stabilization operator or the enrichment of the finite element spaces with special functions, such as bubble functions.

2.3 Boundary Condition

2.3.1 Skin Friction Wall Model

Using a simple wall model such as the skin friction model presented in §Hoffman and Johnson [16] gives the chance to avoid resolving the turbulent boundary layer fully. The boundary condition model for a boundary Γ_{sfpr} with normal $\boldsymbol{n}$ and two orthogonal tangential vectors $\boldsymbol{\tau}_1$, $\boldsymbol{\tau}_2$ takes the form:

$$\begin{aligned} \boldsymbol{u} \cdot \boldsymbol{n} + \alpha\, \boldsymbol{n}^\top \sigma \boldsymbol{n} &= 0, \\ \boldsymbol{u} \cdot \boldsymbol{\tau}_k + \beta^{-1}\, \boldsymbol{n}^\top \sigma \boldsymbol{\tau}_k &= 0, \quad k = 1, 2. \end{aligned} \tag{2.30}$$

with the stress tensor σ and where we use matrix notation with all vectors $\boldsymbol{v}$ being column vectors and the corresponding row vector is denoted $\boldsymbol{v}^\top$.

The first equation of (2.30) is used to define the penetration of the boundary by modifying α, which in our case is always approx zero since the body is solid and the flow will not go through it. The second equation implements friction by the β parameter which corresponds to slip b.c. with $\beta = 0$ and nonslip b.c. with $\beta \to \infty$. By increasing β we increase the resistance at the boundary, and by increasing α we increase the penetration of the boundary.

2.3.2 Slip Boundary Condition

A slip boundary condition is when there is no friction or resistance of the boundary of the object against the flow moving on it. This condition corresponds to setting the normal component of the velocity $\boldsymbol{u} \cdot \boldsymbol{n} = 0$ at the boundary and models a boundary with negligible friction which the flow cannot penetrate. A slip boundary condition corresponds to $(\alpha, \beta) \to (0, 0)$ in formula (2.30).

2.3.3 No Slip Boundary Condition

The homogeneous Dirichlet velocity boundary condition $\boldsymbol{u} = \boldsymbol{0}$ is referred to as a no slip boundary condition expressing that the fluid adheres to the boundary. A non homogeneous Dirichlet boundary condition can be imposed to prescribe, e.g., a given inflow velocity. A no slip boundary condition corresponds to $(\alpha, \beta) \to (0, \infty)$ in formula (2.30).

2.3.4 Outflow Boundary Condition

To simulate an outflow boundary condition we may use a Neumann condition with $\boldsymbol{t} = \mathbf{0}$ in (2.27) corresponding to zero force at outflow as in outflow into a large empty reservoir. It acts as an approximate transparent outflow boundary condition, attempting to let the flow leave the domain with little obstruction (also referred to as a *do nothing* boundary condition).

2.4 Weak Formulation

In order to obtain the weak formulation for problem (2.29), denoting with $H^1(\Omega)$ the standard Hilbert space of functions that are square integrable together with their first order derivatives, let:

$$V_g := \left\{ \boldsymbol{v} \in (H^1(\Omega))^d \colon \boldsymbol{v}\big|_{\Gamma_\mathrm{D}} = \boldsymbol{g} \right\},$$

$$V_0 := \left\{ \boldsymbol{v} \in (H^1(\Omega))^d \colon \boldsymbol{v}\big|_{\Gamma_\mathrm{D}} = \mathbf{0} \right\},$$

$$Q := L^2(\Omega) = \left\{ f \colon \Omega \mapsto \mathbb{R} \colon \int_\Omega f(\boldsymbol{x})^2 \,\mathrm{d}\Omega < +\infty \right\}.$$

The inner product in $(L^2(\Omega))^d$, $d = 1, 2, 3$, is denoted by $(\cdot, \cdot)$ and the variational formulation is obtained in the usual way by multiplying (2.29) with a pair of test functions $(\boldsymbol{v}, q) \in (V_0, Q)$ and integrating momentum equation by parts. The presence of slip with linear friction and penetration with resistance boundary conditions on a portion Γ_sfpr of the boundary requires to use the deformation tensor formulation of the viscous term for the momentum equation:

$$\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} - 2\nu \nabla \cdot \epsilon(\boldsymbol{u}) + \nabla p = \boldsymbol{f} \tag{2.31}$$

Thus:

$$\begin{aligned} \int_\Omega \frac{\partial \boldsymbol{u}}{\partial t} \cdot \boldsymbol{v} \,d\Omega + \int_\Omega (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} \cdot \boldsymbol{v} \,\mathrm{d}\Omega - 2\nu \overbrace{\int_\Omega (\nabla \cdot \epsilon(\boldsymbol{u})) \cdot \boldsymbol{v} \,\mathrm{d}\Omega}^{\bigstar} + \\ + \underbrace{\int_\Omega \nabla p \cdot \boldsymbol{v} \,\mathrm{d}\Omega}_{\spadesuit} = \int_\Omega \boldsymbol{f} \cdot \boldsymbol{v} \,\mathrm{d}\Omega. \end{aligned} \tag{2.32}$$

where, using the Green's formula:

$$\begin{aligned} \bigstar &= -\int_{\Omega} \epsilon(\boldsymbol{u}) : \nabla \boldsymbol{v} \, \mathrm{d}\Omega + \int_{\partial\Omega} \epsilon(\boldsymbol{u})\boldsymbol{v} \cdot \boldsymbol{n} \, \mathrm{d}S \\ \spadesuit &= -\int_{\Omega} p \nabla \cdot \boldsymbol{v} \, \mathrm{d}\Omega + \int_{\partial\Omega} p\boldsymbol{v} \cdot \boldsymbol{n} \, \mathrm{d}S \end{aligned} \tag{2.33}$$

Using $\boldsymbol{u}_t$ to indicate $\frac{\partial \boldsymbol{u}}{\partial t}$ we can write:

$$\begin{aligned} &(\boldsymbol{u}_t, \boldsymbol{v}) + 2\nu(\epsilon(\boldsymbol{u}), \nabla \boldsymbol{v}) + ((\boldsymbol{u} \cdot \nabla)\boldsymbol{u}, \boldsymbol{v}) - (p, \nabla \cdot \boldsymbol{v}) = \\ &= \int_{\partial\Omega} \underbrace{(2\nu\epsilon(\boldsymbol{u}) - p\mathbb{I})}_{\sigma} \boldsymbol{n} \cdot \boldsymbol{v} \, \mathrm{d}S + (\boldsymbol{f}, \boldsymbol{v}) \end{aligned} \tag{2.34}$$

The symmetry of the deformation tensor yields:

$$\begin{aligned} (\epsilon(\boldsymbol{u}), \nabla \boldsymbol{v}) &= \left(\epsilon(\boldsymbol{u}), \frac{\nabla \boldsymbol{v}}{2}\right) + \left(\epsilon^{\top}(\boldsymbol{u}), \frac{\nabla^{\top} \boldsymbol{v}}{2}\right) = \\ &= \left(\epsilon(\boldsymbol{u}), \frac{\nabla \boldsymbol{v}}{2}\right) + \left(\epsilon(\boldsymbol{u}), \frac{\nabla^{\top} \boldsymbol{v}}{2}\right) = (\epsilon(\boldsymbol{u}), \epsilon(\boldsymbol{v})) \end{aligned} \tag{2.35}$$

Thus, the weak problem is to find $(\boldsymbol{u}, p) \in (V_g, Q)$ such that for all $(\boldsymbol{v}, q) \in (V_0, Q)$:

$$\begin{cases} (\boldsymbol{u}_t, \boldsymbol{v}) + 2\nu(\epsilon(\boldsymbol{u}), \epsilon(\boldsymbol{v})) + ((\boldsymbol{u} \cdot \nabla)\boldsymbol{u}, \boldsymbol{v}) - (p, \nabla \cdot \boldsymbol{v}) + \\ \qquad - \int_{\Gamma_{\text{sfpr}}} \sigma \boldsymbol{n} \cdot \boldsymbol{v} \, \mathrm{d}S = (\boldsymbol{f}, \boldsymbol{v}), \\ \qquad (\nabla \cdot \boldsymbol{u}, q) = 0 \end{cases} \tag{2.36}$$

where the boundary term reduces from $\partial\Omega = \Gamma_{\text{D}} \cup \Gamma_{\text{N}} \cup \Gamma_{\text{sfpr}}$ to only Γ_{sfpr} because $\boldsymbol{v} = \boldsymbol{0}$ on Γ_{D} and $\sigma \boldsymbol{n} = \boldsymbol{0}$ on Γ_{N}.
The boundary integral in the variational problem (2.36) can be rewritten by decomposing the test function $\boldsymbol{v}$ on Γ_{sfpr} into d orthonormal components, see §John [17]:

$$\boldsymbol{v} = (\boldsymbol{v} \cdot \boldsymbol{n})\boldsymbol{n} + \sum_{k=1}^{d-1} (\boldsymbol{v} \cdot \boldsymbol{\tau}_k)\boldsymbol{\tau}_k \tag{2.37}$$

This gives, using the definition of the slip with linear friction and penetration with resistance boundary condition:

$$\begin{aligned} &\int_{\Gamma_{\text{sfpr}}} \sigma \boldsymbol{n} \cdot \boldsymbol{v} \, \mathrm{d}S = \\ &= \int_{\Gamma_{\text{sfpr}}} (\boldsymbol{n}^{\top} \sigma \boldsymbol{n}) \boldsymbol{v} \cdot \boldsymbol{n} \, \mathrm{d}S + \int_{\Gamma_{\text{sfpr}}} \sum_{k=1}^{d-1} (\boldsymbol{n}^{\top} \sigma \boldsymbol{\tau}_k) \boldsymbol{v} \cdot \boldsymbol{\tau}_k \, \mathrm{d}S = \\ &= -\int_{\Gamma_{\text{sfpr}}} \alpha^{-1} (\boldsymbol{u} \cdot \boldsymbol{n})(\boldsymbol{v} \cdot \boldsymbol{n}) \, \mathrm{d}S - \int_{\Gamma_{\text{sfpr}}} \sum_{k=1}^{d-1} \beta (\boldsymbol{u} \cdot \boldsymbol{\tau}_k)(\boldsymbol{v} \cdot \boldsymbol{\tau}_k) \, \mathrm{d}S \end{aligned} \tag{2.38}$$

Thus, the variational problem can be reformulated: find $(\boldsymbol{u}, p) \in (V_g, Q)$ such that for all $(\boldsymbol{v}, q) \in (V_0, Q)$

$$\begin{cases} (\boldsymbol{u}_t, \boldsymbol{v}) + 2\nu(\epsilon(\boldsymbol{u}), \epsilon(\boldsymbol{v})) + ((\boldsymbol{u}\cdot\nabla)\boldsymbol{u}, \boldsymbol{v}) - (p, \nabla\cdot\boldsymbol{v}) + \\ \qquad + \int_{\Gamma_{\text{sfpr}}} \alpha^{-1}(\boldsymbol{u}\cdot\boldsymbol{n})(\boldsymbol{v}\cdot\boldsymbol{n})\,\mathrm{d}S + \\ \qquad + \int_{\Gamma_{\text{sfpr}}} \sum_{k=1}^{d-1} \beta(\boldsymbol{u}\cdot\boldsymbol{\tau}_k)(\boldsymbol{v}\cdot\boldsymbol{\tau}_k)\,\mathrm{d}S = (\boldsymbol{f}, \boldsymbol{v}), \\ (\nabla\cdot\boldsymbol{u}, q) = 0 \end{cases} \tag{2.39}$$

The boundary integrals which originate from the slip with friction and penetration with resistance boundary condition give a positive semi-definite contribution to the left-hand side of (2.39). The positivity follows from the positivity of and α and β. Since the boundary integrals become zero for functions which vanish on Γ_{sfpr}, they define a semi-definite operator.

3 G2 for Navier-Stokes equations

3.1 Introduction

To simulate turbulent flow one has to solve the Navier-Stokes equations. These solutions are inherently expensive to compute and the state of the art numerical methods can be divided into three different categories. First, for the most accurate simulation one has to rely on a direct numerical simulation (DNS), which solves the Navier-Stokes equations without any turbulence model. Hence, the method needs to resolve all scales of turbulent flow in order to be accurate, so it is too expensive for complex domains for which we are interested in. Thereby different methods has been proposed to get around the problem in which most of them use turbulent modeling which will introduce a model usually using an averaged N.S. equation. Two most common method used to get around this problem are RANS, Reynolds Averaged Navier Stokes, and LES, Large Eddy Simulations. In the first method, the effort is to solve a modified (averaged) Navier Stokes equations, with some approximate solution which will usually end up adding a term called Reynolds stress, to the modified equation. Formulated in late 1960s, LES uses the idea to only solve for large eddies of turbulent flow and using a subgrid model to model the smaller eddies. The idea behind this method comes from the self similarity theory formulated on 1941 by Andrey Kolmogorov.

An alternative approach is to seek functions that satisfy NSE only in an approximate weak sense. Such functions are the so-called ϵ-weak solutions to NSE, see §Hoffman and Johnson [16]. It is straight forward to construct such ϵ-weak solutions using stabilized Galerkin finite element methods, here referred to as General Galerkin G2 methods. A G2 method is a combination of a Galerkin method, assuring the residual to be small in average, and a weighted least squares stabilization, corresponding to a certain strong control of the residual. In this method we don't resolve all physical scales as oppose to DNS. Also, we won't resolve turbulent boundary layer so there is no need for turbulence modeling based on physics of unresolved scales as oppose to other turbulent modelings.

3.2 General Galerkin for Turbulent flow

The standard Galerkin finite element method is not stable for convection dominated problems: a mesh-dependent consistent numerical stabilization is thus added; the development of stabilization methods started in the late 70's. The simulation methodology used in this work for turbulent flow computations uses a finite element method with piecewise linear approximation in space and time, with numerical stabilization in the form of a weighted least squares method based on the residual. We refer to it as General Galerkin method (G2). The stabilization of G2 acts as an automatic turbulence model in the form of a generalized artificial viscosity model acting selectively on the smallest scales of the mesh.

3.3 Eulerian cG(1)cG(1) method

The stabilized cG(1)cG(1) method is a type of G2 method, with continuous piecewise linear trial functions both in time and space and continuous linear piecewise test functions in space and constant piecewise test functions in time. The stabilization term in the discretized system will introduce dissipation where the residual is large. This method has been explained in §Hoffman and Johnson [16]. We start by introducing the following notation: the scalar product over the finite element mesh $\mathcal{T}_n$ is defined as

$$(\boldsymbol{v}, \boldsymbol{w}) = \sum_{K \in \mathcal{T}_n} \int_K \boldsymbol{v} \cdot \boldsymbol{w} \, \mathrm{d}\boldsymbol{x} \tag{3.1}$$

We are looking for approximate solutions for velocity, $\boldsymbol{U}^n \equiv \boldsymbol{U}(t_n)$ and pressure, $P^n \equiv P(t_n)$, which will satisfy N.S. equations with homogeneous Dirichlet boundary conditions (without any loss in generality).

Let $0 = t_0 < t_1 < \cdots < t_N = T$ be a sequence of discrete time steps with associated time intervals $I_n = (t_{n-1}, t_n]$ of length $k_n = t_n - t_{n-1}$. Assuming the domain to be $S_n = \Omega \times I_n$, and let $W^n \subset H^1(\Omega)$ be a finite element space consisting of continuous piecewise linear functions on a mesh $\mathcal{T}_n = \{ K \}$ on Ω of mesh size $h_n(\boldsymbol{x})$ with W_0^n the functions in W^n satisfying the Dirichlet boundary condition $v|_{\Gamma_D} = 0$.

Choosing the test functions $v \in V_0^n \equiv [W_0^n]^3$ and $q \in W^n$, we can formulate the numerical approximation of the discretized system. We seek $\hat{\boldsymbol{U}} = (\boldsymbol{U}, P)$, continuous piecewise linear in space and time, and G2 in form of cG(1)cG(1) for NSE with homogeneous Dirichlet boundary conditions reads: for $n = 1, ..., N$, find $(\boldsymbol{U}^n, P^n) = (\boldsymbol{U}(t_n), P(t_n))$ with $\boldsymbol{U}^n \in$

V_0^n and $P^n \in W^n$, such that

$$\begin{gathered}((\boldsymbol{U}^n - \boldsymbol{U}^{n-1})k_n^{-1} + \bar{\boldsymbol{U}}^n \cdot \nabla \bar{\boldsymbol{U}}^n, v) + (2\nu\epsilon(\bar{\boldsymbol{U}}^n), \epsilon(\boldsymbol{v})) - (P^n, \nabla \cdot \boldsymbol{v}) + \\ +(\nabla \cdot \bar{\boldsymbol{U}}^n, q) + SD_\delta(\bar{\boldsymbol{U}}^n, P^n; \boldsymbol{v}, q) = (\boldsymbol{f}, \boldsymbol{v}) \quad \forall \hat{\boldsymbol{v}} = (\boldsymbol{v}, q) \in V_o^n \times W^n\end{gathered} \tag{3.2}$$

where $\bar{\boldsymbol{U}}^n = \frac{1}{2}(\boldsymbol{U}^n + \boldsymbol{U}^{n-1})$ and P^n are piecewise constant in time over I_n, with the stabilizing term

$$\begin{aligned}SD_\delta(\bar{\boldsymbol{U}}^n, P^n; \boldsymbol{v}, q) = (\delta_1(\bar{\boldsymbol{U}}^n \cdot \nabla \bar{\boldsymbol{U}}^n + \nabla P^n - \boldsymbol{f}), \bar{\boldsymbol{U}}^n \cdot \nabla \boldsymbol{v} + \nabla q) + \\ + (\delta_2 \nabla \cdot \bar{\boldsymbol{U}}^n, \nabla \cdot \boldsymbol{v})\end{aligned} \tag{3.3}$$

The stabilization parameters are set to $\delta_1 = \kappa_1(k_n^{-2} + |\boldsymbol{U}|^2 h_n^{-2})^{-\frac{1}{2}}$ and $\delta_2 = \kappa_2 h_n |\boldsymbol{U}|$ in the convection-dominated case; that is, if $\nu < |\boldsymbol{U}| h_n$. In the diffusion dominated case, the parameters are set to $\delta_1 = \kappa_1 h_n^2/(|\boldsymbol{U}|L)$ and $\delta_2 = \kappa_2 h_n^2 |\boldsymbol{U}|/L$. Here, κ_1 and κ_2 are positive constants of unit size and L the reference length. Furthermore

$$(\epsilon(\boldsymbol{v}), \epsilon(\boldsymbol{w})) = \sum_{i,j=1}^{3} (\epsilon_{ij}(\boldsymbol{v}), \epsilon_{ij}(\boldsymbol{w})) \tag{3.4}$$

This method corresponds to a second order accurate Crank-Nicolson time-stepping. We note that in the stabilizing the time derivative $\dot{\boldsymbol{U}}$ doesn't appear, which is a consequence of the piecewise constancy (in time) of the test functions.

If we have Neumann boundary conditions, we use the standard technique to apply these boundary conditions weakly. Instead, to implement the friction boundary conditions (2.30) we must add at the left hand side of (3.2) the term:

$$\alpha^{-1} \langle (\boldsymbol{U} \cdot \boldsymbol{n}), (\boldsymbol{v} \cdot \boldsymbol{n}) \rangle_{\Gamma_{\mathrm{sfpr}}} + \beta \langle \boldsymbol{U} \otimes \boldsymbol{v}, \mathbb{I} - \boldsymbol{n} \otimes \boldsymbol{n} \rangle_{\Gamma_{\mathrm{sfpr}}}$$

where $\otimes$ stands for the outer product and

$$\langle \boldsymbol{v}, \boldsymbol{w} \rangle_{\Gamma_{\mathrm{sfpr}}} = \int_{\Gamma_{\mathrm{sfpr}}} (\boldsymbol{v} : \boldsymbol{w}) \, \mathrm{d}S.$$

In fact, assuming an Euclidean space in $d = 3$ dimensions, recalling that for three orthonormal unit vector (e.g., $\boldsymbol{\tau}_1$, $\boldsymbol{\tau}_2$ and $\boldsymbol{n}$):

$$\boldsymbol{\tau}_1 \otimes \boldsymbol{\tau}_1 + \boldsymbol{\tau}_2 \otimes \boldsymbol{\tau}_2 + \boldsymbol{n} \otimes \boldsymbol{n} = \mathbb{I}$$

the summatory in (2.39) can be expressed as:

$$\begin{aligned}\sum_{k=1}^{2} (\boldsymbol{u} \cdot \boldsymbol{\tau}_k)(\boldsymbol{v} \cdot \boldsymbol{\tau}_k) = (\boldsymbol{u} \cdot \boldsymbol{\tau}_1)(\boldsymbol{v} \cdot \boldsymbol{\tau}_1) + (\boldsymbol{u} \cdot \boldsymbol{\tau}_2)(\boldsymbol{v} \cdot \boldsymbol{\tau}_2) = \\ = (\boldsymbol{u} \otimes \boldsymbol{v}) : (\boldsymbol{\tau}_1 \otimes \boldsymbol{\tau}_1 + \boldsymbol{\tau}_2 \otimes \boldsymbol{\tau}_2) = (\boldsymbol{u} \otimes \boldsymbol{v}) : (\mathbb{I} - \boldsymbol{n} \otimes \boldsymbol{n})\end{aligned}$$

3.4 G2 as adaptive DNS/LES

Turbulent solutions fluctuate rapidly in restricted regions, so these regions require a higher degree of resolution compared to the free stream region, but an increased resolution will also raise the computational cost. Therefore it seems natural to increase the resolution only in these restricted regions.

We can use cG(1)cG(1) to adaptively compute approximate solutions with the goal of satisfying a given tolerance with respect to the error in a specified output of interest. With respect to this quantity of interest, G2 chooses the mesh automatically based on “a posteriori” error estimation and resolves the flow features which has large influence on the quantity of interest while other scales remain unresolved with the stabilization acting as a numerical turbulence model. Therefore, adaptive G2 can be characterized as a DNS/LES method, since a part of the flow is resolved according to the quantity of interest as DNS and the rest of the flow is left unresolved in a LES.

We based the adaptive mesh refinement algorithm on the approach presented in §Rognes and Logg [24] to automated goal-oriented error control in the solution of nonlinear finite element variational problems, which is based on the automated solution of an auxiliary linearized adjoint (dual) problem, automated derivation and evaluation of “a posteriori” error estimates, and automated adaptive mesh refinement to control the error in a given goal functional to within a given tolerance. This allows the construction of adaptive algorithms that target a simulation to efficient computation of a specific quantity of interest. More precisely, a general variational problem of the following form were considered: find $\boldsymbol{u} \in V$ such that:

$$F(\boldsymbol{u}; \boldsymbol{v}) = 0 \quad \forall \boldsymbol{v} \in \hat{V}, \tag{3.5}$$

where $F\colon V \times \hat{V}\colon \to \mathbb{R}$ is a semilinear form (linear in $\boldsymbol{v}$) on a pair of trial and test spaces $(V, \hat{V})$. A general adaptive algorithm seeks to find an approximate solution $\boldsymbol{u}_h \approx \boldsymbol{u}$ of the variational problem (3.5) such that:

$$|M(\boldsymbol{u}) - M(\boldsymbol{u}_h)| < \varepsilon, \tag{3.6}$$

where $M\colon V \to \mathbb{R}$ is a given goal functional and $\varepsilon > 0$ is a given tolerance. The input to adaptive algorithm is the semilinear form F, the functional M, and the tolerance ε. Based on the given input, the adaptive algorithm automatically generates the dual problem, the “a posteriori” error estimate, and attempts to compute an approximate solution $\boldsymbol{u}_h$ that meets the given tolerance for the given functional. For a more detailed

Scheme 1: Adaptive DNS/LES

1. Given an initial coarse mesh $\mathcal{T}_0$, start at $n = 0$, then do
2. Compute approximation to the primal problem using $\mathcal{T}_n$.
3. Compute approximation to the dual problem using $\mathcal{T}_n$.
4. If $\sum_{K \in \mathcal{T}_n} \varepsilon_K < \text{TOL}$ then stop, else:
5. Mark a fraction P of the elements in $\mathcal{T}_n$ with highest ε_K for refinement.
6. Refine the mesh $\mathcal{T}_n$ using a standard mesh refinement algorithm, which gives a new refined mes $\mathcal{T}_{n+1}$.
7. Set $n = n + 1$ and goto 2.

discussion how the derivation of an "a posteriori" error estimate and corresponding error indicators from the error representation, and the derivation of the dual problem can be automated we refer to §Rognes and Logg [24].

An adaptive DNS/LES algorithm can then be stated as in Scheme 1; for this adaptive algorithm we constructed an adaptive solver following the framework illustrated in Figure 3.1.

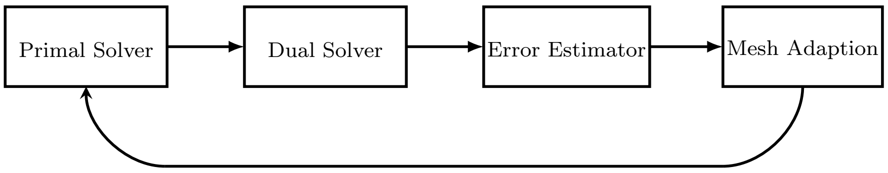


Figure 3.1: Overview of an adaptive finite element framework.

A key part for efficiency in the adaptive framework of cG(1)cG(1) is the ability to optimize the mesh according to some chosen flow quantity, represented by the data $M(\cdot)$ in the dual problem.

Finally, we have all the components we need to derive an adaptive finite element solver. G2 with skin friction boundary conditions gives us the ability to simulate turbulent flows with a simple wall layer model, avoiding resolution of the boundary layer. Adaptivity is a powerful tool that we can use to derive an efficient solver, that can be used to compute a solution to a large scale problem, on workstations as well as on massively parallel supercomputers.

Adaptive DNS/LES was developed and successfully applied to incompressible flow by §Hoffman [10, 11, 12, 13]; Hoffman and Johnson [14, 15]. For a detailed analysis of G2 we refer to §Hoffman and Johnson [16].

4 ALE Explained

The continuum mechanics usually make use of two classical descriptions of motion: the Lagrangian description and the Eulerian description. The arbitrary Lagrangian-Eulerian (in short ALE) description, which is the subject of the present chapter, was developed in an attempt to combine the advantages of the above cited classical kinematical descriptions, while minimizing as far as possible their respective drawbacks.

The differences between the Eulerian and the Lagrangian description are not significantly large. All that differentiates the two descriptions is basically the position of the observer. In the Lagrange description, also called the material coordinate system, the observer can be though of as attached to a particle and moves with the material. He records the changes of just this particle. For the Euler description, the point of observation is no longer fixed to the material, but fixed in space: the properties of not only one particle but of the whole material are now recorded. What follows is a more formal discussion taken from §Donea and Huerta [5]; Donea et al. [6], that provide an in-depth survey of ALE methods, including both conceptual aspects and numerical implementation details in view of applications in large deformation material response, fluid dynamics, nonlinear solid mechanics, and coupled fluid-structure problems. The notation has intentionally not been changed to that from §Donea et al. [6].

4.1 Lagrangian and Eulerian viewpoints

For the Lagrangian material description the domain $R_{\boldsymbol{X}}$ is given with spatial dimension made up of material particles $\boldsymbol{X}$. For the Eulerian configuration, $R_{\boldsymbol{x}}$ is the spatial domain consisting of spatial points $\boldsymbol{x}$. When deformation of the continuum occurs, the changes can be mapped to the spatial coordinates, written as a function $\boldsymbol{\varphi}$ which takes into account time t

$$\begin{aligned} \boldsymbol{\varphi}\colon R_{\boldsymbol{X}} \times [t_0, t_{\text{end}}[&\to R_{\boldsymbol{x}} \times [t_0, t_{\text{end}}[\\ (\boldsymbol{X}, t) &\mapsto \boldsymbol{\varphi}(\boldsymbol{X}, t) = (\boldsymbol{x}, t) \end{aligned}$$

and so $\boldsymbol{x}$ can be written as a function of $\boldsymbol{X}$ and time t:

$$\boldsymbol{x} = \boldsymbol{x}(\boldsymbol{X}, t), \quad t = t$$

which states that the spatial coordinates $\boldsymbol{x}$ depend both on the material particle $\boldsymbol{X}$ and time t, and that the physical time is measured by the same variable t in both material and spatial domains. It's convenient to employ a matrix representation for the gradient of $\boldsymbol{\varphi}$:

$$\frac{\partial \boldsymbol{\varphi}}{\partial (\boldsymbol{X}, t)} = \begin{pmatrix} \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{X}} & \boldsymbol{v} \\ \boldsymbol{0}^\top & 1 \end{pmatrix}$$

where the material velocity $\boldsymbol{v}$ is

$$\boldsymbol{v}(\boldsymbol{X}, t) = \frac{\partial \boldsymbol{x}}{\partial t}\Big|_{\boldsymbol{X}} \tag{4.1}$$

with $|_{\boldsymbol{X}}$ meaning holding the material coordinate $\boldsymbol{X}$ fixed.

Since $\boldsymbol{\varphi}$ must verify $\det(\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{X}}) > 0$ (non zero to impose a one-to-one correspondence and positive to avoid orientation change of the reference axes) at each point $\boldsymbol{X}$ and instant $t > t_0$, we are able to keep track of the history of motion and, by the inverse transformation $(\boldsymbol{X}, t) = \boldsymbol{\varphi}^{-1}(\boldsymbol{x}, t)$, to identify at any instant the initial position of the material particle occupying position $\boldsymbol{x}$ at time t.

In an Eulerian description the finite element mesh is thus fixed and the continuum moves and deforms with respect to the computational grid. The velocity $\boldsymbol{v}$ is consequently expressed with respect to the fixed element mesh without any reference to the initial configuration of the continuum and the material coordinates $\boldsymbol{X}$: $\boldsymbol{v} = \boldsymbol{v}(\boldsymbol{x}, t)$.

4.2 ALE kinematic description

ALE, as its name suggests, is a combination of the Lagrangian and Eulerian descriptions. It provides a means to do calculations using both descriptions with the use of a third coordinate system, the referential coordinate system denoted $R_{\boldsymbol{\chi}}$ where reference coordinates $\boldsymbol{\chi}$ are introduced to identify the grid points. For a more thorough understanding, we will once again refer to §Donea et al. [6]. Figure 4.1 shows these domains and the one-to-one transformations relating the configurations. The referential domain $R_{\boldsymbol{\chi}}$ is mapped into the material and spatial domains by $\boldsymbol{\Psi}$ and $\boldsymbol{\Phi}$ respectively. The particle motion $\boldsymbol{\varphi}$ may then be expressed as $\boldsymbol{\varphi} = \boldsymbol{\Phi} \circ \boldsymbol{\Psi}^{-1}$, clearly showing that, of course, the three mappings $\boldsymbol{\Psi}$, $\boldsymbol{\Phi}$ and $\boldsymbol{\varphi}$ are not independent.

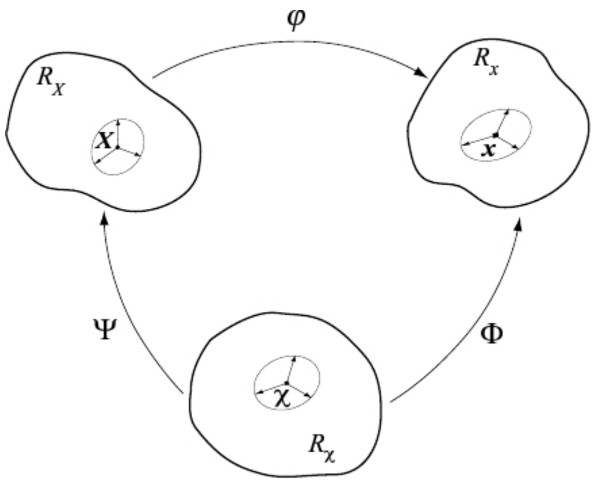


Figure 4.1: Lagrangian, Eulerian and ALE domains: illustration of the different configurations. Taken from §Donea et al. [6]

The mapping $\boldsymbol{\Phi}$ from the referential domain to the spatial domain, which can be understood as the motion of the grid points in the spatial domain, is represented by

$$\begin{aligned} \boldsymbol{\Phi}\colon R_{\boldsymbol{\chi}} \times [t_0, t_{\text{end}}[&\to R_{\boldsymbol{x}} \times [t_0, t_{\text{end}}[\\ (\boldsymbol{\chi}, t) &\mapsto \boldsymbol{\Phi}(\boldsymbol{\chi}, t) = (\boldsymbol{x}, t) \end{aligned}$$

and its gradient is

$$\frac{\partial \boldsymbol{\Phi}}{\partial(\boldsymbol{\chi}, t)} = \begin{pmatrix} \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\chi}} & \hat{\boldsymbol{v}} \\ \mathbf{0}^\top & 1 \end{pmatrix}, \quad \hat{\boldsymbol{v}}(\boldsymbol{\chi}, t) = \frac{\partial \boldsymbol{x}}{\partial t}\Big|_{\boldsymbol{\chi}} \tag{4.2}$$

where $\hat{\boldsymbol{v}}$ is the mesh velocity.

Finally, regarding $\boldsymbol{\Psi}$, it is convenient to represent directly its inverse $\boldsymbol{\Psi}^{-1}$,

$$\begin{aligned} \boldsymbol{\Psi}^{-1}\colon R_{\boldsymbol{X}} \times [t_0, t_{\text{end}}[&\to R_{\boldsymbol{\chi}} \times [t_0, t_{\text{end}}[\\ (\boldsymbol{X}, t) &\mapsto \boldsymbol{\Psi}^{-1}(\boldsymbol{X}, t) = (\boldsymbol{\chi}, t) \end{aligned}$$

and its gradient is

$$\frac{\partial \boldsymbol{\Psi}^{-1}}{\partial(\boldsymbol{X}, t)} = \begin{pmatrix} \frac{\partial \boldsymbol{\chi}}{\partial \boldsymbol{X}} & \boldsymbol{w} \\ \mathbf{0}^\top & 1 \end{pmatrix}, \quad \boldsymbol{w}(\boldsymbol{\chi}, t) = \frac{\partial \boldsymbol{\chi}}{\partial t}\Big|_{\boldsymbol{X}} \tag{4.3}$$

where $\boldsymbol{w}$ is the material particle velocity in the referential domain, since it measures the time variation of referential coordinates $\boldsymbol{\chi}$ holding the material particle fixed.

The relation between velocities $\boldsymbol{v}$, $\hat{\boldsymbol{v}}$ and $\boldsymbol{w}$ can be obtained by differentiating $\boldsymbol{\varphi} = \boldsymbol{\Phi} \circ \boldsymbol{\Psi}^{-1}$, in matrix format:

$$\begin{pmatrix} \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{X}} & \boldsymbol{v} \\ \boldsymbol{0}^\top & 1 \end{pmatrix} = \begin{pmatrix} \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\chi}} & \hat{\boldsymbol{v}} \\ \boldsymbol{0}^\top & 1 \end{pmatrix} \begin{pmatrix} \frac{\partial \boldsymbol{\chi}}{\partial \boldsymbol{X}} & \boldsymbol{w} \\ \boldsymbol{0}^\top & 1 \end{pmatrix}$$

which yields, after rearranging:

$$\boldsymbol{c} = \boldsymbol{v} - \hat{\boldsymbol{v}} = \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\chi}} \cdot \boldsymbol{w}$$

thus defining the convective velocity $\boldsymbol{c}$, that is, the relative velocity between the material and the mesh. The convective velocity $\boldsymbol{c}$ should not be confused with $\boldsymbol{w}$, that is the particle velocity as seen from the referential domain $R_{\boldsymbol{\chi}}$, whereas $\boldsymbol{c}$ is the particle velocity relative to the mesh as seen from the spatial domain $R_{\boldsymbol{x}}$ (both $\boldsymbol{v}$ and $\hat{\boldsymbol{v}}$ are variations of coordinate $\boldsymbol{x}$). In fact $\boldsymbol{c} = \boldsymbol{w}$ if and only if $\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\chi}} = \mathbb{I}$, that is, when the mesh motion is purely translational, without rotations or deformations of any kind.

With the choice $\boldsymbol{\Psi} = \boldsymbol{I}$ we obtain $\boldsymbol{X} = \boldsymbol{\chi}$ and a Lagrangian description results: the material and mesh velocities coincide and the convective velocity $\boldsymbol{c}$ is null. If $\boldsymbol{\Phi} = \boldsymbol{I}$ we obtain $\boldsymbol{x} = \boldsymbol{\chi}$ implying an Eulerian description: a null mesh velocity is obtain and the convective velocity $\boldsymbol{c}$ is identical to the material velocity $\boldsymbol{v}$.

4.3 The foundamental ALE equation

In order to relate the time derivative in the material, spatial and referential domains, let a scalar physical quantity be described by $f(\boldsymbol{x}, t)$, $f^\star(\boldsymbol{\chi}, t)$ and $f^{\star\star}(\boldsymbol{X}, t)$ in the spatial, referential and material domains, respectively. Stars are employed to emphasize that the functional forms are, in general, different.

Since $f^{\star\star}(\boldsymbol{X}, t) = f(\boldsymbol{\varphi}(\boldsymbol{x}, t), t)$, by taking the gradient of this expression, we obtain the following equation

$$\begin{pmatrix} \frac{\partial f^{\star\star}}{\partial \boldsymbol{X}} & \frac{\partial f^{\star\star}}{\partial t} \end{pmatrix} = \begin{pmatrix} \frac{\partial f}{\partial \boldsymbol{x}} & \frac{\partial f}{\partial t} \end{pmatrix} \begin{pmatrix} \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{X}} & \boldsymbol{v} \\ \boldsymbol{0}^\top & 1 \end{pmatrix} \quad \text{or} \quad \begin{cases} \dfrac{\partial f^{\star\star}}{\partial \boldsymbol{X}} = \dfrac{\partial f}{\partial \boldsymbol{x}} \dfrac{\partial \boldsymbol{x}}{\partial \boldsymbol{X}} \\ \dfrac{\partial f^{\star\star}}{\partial t} = \dfrac{\partial f}{\partial t} + \dfrac{\partial f}{\partial \boldsymbol{x}} \cdot \boldsymbol{v} \end{cases} \tag{4.4}$$

Note that the second of (4.4) is the well known equation that relates the material and the spatial time derivatives. Dropping the stars to ease the notation, this relation is finally cast as

$$\left.\frac{\partial f}{\partial t}\right|_{\boldsymbol{X}} = \left.\frac{\partial f}{\partial t}\right|_{\boldsymbol{x}} + \boldsymbol{v}\cdot\nabla f \quad \text{or} \quad \frac{df}{dt} = \frac{\partial f}{\partial t} + \boldsymbol{v}\cdot\nabla f$$

This states that the variation of a physical quantity for a given particle $\boldsymbol{X}$ is the local variation plus a convective term taking into account the relative motion between the material and spatial systems.

In a similar way, the relation between material and spatial time derivatives can be extended to include the referential time derivative and, considering that $f^{\star\star} = f^{\star}\circ\boldsymbol{\Psi}^{-1}$, we obtain

$$\begin{pmatrix} \frac{\partial f^{\star\star}}{\partial \boldsymbol{X}} & \frac{\partial f^{\star\star}}{\partial t} \end{pmatrix} = \begin{pmatrix} \frac{\partial f^{\star}}{\partial \boldsymbol{\chi}} & \frac{\partial f^{\star}}{\partial t} \end{pmatrix} \begin{pmatrix} \frac{\partial \boldsymbol{\chi}}{\partial \boldsymbol{X}} & \boldsymbol{w} \\ \boldsymbol{0}^{\top} & 1 \end{pmatrix} \quad \text{or} \quad \begin{cases} \dfrac{\partial f^{\star\star}}{\partial \boldsymbol{X}} = \dfrac{\partial f^{\star}}{\partial \boldsymbol{\chi}}\dfrac{\partial \boldsymbol{\chi}}{\partial \boldsymbol{X}} \\ \dfrac{\partial f^{\star\star}}{\partial t} = \dfrac{\partial f^{\star}}{\partial t} + \dfrac{\partial f^{\star}}{\partial \boldsymbol{\chi}}\cdot\boldsymbol{w} \end{cases} \tag{4.5}$$

Using the definition of $\boldsymbol{w}$ we can rearrange the second of (4.5) into

$$\frac{\partial f^{\star\star}}{\partial t} = \frac{\partial f^{\star}}{\partial t} + \frac{\partial f}{\partial \boldsymbol{x}}\cdot\boldsymbol{c}$$

The fundamental ALE relation between material time derivatives, referential time derivatives and spatial gradient is finally cast as (stars dropped)

$$\left.\frac{\partial f}{\partial t}\right|_{\boldsymbol{X}} = \left.\frac{\partial f}{\partial t}\right|_{\boldsymbol{\chi}} + \frac{\partial f}{\partial \boldsymbol{x}}\cdot\boldsymbol{c} = \left.\frac{\partial f}{\partial t}\right|_{\boldsymbol{\chi}} + \boldsymbol{c}\cdot\nabla f$$

and shows that the time derivative of the physical quantity f for a given particle $\boldsymbol{X}$, that is, its material derivative, is its local derivative (with the reference coordinate $\boldsymbol{\chi}$ held fixed) plus a convective term taking into account the relative velocity $\boldsymbol{c}$ between the material and the reference system.

4.4 ALE form of conservation equations

The ALE differential form of the conservation equations for mass and momentum are readily obtained from the corresponding well-known Eulerian forms by replacing in the various convective terms the material velocity $\boldsymbol{v}$ with the convective velocity $\boldsymbol{c}$. The result is, for the equation of conservation of mass

$$\left.\frac{\partial \rho}{\partial t}\right|_{\boldsymbol{\chi}} + \boldsymbol{c}\cdot\nabla\rho = -\rho\nabla\cdot\boldsymbol{v} \tag{4.6}$$

and for momentum

$$\rho \left(\frac{\partial \boldsymbol{v}}{\partial t}\Big|_{\boldsymbol{\chi}} + (\boldsymbol{c} \cdot \nabla)\boldsymbol{v} \right) = \nabla \cdot \sigma + \rho \boldsymbol{b} \tag{4.7}$$

The starting point for deriving the ALE integral form of the conservation equations is Reynolds transport theorem applied to an arbitrary volume V_t whose boundary $S_t = \partial V_t$ moves with the mesh velocity $\hat{\boldsymbol{v}}$

$$\frac{\partial}{\partial t}\Big|_{\boldsymbol{\chi}} \int_{V_t} f(\boldsymbol{x}, t)\, \mathrm{d}\Omega = \int_{V_t} \frac{\partial f(\boldsymbol{x}, t)}{\partial t}\Big|_{\boldsymbol{x}} \mathrm{d}\Omega + \int_{S_t} f(\boldsymbol{x}, t) \hat{\boldsymbol{v}} \cdot \boldsymbol{n}\, \mathrm{d}S \tag{4.8}$$

We then successively replace the scalar $f(\boldsymbol{x}, t)$ by the fluid density ρ and momentum $\rho\boldsymbol{v}$ to obtain the ALE integral forms of conservation equation.

We can now implement ALE with Navier-Stokes equations by just taking into account the mesh movement $\hat{\boldsymbol{v}}$ alongside the existing velocity $\boldsymbol{v}$ from above, in this case $\boldsymbol{u}$. To illustrate this small change, here are the new ALE incompressible Navier-Stokes equations.

$$\begin{cases} \boldsymbol{u}_t + ((\boldsymbol{u} - \hat{\boldsymbol{v}}) \cdot \nabla)\boldsymbol{u} - \nu \nabla^2 \boldsymbol{u} + \nabla p = \boldsymbol{f}, & \boldsymbol{x} \in \Omega, t > 0, \\ \nabla \cdot \boldsymbol{u} = 0, & \boldsymbol{x} \in \Omega, t > 0 \end{cases} \tag{4.9}$$

Since boundary conditions are related to the problem, not to the description employed, the same boundary conditions employed in Eulerian or Lagrangian descriptions are implemented in the ALE formulation. We may also model friction with respect to a non zero velocity $\hat{\boldsymbol{v}}$ at the boundary simply by changing (2.30) into

$$\begin{aligned} (\boldsymbol{u} - \hat{\boldsymbol{v}}) \cdot \boldsymbol{n} + \alpha\, \boldsymbol{n}^\top \sigma \boldsymbol{n} &= 0, \\ (\boldsymbol{u} - \hat{\boldsymbol{v}}) \cdot \boldsymbol{\tau}_k + \beta^{-1}\, \boldsymbol{n}^\top \sigma \boldsymbol{\tau}_k &= 0, \quad k = 1, 2. \end{aligned} \tag{4.10}$$

4.4.1 Mesh moving

While ALE allows a moving mesh, the velocity of the mesh still has to be determined. In the case of FSI, in the structure part the mesh velocity was known because it was just the calculated velocity of the material, i.d., the Lagrangian description was used. For the rest of the domain, i.d., the fluid part, the movement of the mesh was determined by some mesh smoothing criterion: the mesh at the interface between the solid and the fluid becomes deformed by the mesh movement. Therefore the vertices of the mesh belonging to the fluid domain must be redistributed. This is done by solving a “fictitious” elastic problem: we determinate the displacement and velocity of the vertices of the fluid mesh, once we known the displacement and velocity of the vertices at the fluid-structure interface, by creating an elastic problem as follow

$$a(\boldsymbol{u}, \boldsymbol{v}) = L(\boldsymbol{v}) \tag{4.11}$$

where

$$\begin{aligned} a(\boldsymbol{u}, \boldsymbol{v}) &= \int_\Omega \sigma(\boldsymbol{u}) : \epsilon(\boldsymbol{v}) \, \mathrm{d}\Omega \\ L(\boldsymbol{v}) &= \int_\Omega \boldsymbol{f} \cdot \boldsymbol{v} \, \mathrm{d}\Omega \end{aligned}$$

Here, $\epsilon(\boldsymbol{v}) = (\nabla \boldsymbol{v} + \nabla^\top \boldsymbol{v})/2$ denotes the symmetric gradient and $\sigma(\boldsymbol{v}) = 2\mu\epsilon(\boldsymbol{v}) + \lambda \mathrm{Tr}(\epsilon(\boldsymbol{v}))\mathbb{I}$ is the stress tensor. To solve the linear elastic problem (4.11) for a specific choice of parameter values (the Lamé constants μ and λ), we define an elastic module for each element proportional to the inverse of the element volume and, by imposing homogeneous Dirichlet boundary conditions at external boundaries and imposing on the internal boundary (interface), at first the known displacement and then the velocity of its vertices, we can find the displacement and the velocity of all fluid mesh vertices, assembling just once the variational forms.

The implemented code to solve the linear elastic problem is shown in Figure 4.2 and it is now briefly explained: the FEniCS tools used are imported from the `dolfin` package, which defines classes like `Mesh`, `DirichletBC`, `FunctionSpace`, `TrialFunction`, `TestFunction`, and key functions such as `assemble` and `solve`. See Chapter 7 for more details of FEniCS project.
We first load a mesh and boundary indicators from files (see Chapter 7.2.3). For brevity, the code for specifying Dirichlet conditions is omitted (the boundary conditions are assumed to be available as lists in the program: we imposed homogeneous Dirichlet boundary conditions at external boundaries while at internal boundary of the fluid mesh, i.d., interface, we imposed the displacements, `bcs`, and then the velocities, `bscp`, of the nodes). The type of discrete function space is defined in terms of a mesh, a class of finite element (here `CG` means standard continuous Lagrange finite elements, while `DG` means discontinuous Lagrange finite elements) and a polynomial degree. The variational problem is expressed in terms of the Unified Form Language (UFL), which is another component of FEniCS, see Chapter 7.1.4. Terms multiplied by `dx` correspond to volume integrals, while multiplication by `ds` implies a boundary integral. Meshes may include several sub-domains and boundary segments, each with its corresponding volume or boundary integral. The Dirichlet boundary conditions may then be enforced as part of the

Python code

```python
from dolfin import *
mesh=Mesh('mesh.xml')
bfc = mesh.data().mesh_function('bfc')
V = VectorFunctionSpace(mesh, "CG", 1)
u = TrialFunction(V)
v = TestFunction(V)
f = Constant((0.0, 0.0, 0.0))
DG = FunctionSpace(mesh, "DG", 0)
E = Function(DG)
for c in cells(mesh):
     E.vector()[c.index()]=1./c.volume()
nu = 0.0
mu = E / (2.0*(1.0 + nu))
lmbda = E*nu / ((1.0 + nu)*(1.0 - 2.0*nu))
def sigma(v):
   return 2.0*mu*sym(grad(v)) +
      lmbda*tr(sym(grad(v)))*Identity(v.cell().d)
a = inner(sigma(u), sym(grad(v)))*dx
L = inner(f, v)*dx
A = assemble(a)
b = assemble(L)
# set Dirichlet conditions for displacements
[bc.apply(A, b) for bc in bcs]
u = Function(V)
solve(A, u.vector(), b, 'gmres', 'ilu')
# set Dirichlet conditions for velocities
u_p = Function(V)
[bc.apply(A,b) for bc in bcsp]
solve(A, u_p.vector(), b, 'gmres', 'ilu')
```

Figure 4.2: Elastic problem for mesh moving.

linear system $AU = b$ by the call `bc.apply(A, b)`. Finally, we solve the linear system using the generalized minimal residual method (`gmres`) with ILU preconditioning (`ilu`).

A control if *negative volumes* occur is also implemented: if the displacement of the interface nodes causes some elements to become negative volume elements, then the elastic module of these element was increased and the linear elastic problem was resolved again. As example, we considered the computational domain given by the rectangle $[-2, 5] \times [-3, 3]$, with an unit chord NACA 0012 airfoil (the airfoil center is placed at the origin). This mesh has been used for computations in Chapter 9, see Figure 9.1. The mesh distortion when the airfoil is rigidly moved or rotated around its center is shown in Figure 4.3: as we can see, since elastic module of a element is proportional to the inverse of its volume (or area in 2D case), the coarser is the mesh, the larger is the distortion of its elements. This method has been tested working for large displacements and rotations.

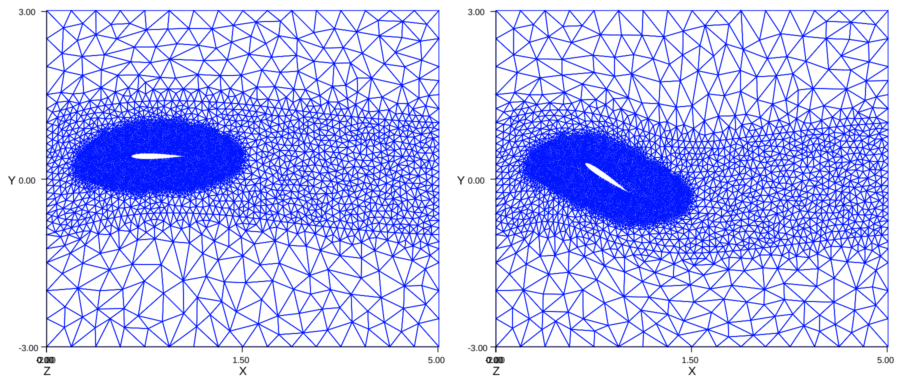


Figure 4.3: Mesh distortion when airfoil is rigidly moved (left) or rotated around its center (right).

5 Structural Multibody Modeling

This chapter briefly describes the multibody modeling technique adopted in MBDyn (see `http://www.aero.polimi.it/mbdyn/`), an Open Source multibody-multidisciplinary code developed at the "Dipartimento di Ingegneria Aerospaziale" of Politecnico di Milano University, which has been used as the core of the coupled simulation. For a more detailed review of MBDyn see §Masarati [21].

We can use the principles of classical mechanics (i.d., Newton's laws) to describe the motion of a system of rigid/deformable bodies. They may lead towards two different set approaches:

- a redundant coordinate set approach, in which the constraints between inertial bodies are explicitly expressed

- a minimal coordinate set approach where all the constraints are eliminated

The redundant approach can be used to easily and automatically generate the equations that model complex systems by means of the multibody formalism.

5.1 Multibody Formulation

A multibody system can be defined as a system of bodies, connected by algebraic constraints and deformable elements, in arbitrary motion with respect to each other. These bodies can also be subjected to interactional forces (e.g., aerodynamic forces). The advantage of the redundant coordinate set approach is a simple formulation, since it is possible to write the free-body dynamics equations for each body in the global reference frame with the addition of the constraint equations, which results in a easy and versatile implementation. The drawback is a larger amount of degrees of freedom, compared to that of the reduced coordinate set formulation.

Appropriate algebraic equations are used to impose algebraic constraints, which result in *reaction forces* as unknowns in a way that resembles the Lagrangian Multipliers method. This solution was adopted in order to simplify the writing of equations and reduce the computational efforts.

The system that describe the dynamics of the bodies is written by coupling the definition of the momentum, $\boldsymbol{\beta}$, and of the momenta moment, $\boldsymbol{\Gamma}$, of each body to the force and moment equilibrium equations. Denoting with $\boldsymbol{S}$ the static moments vector and with J the inertia moments matrix, the resulting equations for each body is

$$\begin{aligned} \boldsymbol{\beta} &= m\dot{\boldsymbol{x}} + \boldsymbol{\omega} \times \boldsymbol{S}, \\ \boldsymbol{\Gamma} &= \boldsymbol{S} \times \dot{\boldsymbol{x}} + J\boldsymbol{\omega}, \\ \dot{\boldsymbol{\beta}} &= \boldsymbol{F}(\boldsymbol{x}, \dot{\boldsymbol{x}}, \boldsymbol{R}, \boldsymbol{\omega}, \boldsymbol{a}, \dot{\boldsymbol{a}}, \dots, t) + \boldsymbol{V}_{\mathrm{F}}, \\ \dot{\boldsymbol{\Gamma}} + \dot{\boldsymbol{x}} \times \boldsymbol{\beta} &= \boldsymbol{C}(\boldsymbol{x}, \dot{\boldsymbol{x}}, \boldsymbol{R}, \boldsymbol{\omega}, \boldsymbol{a}, \dot{\boldsymbol{a}}, \dots, t) + \boldsymbol{V}_{\mathrm{C}} \end{aligned} \tag{5.1}$$

where $\boldsymbol{x}$ represents the position vector, $\boldsymbol{R}$ the orientation matrix, $\boldsymbol{a}$ the internal states, $\boldsymbol{\omega}$ the angular velocity, $\boldsymbol{F}$ the forces and $\boldsymbol{C}$ the couples. The forces $\boldsymbol{V}_{\mathrm{F}}$ and $\boldsymbol{V}_{\mathrm{C}}$ are related to the Lagrangian multipliers that represent the actual reaction forces and couples generated by the constraints. Kinematic constraints, as we said above, are added as algebraic/differential equations: i.d., $\boldsymbol{\Psi}(\boldsymbol{x}, \boldsymbol{R}, \dots, t) = 0$.

By calling $\boldsymbol{y}$, $\boldsymbol{z}$ and $\boldsymbol{\lambda}$ the kinematic unknowns, the momentum unknowns and the algebraic Lagrangian multipliers unknowns, respectively, the resulting system becomes

$$\begin{aligned} &\boldsymbol{M}(\boldsymbol{y}, t)\dot{\boldsymbol{y}} = \boldsymbol{z} \\ &\dot{\boldsymbol{z}} = \boldsymbol{Q}(\boldsymbol{y}, \dot{\boldsymbol{y}}, t) - \boldsymbol{G}^{\top}\boldsymbol{\lambda} \\ &\boldsymbol{\Psi}(\boldsymbol{y}, t) = 0 \end{aligned} \tag{5.2}$$

Here, $\boldsymbol{M}$ is a configuration dependent inertia matrix, $\boldsymbol{Q}$ are arbitrary external forces and couples and $\boldsymbol{G} = \nabla_{\boldsymbol{y}}\boldsymbol{\Psi}$ is the derivative of the (holonomic) constraints with respect to the kinematic unknowns.

We emphasize again that the unknowns are $\boldsymbol{x}, \boldsymbol{R}, \boldsymbol{\beta}, \boldsymbol{\Gamma}, \boldsymbol{V}_{\mathrm{F}}, \boldsymbol{V}_{\mathrm{C}}$, and that the overall system is solved without any further substitution, except for the orthonormal matrix $\boldsymbol{R}$ (that describes the orientation of a local frame since it maps vectors from the local to the global frame), which is expressed in terms of the rotation parameters as detailed in §Quaranta [22], where, in order to obtain an accurate as well as efficient solution, an appropriate formulation for the rotations is discussed.

The parametrization of large rigid rotations, which are required because the formulation of a multibody problem is based on the kinematic description of large displacements and rotations, requires at least three unknowns, but in order to avoid singularities that arise when the sign of the rotation cannot be uniquely determined, four parameters are needed. If only incremental rotations, which is assumed to be small enough to avoid any singularities, are considered, we can prevent the introduction of further unknowns and the need to resort to the four parameter rotation. The rotation matrix $\boldsymbol{R}$ is expressed in terms of the Gibbs-Rodriguez parameters $\boldsymbol{g}$, slightly modified with respect to the conventional notation, in order to make the linearized expressions of the rotational entities match those related to the rotational vector $\boldsymbol{\varphi}$ (that describes a finite rotation about an axis embodied by the vector itself, whose amplitude is given by the modulus of the vector).

5.2 Structural Model

In order to analyze the fluid-structure coupling problem defined by flexible flapping wings, a consistent geometrically nonlinear four-node shell element formulation, implemented and validated in §Quaranta et al. [23], within the multibody environment provided by the free general-purpose multibody solver MBDyn, has been adopted for our computations. Here the 4-node shell element formulation is briefly explained. It's based on a combination of the Enhanced Assumed Strain (EAS) and Assumed Natural Strain (ANS) formulations.

Let $\boldsymbol{y}$ be the position of the shell reference surface, and define a local orthogonal coordinate system on the undeformed shell surface. Let $\boldsymbol{T}$ also be a local orthonormal triad defined on the surface. By comparing the deformed and undeformed back-rotated derivatives of the position, two Biot-like linear deformation vectors can be computed, i.d.,

$$\tilde{\boldsymbol{\epsilon}}_k = \boldsymbol{T}^\top \boldsymbol{y}_{/k} - \boldsymbol{T}_0^\top \boldsymbol{y}_{0/k} \tag{5.3}$$

where $(\cdot)_{/k}$ indicates the partial derivative with respect to the arc length coordinate k and the subscript $(\cdot)_0$ identifies the undeformed configuration. Vectors $\tilde{\boldsymbol{\epsilon}}_k$ are work-conjugated with the force per unit length vectors $\boldsymbol{n}_k$.

The Biot-like angular deformation is defined as

$$\tilde{\boldsymbol{k}}_k = \boldsymbol{T}^\top \boldsymbol{k}_{/k} - \boldsymbol{T}_0^\top \boldsymbol{k}_{0/k} \tag{5.4}$$

where $\boldsymbol{k}$ is related to tensor $\boldsymbol{T}$ by $\boldsymbol{k}_{k\times} = \boldsymbol{T}_{/k}^\top \boldsymbol{T}$.

The angular strain vectors $\tilde{\boldsymbol{k}}_k$ are work-conjugated with the internal couple per unit length vectors $\boldsymbol{m}_k$.

The vector $\boldsymbol{\epsilon} = (\tilde{\boldsymbol{\epsilon}}_1, \tilde{\boldsymbol{\epsilon}}_2, \tilde{\boldsymbol{k}}_1, \tilde{\boldsymbol{k}}_2)^\top$ can be used to completely define the straining of the shell. Since $\boldsymbol{\epsilon}$ is work-conjugate to $\boldsymbol{\sigma} = (\boldsymbol{n}_1, \boldsymbol{n}_2, \boldsymbol{m}_1, \boldsymbol{m}_2)^\top$, the virtual internal work is thus expressed by

$$\delta L_i = \int_S \delta \boldsymbol{\epsilon}^\top \boldsymbol{\sigma} \, \mathrm{d}S \tag{5.5}$$

In §Quaranta et al. [23], the angular strain vectors $\tilde{\boldsymbol{k}}_k$ are computed directly from their definition (5.4), and not from the back-rotated gradient of the rotation tensor $\boldsymbol{\Phi} = \boldsymbol{T}\boldsymbol{T}_0^\top$, as in many other works. Furthermore, using eqs. (5.3) and (5.4) directly, together with their work-conjugated forces/couples per unit length, we don't need to resort to co-rotational derivatives in the definition of the strain vectors.

After defining in the reference configuration a triad of unit orthogonal vectors $\boldsymbol{t}_{n1}$, $\boldsymbol{t}_{n2}$, $\boldsymbol{t}_{n3}$ for each node n, with $\boldsymbol{t}_{n1}$ and $\boldsymbol{t}_{n2}$ tangent to the shell surface and $\boldsymbol{t}_{n3} = \boldsymbol{t}_{n1} \times \boldsymbol{t}_{n2}$, the interpolation of the orientation field can be performed. If $\boldsymbol{R}_n$ denotes the orientation tensor of the node, and if the local shell orientation is defined as

$$\boldsymbol{T}_n = \boldsymbol{R}_n \boldsymbol{R}_{0n}^\top [\boldsymbol{t}_{n1}, \boldsymbol{t}_{n2}, \boldsymbol{t}_{n3}]$$

where $\boldsymbol{R}_{0n}$ is the nodal orientation in the reference configuration, the average orientation $\bar{\boldsymbol{T}}$ of the shell can be computed as

$$\bar{\boldsymbol{T}} = \exp\left(\log\left(\frac{1}{4}\sum_{i=1,4} \boldsymbol{T}_n\right)\right)$$

where $\log(\cdot)$ extracts a skew symmetric tensor, i.d., $\boldsymbol{a}_\times$, and $\exp(\boldsymbol{a}_\times)$ computes the rotation tensor defined by the rotation vector $\boldsymbol{a}$.

For each node n, in order to interpolate the relative rotation vectors that define the relative nodal rotations $\tilde{\boldsymbol{R}}_n = \bar{\boldsymbol{T}}^\top \boldsymbol{T}_n$, i.d.,

$$\tilde{\boldsymbol{\varphi}}(\boldsymbol{\xi})_\times = \log(\tilde{\boldsymbol{R}}_n) = \sum N_n(\boldsymbol{\xi}) \log(\bar{\boldsymbol{T}}^\top \boldsymbol{T}_n)$$

standard bi-linear interpolation shape functions $N_n(\boldsymbol{\xi})$ are thus defined. Finally, the interpolated orientation states as $\boldsymbol{T}_i = \bar{\boldsymbol{T}} \exp(\tilde{\boldsymbol{\varphi}}_{i\times})$.

Since we have to compute virtual internal work (5.5) and its linearization, the computing of the first (δ) and second ($\partial\delta$) variations of the linear and angular deformations vectors (eqs. (5.3) and (5.4)) is needed. Consequently an explicit expressions for the interpolated virtual rotation vector $\boldsymbol{\varphi}_{i\delta}$, defined by $\boldsymbol{\varphi}_{i\delta\times} = \partial \boldsymbol{T}_i \boldsymbol{T}_i^\top$, is required.

The virtual rotation vector $\boldsymbol{\varphi}_\delta$ can be computed from the virtual variation of the rotation vector as $\boldsymbol{\varphi}_\delta = \boldsymbol{\Gamma}(\boldsymbol{\varphi})\delta\boldsymbol{\varphi}$, with $\boldsymbol{\Gamma}(\boldsymbol{\varphi})$ a second order tensor.

The interpolated virtual rotation vector thus is

$$\boldsymbol{\varphi}_{i\delta} = \sum_{n} \boldsymbol{\Phi}_{in} N_{in} \boldsymbol{\varphi}_{n\delta}$$

with

$$\boldsymbol{\Phi}_{in} = \bar{\boldsymbol{T}} \tilde{\boldsymbol{\Gamma}}_i \tilde{\boldsymbol{\Gamma}}_n^{-1} \bar{\boldsymbol{T}}^\top$$

Thus, the first and second variations of the linear strain vectors and of the back-rotated curvature can be calculated.

The constitutive law of the shell must be computed beforehand: only linear elastic constitutive properties can be currently modeled. They consist in a 12×12 matrix that expresses the force and moment fluxes as functions of linear and angular strains according to:

$$\begin{Bmatrix} \boldsymbol{n}_1 \\ \boldsymbol{n}_2 \\ \boldsymbol{m}_1 \\ \boldsymbol{m}_2 \end{Bmatrix} = \mathbb{D} \begin{Bmatrix} \tilde{\boldsymbol{\epsilon}}_1 \\ \tilde{\boldsymbol{\epsilon}}_2 \\ \tilde{\boldsymbol{k}}_1 \\ \tilde{\boldsymbol{k}}_2 \end{Bmatrix}$$

6 Interfacing the Structural and the Aerodynamic model

In this work, an original general-purpose, consistent meshless approach has been used to model Fluid-Structure Interaction (FSI). It is based on Moving Least Squares (MLS) using Radial Basis Functions (RBF) and represents a very efficient and reliable way to couple a multifield analysis with possibly incompatible interface boundaries, see §Quaranta [22].

6.1 Introduction

Given an aeroelastic system, there are mainly two ways to solve it: one can directly try to write down the system of PDE which govern the coupled FSI problem, and then try to discretize it as a whole (this approach is hardly ever used because each model has different mathematical and numerical properties), otherwise one can develop specialized methods to solve each model independently, or better can resort to existing and well-established numerical techniques for each discipline (this approach is widely followed in usual practice). Anyhow, to solve a coupled fluid structure problem it is not sufficient to be able to compute the solution of the structural and of the aerodynamic model, but it is also necessary to exchange information between the two models: the modification of boundary conditions must be transferred from the deformable structure to the aerodynamic boundary, and conversely, the loads developed in the aerodynamic field must be applied to the discretized structural model. This task is left to a correct *interface scheme*, which must interpolate/extrapolate data in an appropriate manner: by this interface scheme it's possible to interpolate the structural displacements and velocities at the aerodynamic nodes lying on body's boundary. To translate this information in a variation of the boundary condition of the aerodynamic system, the more simple and conceptually correct approach consists of deforming the aerodynamic grid and reformulate the aerodynamic problem according to ALE description.

The problem of building a methodology to couple fluid and structure has always been a key aspect of aeroelastic methods. In §Quaranta [22] a novel interfacing methodology is presented: it is an attempt to develop an accurate method capable of interfacing any possible structural or aerodynamic discretization scheme.

6.2 Fluid-Structure Interface

This section describes the coupled fluid-structure interaction problem. As a prototypical application, the configuration shown in Figure 6.1 has to keep in mind:

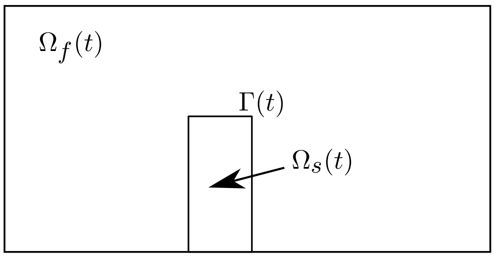


Figure 6.1: Fluid-structure interaction domain: Ω_f is the fluid domain, Ω_s is the structure domain, Γ is the common interface.

Any fluid dynamic model will compute the stresses applied on the wet surfaces Γ of a body, given the dynamic state of the boundaries. On the other hand there is the structural model: the loads developed in the aerodynamic field must be applied to it and the consequently modification of boundary conditions must be re-transferred to the aerodynamic boundary. Coupling conditions must be added to complete the model. If $\boldsymbol{y}_\mathrm{f}$ denotes the fluid boundary movement, so $\dot{\boldsymbol{y}}_\mathrm{f} = \boldsymbol{u}_f = (u, v, w)^\top$, $\boldsymbol{y}_\mathrm{S}$ the structural boundary movement, p its pressure, σ_S the structure stress tensor, $\sigma_\mathrm{f} = \nu_f(\nabla \boldsymbol{u}_f + \nabla^\top \boldsymbol{u}_f) - p\mathbb{I}$ the fluid stress tensor, and $\boldsymbol{n}$ the normal vector to Γ, we can express these conditions (for a continuous system) as

$$\left.\begin{aligned} \sigma_\mathrm{S} \cdot \boldsymbol{n} &= \sigma_\mathrm{f} \cdot \boldsymbol{n}, \\ \boldsymbol{y}_\mathrm{S} &= \boldsymbol{y}_\mathrm{n}, \\ \dot{\boldsymbol{y}}_\mathrm{S} &= \dot{\boldsymbol{y}}_\mathrm{n} \end{aligned}\right\} \quad \text{on } \Gamma. \tag{6.1}$$

The interaction of the fluid with the solid is given by the continuity of the velocity and by a balance of forces: the first of (6.1) expresses the dynamic equilibrium between the stresses on the structure with those on the fluid side. The other two are kinematic compatibility conditions. If a moving grid is considered, the conditions on the kinematics of the boundary must be reflected also on the moving grid, so $\boldsymbol{x} = \boldsymbol{y}_\mathrm{S}$, $\dot{\boldsymbol{x}} = \dot{\boldsymbol{y}}_\mathrm{S}$ on Γ, where here $\dot{\boldsymbol{x}}$ is the velocity of the moving boundary of the mesh elements (where possible the notation has intentionally not been changed to that from §Quaranta [22], in order to allow an easier and quicker comparison).

In order to derive the correct expression of (6.1) for the discretized problem, since the fluid and structure fields are solved resorting to a discrete approximation, the constraint of retaining the conservation properties of the original problem is fixed: the conservation of mass is satisfied directly by the continuity equation for the fluid field (mass is not added nor removed), the change of energy in the fluid-structure system equals the energy supplied (or absorbed) by an external force (energy is not added nor removed), and finally, the change of momentum of the fluid-structure system is equivalent to the acting (external) forces.

When the fluid and structure domains have matching discrete interfaces and compatible approximation spaces, the conservation is trivially retained. In realistic applications, however, the fluid and structural meshes are not compatible along the interface, either because discretized with major geometric discrepancies or because each problem has different resolution requirements: typically, the fluid grid, at the interface, is finer than the structural mesh. In the following, Γ_f and Γ_s will respectively indicate the discrete fluid/structure non-matching representation of Γ.

In order to respect the conservation between the two models, the coupling conditions was enforce only in a weak sense, through the use of simple variation principles such as that of Virtual Works. Let $\delta\boldsymbol{y}_\mathrm{f}$ and $\delta\boldsymbol{y}_\mathrm{s}$ be two admissible virtual displacements for each field. Admissible means that, for the viscous case, the trace of these two fields on Γ must be equal

$$\mathrm{Tr}(\delta\boldsymbol{y}_\mathrm{f})\big|_\Gamma = \mathrm{Tr}(\delta\boldsymbol{y}_\mathrm{s})\big|_\Gamma \tag{6.2}$$

The relationship between the admissible virtual displacements can be written, independently of the interpolation method chosen to enforce the compatibility of the displacements, as:

$$(\delta\boldsymbol{y}_\mathrm{f})_i = \sum_{j=1}^{j_\mathrm{s}} h_{ij}(\delta\boldsymbol{y}_\mathrm{s})_j$$

where $(\delta\boldsymbol{y}_\mathrm{f})_i$ are the discrete values of $\delta\boldsymbol{y}_\mathrm{f}$ at the fluid nodes of the grid on the wet surface, and h_{ij} are the coefficients of the displacements interpolation matrix H. So, the resulting virtual displacement is

$$\delta\boldsymbol{y}_\mathrm{f} = \sum_{i=1}^{i_\mathrm{f}} N_i(\delta\boldsymbol{y}_\mathrm{f})_i = \sum_{i=1}^{i_\mathrm{f}} N_i \sum_{j=1}^{j_\mathrm{s}} h_{ij}(\delta\boldsymbol{y}_\mathrm{s})_j$$

where i_f is the number of nodes which belong to the interface surface Γ and the functions N_i belong to the approximation space of the aerodynamic field discretization. The virtual work of the aerodynamic load results in

$$\begin{aligned}\delta W_\mathrm{f} &= \int_{\Gamma_\mathrm{f}} (\sigma_\mathrm{f} \cdot \boldsymbol{n}) \cdot \delta \boldsymbol{y}_\mathrm{f} \, \mathrm{d}S = \\ &= \int_{\Gamma_\mathrm{f}} (\sigma_\mathrm{f} \cdot \boldsymbol{n}) \cdot \sum_{i=1}^{i_\mathrm{f}} N_i \sum_{j=1}^{j_\mathrm{s}} h_{ij} (\delta \boldsymbol{y}_\mathrm{s})_j \, \mathrm{d}S\end{aligned}$$

On the other side, the virtual work of forces and moments acting on Γ_s is equal to

$$\delta W_\mathrm{s} = \sum_{j=1}^{j_\mathrm{s}} \boldsymbol{f}_j \cdot (\delta \boldsymbol{y}_\mathrm{s})_j$$

Imposing the equality of the virtual works, the nodal loads applied to the structure are

$$\boldsymbol{f}_j = \sum_{i=1}^{i_\mathrm{f}} \boldsymbol{F}_i h_{ij} \tag{6.3}$$

where

$$\boldsymbol{F}_i = \int_{\Gamma_\mathrm{f}} (\sigma_\mathrm{f} \cdot \boldsymbol{n}) N_i \, \mathrm{d}S$$

From equation (6.3) we can observe that, after computing the nodal loads $\boldsymbol{F}_i$ for the aerodynamic boundary grid points, the loads $\boldsymbol{f}_j$ on the structural nodes can be obtained by simply multiplying $\boldsymbol{F}_i$ for the transpose of the interpolation matrix H computed to connect the two grid displacements. The problem of conservativeness[1] is so shifted on the definition of the correct interpolation matrix H.

To build a conservative interpolation matrix which enforces the compatibility, Eq. (6.2), a weak/variational formulation was used in §Quaranta [22]. The problem is expressed in a weighted least-square form:

$$\text{Minimize} \quad \int_\Gamma \Phi \left(\mathrm{Tr}(\delta \boldsymbol{y}_\mathrm{f})\big|_\Gamma - \mathrm{Tr}(\delta \boldsymbol{y}_\mathrm{s})\big|_\Gamma \right)^2 \mathrm{d}S \tag{6.4}$$

where $\Phi(\boldsymbol{x})$ is a weight function. The adopted solution, which also is required to possess additional qualities such as smoothness of the resulting interpolated field, computational efficiency and some control on the interpolation error, was to resort to the Moving Least Square (MLS) technique. This technique allows to build a sufficiently regular and accurate

[1]Given a unitary vector $\mathbf{1}$, H is said to be conservative if $H \cdot \mathbf{1} = \mathbf{1}$. Since $\boldsymbol{f} = H^\top \boldsymbol{F}$, resultants are thus conserved: $\mathbf{1}^\top \cdot \boldsymbol{f} = \mathbf{1}^\top \cdot H^\top \boldsymbol{F} = \mathbf{1}^\top \boldsymbol{F}$.

approximation of the field of structural displacements and velocities at the aerodynamic nodes belonging to the boundary of the body, knowing the solution only within a set of structural nodes, in general irregularly distributed. The field of structural displacements and velocities is thus approximated at the aerodynamic nodes lying on the body's boundary as a sum of monomial basis functions $p_i(\boldsymbol{x})$

$$\hat{\boldsymbol{u}}(\boldsymbol{x}) = \sum_{i=1}^{m} p_i(\boldsymbol{x}) a_i(\boldsymbol{x}) = \boldsymbol{p}^\top \boldsymbol{a}$$

where m in the number of basis functions, and a_i are their coefficients. Commonly adopted basis functions are linear or quadratic polynomials to avoid numeric instability problems:

$$\boldsymbol{p}^\top(\boldsymbol{x}) = (1, x, y, z)^\top \qquad \in C^1(\mathbb{R}^3)$$
$$\boldsymbol{p}^\top(\boldsymbol{x}) = (1, x, y, z, x^2, xy, y^2, yz, z^2, zx)^\top \qquad \in C^2(\mathbb{R}^3)$$

The coefficients $a_i(\boldsymbol{x})$ can be obtained performing a weighted least square fit for an appropriate norm of the local error given by adopting the interpolating function $\hat{\boldsymbol{u}}(\boldsymbol{x})$, i.d., minimizing the following functional, written in a variational form:

$$J(\boldsymbol{x}) = \int_\Omega \Phi(\boldsymbol{x}) \|\boldsymbol{\varepsilon}(\boldsymbol{x})\|^2 \, \mathrm{d}\Omega \tag{6.5}$$

where $\Phi(\boldsymbol{x})$ is an appropriate weight function and $\boldsymbol{\varepsilon}(\boldsymbol{x})$ the local error. The equation (6.5) is equivalent to (6.4), which expresses the interface problem (all the details may be found in §Quaranta [22]).

If we consider, as example, a set of N_j structural nodes $\boldsymbol{x}_{s,j}$, in general irregularly distributed and in correspondence of which are known the structural displacements $\boldsymbol{u}_{s,j}$, then the unknown coefficient $a_i(\boldsymbol{x})$ can be calculated minimizing the following functional written in a discrete form:

$$J(\boldsymbol{x}) = \sum_{j=1}^{N_j} \Phi(\boldsymbol{x} - \boldsymbol{x}_{s,j}) \left\| \sum_{i=1}^{m} p_i(\boldsymbol{x}_{s,j}) a_i(\boldsymbol{x}) - \boldsymbol{u}_{s,j} \right\|^2 .$$

The regularity of the field of the interpolated displacements and velocities at the aerodynamic nodes belonging to body's boundary depends significantly on the choice of the weight function. The problem is thus localized by choosing weight functions that are smooth non-negative Radial Basis Functions (RBFs) with compact support. The adoption of different RBF functions, together with the definition of the local support dimension, allows to achieve the required regularity of the interpolated function. The resulting interface matrix satisfies all the requirements:

1. is effective, because the computational complexity can be accomplished in an efficient manner;

2. can produce surfaces with any prescribed smoothness;

3. allows a high control on the behavior for each local region.

The interface method is completely independent from the implementation details of the codes that are interfaced: as a consequence, the implementation of the structural elements or the code used to solve the aerodynamic field can be changed without affecting the interface code.

Usually the weight RBF are written as $\Phi(r/\delta)$, where $r = \|\boldsymbol{x} - \boldsymbol{x}_{s,j}\|$ and δ a scaling factor that allows the user to adapt the support dimension N_j of the structural nodes $\boldsymbol{x}_{s,j}$. In this way, one can be sure that enough points are covered and far away points have no influence.

6.3 Application to FSI problem

Since we are dealing with structures that can experience large changes in orientation, a problem that requires special attention is related to the treatment of rotations, which are complex tensorial entities that can be subjected to singularity problems when parametrized. In §Quaranta [22] a simple procedure has been developed to avoid the need to build a specialized class of interpolation matrices for these entities, since the updated position of an aerodynamic surface node is obtained by combining the effect of linear displacements and rotations: the idea is to add three rigid arms to each structural beam node, directed along the node local reference axes, creating a sort of *fish-bone* structure. The effect of the rotation is accounted for by the displacements of the added dummy nodes at the end of each arm, and matrix H is computed using only spatial displacements (of course, the length of the rigid arms needs to be smaller than the distance between two structural beam nodes).

Using the transpose of matrix H, the loads computed at the aerodynamics nodes can be transferred to the structural mesh. All the forces and moments applied to the fictitious nodes must be transferred to the structural node they are attached, exploiting the property that the fishbone arms are rigidly connected to the structural node.

7 The FEniCS project

A significant part of the work on this thesis has been carried out using the software provided by the open source project FEniCS. The FEniCS Project (`http://fenicsproject.org`) set out in 2003 with an idea to automate the solution of mathematical models based on differential equations. At first, the FEniCS Project consisted of two libraries: DOLFIN and FIAT. Since then, the project has grown and now consists of the core components DOLFIN, FFC, FIAT, Instant, UFC and UFL. This project is an international collaboration between many research institutions: University of Chicago, Argonne National Laboratory, Delft University of Technology, Royal Institute of Technology KTH, Simula Research Laboratory, Texas Tech University, and University of Cambridge.

7.1 FEniCS

FEniCS is a platform for the creation and maintenance of tools used for the solution of differential equations using the finite element method[1]. FEniCS includes tools for working with computational meshes, linear algebra and finite element variational formulations of PDEs. In addition, FEniCS provides a collection of ready-made solvers for a variety of partial differential equations.

At the *heart* of FEniCS is DOLFIN which is a set of libraries that enable the user to create code in a consistent problem solving environment. For more details see §Alnaes et al. [4]; Logg et al. [20].

7.1.1 DOLFIN

For a detailed discussion on the design and implementation of DOLFIN (Dynamic Object-oriented Library for FINite element computation) we

[1]FEniCS supports finite element schemes, including discontinuous Galerkin methods, but not finite difference methods. Many finite volume methods can be constructed as low-order discontinuous Galerkin methods using FEniCS.

refer to §Logg and Wells [18, 19].

DOLFIN is a C++/Python library that functions as the main user interface of FEniCS. It provides a problem solving environment for models based on ordinary or partial differential equations and implements core parts of the functionally of FEniCS, including data structures and algorithms for computational meshes and finite element assembly. In order to provide a simple and consistent user interface, DOLFIN wraps the functionality of other FEniCS components and external software, and handles the communication between these components. Figure 7.1 presents an overview of the relationships between the components of FEniCS and external software.

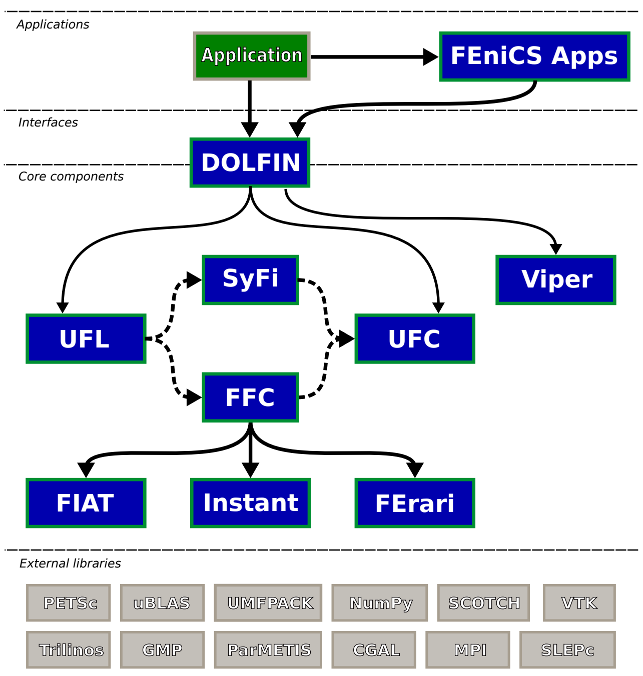


Figure 7.1: DOLFIN functions as the main user interface of FEniCS and handles the communication between the various components of FEniCS and external software. Solid lines indicate dependencies and dashed lines indicate data flow. Taken from §Logg et al. [20]

DOLFIN itself functions as both a user interface and a core component of FEniCS. All communication between a user program, other core components of FEniCS and external software is routed through wrapper layers that are implemented as part of DOLFIN user interface. It also provides important classes for mesh generation, matrix and vector assembly, and various finite element functions and operations. In particular, as shown above, variational forms expressed in the UFL form language are passed to the form compiler FFC to generate UFC code, which can then be used by DOLFIN to evaluate (assemble) variational forms. This code generation depends on the finite element back-end FIAT (FInite element Automatic Tabulator), the just-in-time compilation utility Instant and the optional optimizing back-end FErari. Finally, the plotting capabilities provided by DOLFIN are implemented by Viper. DOLFIN also relies on external software for important functionality such as the linear algebra back-end PETSc, Trilinos and the mesh partitioning libraries. As mentioned above, DOLFIN provides two user interface. One interface is implemented as a traditional C++ library, and another interface is implemented as a standard Python module. The two interfaces are near-identical, but in some cases particular language features of either C++ or Python require variations in the interface. In particular, the Python interface adds an additional level of automation and offers some functionality that is not available from the C++ interface: the UFL form language is seamlessly integrated into the Python interface and code generation is automatic handled at run-time.

7.1.2 FFC

The automated code generation, which is a key features of FEniCS, relies on a form compiler for offline or just-in-time compilation of code for individual forms. Two different form compilers are available as part of FEniCS: here the form compiler FFC is described. The other form compiler, SyFi/SFC, is not described.

The solution of finite element variational problems is based on the assembly of linear or nonlinear system of equations and the solution of these equations. As a result, many finite element codes are similar in their design and implementation: a central part of most finite element codes is the assembly of sparse matrices from finite element bi-linear forms.

FFC takes as input a variational form specified in the UFL form language (see 7.1.4) and generates as output C++ code that conforms to the UFC interface (see 7.1.3). FFC provides three different interfaces a Python interface, a command-line interface, and a “just-in-time” (JIT) compilation interface. While Python users mostly rely on DOLFIN to handle the communication with FFC, the command-line interface is familiar for DOLFIN C++ users, who must call FFC on the command-line to generate code for inclusion in their C++ programs. The JIT interface is rarely called directly by users, but it is the main interface between DOLFIN and FFC, which allows DOLFIN to seamlessly generate and compile code when running solver scripts implemented in Python. When the JIT compiler is called, FFC generates internally UFC code for the given form or finite element, compiles the generated code using a C++ compiler, and then wraps the result as a Python module using SWIG and Instant. The returned objects are ready to be used from Python. The generated and wrapped code is cached by the JIT compiler, so if the JIT compiler is called twice for the same form or finite element, the cached version is used (the cache directory is created by Instant). The Python interface of DOLFIN makes extensive use of JIT compilation.

7.1.3 UFC

The "Unified Form-assembly Code" (UFC, §Alnæs et al. [1]) is an interface between problem-specific and general-purpose components of finite element programs. In particular, the UFC interface defines the structure and signature of the code that is generated by the form compilers FFC for DOLFIN. The UFC interface applies to a wide range of finite element problems (including mixed finite elements and discontinuous Galerkin methods) and may be used with libraries that differ widely in their design. For this purpose, the interface does not depend on any other FEniCS components (or other libraries) and consists only of a minimal set of abstract C++ classes using plain C arrays for data transfer.

The implementation of solvers for the solution of partial differential equations by the finite element method is much helped by the existence of generic software libraries that provide data structures and algorithms for computational meshes and linear algebra. This allows the implementation of a generic assembly algorithm that may be partly reused from one application to another. However, since the inner loop of the assembly algorithm inherently depends on the partial differential equation being solved and the finite elements used to produce the discretization, the writing of the inner loop (which is typically supplied by the user) is a challenging task that is prone to errors, and which prohibits experimentation with models and discretization methods.

The FEniCS tool-chain of FIAT-UFC-FFC/SFC-UFC-DOLFIN is an attempt to solve this problem: it is able to provide a completely generic implementation of the assembly algorithm as part of DOLFIN by generating automatically the inner loop based on a high-level description of the finite element variational problem (in the UFL form language).

7.1.4 UFL

DOLFIN relies on the "Unified Form Language" (UFL, §Alnæs and Logg [2, 3]) for the expression of variational forms. The Unified Form Language UFL is a domain specific language for the declaration of finite element discretization of variational forms and functionals. More precisely, the language defines a flexible user interface for defining finite element spaces and expressions for weak forms in a notation close to mathematical notation. Forms can involve integrals over cells, interior facets, and exterior facets.

7.2 Implementation details

7.2.1 Preliminaries

We started from the definition of the stress tensor $\sigma(\boldsymbol{u}, p) = 2\nu\varepsilon(\boldsymbol{u}) - p\mathbb{I}$, where the $\varepsilon(\boldsymbol{u}) = \frac{1}{2}(\nabla\boldsymbol{u} + \nabla\boldsymbol{u}^\top)$ is the symmetric velocity gradient.

Python code

```
def epsilon(u):
  "Return symmetric gradient."
    return 0.5*(grad(u) +  grad(u).T)

def sigma(u, p, nu):
  "Return stress tensor."
   return 2*nu*epsilon(u) - p*Identity(u.cell().d)
```

7.2.2 Time discretization

When performing unsteady simulations it is important that flow information is transferred correctly between time steps. The time step size is limited to CFL number below one, i.e

$$\text{CFL} = \frac{U\Delta t}{\Delta x} \leq 1$$

Physically, this criteria states that a fluid particle, traveling at speed U should not travel through more than one grid cell during on time step. This applies in general but in order to compromise between accuracy and simulation time it is acceptable that the CFL number exceeds unity in few small areas in the flow field. This is possible since the solver used in this study solves the flow equations implicitly whereas in an explicit solver this criteria must be met.

7.2.3 Mesh

The mesh, that has been generated by an external mesh generator, is read from file: a usefull class for storing data associated with a mesh, the `MeshFunction` class, is used. This makes it simple to store, for example, material parameters, subdomain indicators, refinement markers on the “Cells” of a mesh or boundary markers on the “Facets” of a mesh. Then, the `MeshData` class provides a simple way to associate data with a mesh. It allows arbitrary `MeshFunctions` (and other quantities) to be associated with a mesh. The following code illustrates how to attach a `MeshFunction` named “bfc” to a mesh:

Python code

```python
# Read mesh from file
mesh = Mesh("mesh.xml")
bfc =  mesh.data().create_mesh_function("bfc")
# Read meshfunction from file
file_in = File("mesh_function.xml")
file_in >> bfc
```

DOLFIN predefines the “measures” `dx`, `ds` and `dS` representing integration over cells, exterior facets (that is, facets on the boundary) and interior facets, respectively. These measures can take an additional integer argument. In fact, `dx` defaults to `dx(0)`, `ds` defaults to `ds(0)`, and `dS` defaults to `dS(0)`. Integration over sub-regions can be specified by measures with different integer labels as arguments. However, we also need to map the geometry information stored in the mesh functions to these measures. The easiest way of accomplishing this is to define new measures with the mesh functions as additional input:

Python code

```python
  dss = Measure("ds")[bfc]
```

We can now define the variational forms corresponding to the variational problem above using these measures and the tags for the different sub-regions or sub-boundaries.

7.2.4 Solver

We next defined a pair of function spaces for the velocity and pressure, and trial and test functions on these spaces. For the G2 scheme, continuous piecewise linear polynomials are used for both the velocity and the pressure.

Python code

```python
V = VectorFunctionSpace(mesh, "CG", 1)
Q = FunctionSpace(mesh, "CG", 1)
W = V * Q
# Test functions
v, q = TestFunctions(W)
# Trial  function
```

```
dw = TrialFunction(W)
# Current solution
w = Function(W)
u, p = split(w)
# Solution from previous converged step
w0 = Function(W)
u0, p0 = split(w0)
```

In order to compute the stabilization parameters δ_1 and δ_2, we defined an `Expression` using a more complicated logic with the "cppcode" argument. This argument should be a string of C++ code that implements a class that inherits from DOLFIN Expression class.

Python code

```
DGe = FiniteElement("DG", mesh.ufl_cell(), 0)
delta1 = Expression(cppcode_d1, element=DGe)
delta2 = Expression(cppcode_d2, element=DGe)
```

The version of G2 studied in this thesis, explained in Chapter 3, is based on the version presented in §Hoffman and Johnson [16], and its implementation in DOLFIN is shown in Figure 7.2. For brevity, the code for specifying Dirichlet conditions is omitted (the boundary conditions are assumed to be available as lists, `bcs`, in the program).

Lift and Drag

When solving the Navier-Stokes equations, it may be of interest to compute the lift and drag of the object immersed in the fluid. The lift and drag are given by the z- and x-components of the force generated on the object (for a flow in the x-direction), as shown in Figure 7.3:

$$L(\boldsymbol{u}, p) = \int_\Gamma (\sigma(\boldsymbol{u}, p) \cdot \boldsymbol{n}) \cdot \boldsymbol{e}_z \,\mathrm{d}S,$$

$$D(\boldsymbol{u}, p) = \int_\Gamma (\sigma(\boldsymbol{u}, p) \cdot \boldsymbol{n}) \cdot \boldsymbol{e}_x \,\mathrm{d}S$$

Here, Γ is the boundary of the body and $\boldsymbol{e}_x$, $\boldsymbol{e}_z$ are unit vectors in the x- and z-directions respectively. Thus, the traction is $\boldsymbol{T} = \sigma \cdot \boldsymbol{n}$, where $\boldsymbol{n}$ is the inward-pointing unit normal. In the code, `n` is the outward-pointing unit normal. Then, Figure 7.4 shows how to compute lift and drag in FEniCS from a computed velocity field $\boldsymbol{u}$ and pressure field p. In the code, `[0]` and `[2]` are the x- and z-directions, respectively, and `dss(7)` is the boundary marker representing Γ.

In order to compute the forces acting on the nodes of the surface Γ, we can "test" the product of the stress tensor σ and the unit normal

Python code

```
knu = Constant(nu)
k = Constant(dt)
n = FacetNormal(mesh)
kappa1 = 1.0
kappa2 = 2.0

# Update stabilization parameters
uu0, pp0 = w0.split(True)
delta1.update(uu0, nu, dt, kappa1)
delta2.update(uu0, nu, dt, kappa2)
# Solve nonlinear system
U = 0.5*(u + u0)
F = inner((u - u0) / k, v) * dx +
    inner(dot(grad(U), U), v)*dx +
    2.*knu*inner(epsilon(U), epsilon(v))*dx -
    p*div(v)*dx+div(U)*q*dx +
    delta1*inner((dot(grad(U), U) + grad(p)),
        dot(grad(v), U) + grad(q))*dx +
    delta2*div(U)*div(v)*dx +
    1./alfa*dot(u,n)*dot(v,n)*dss(7) +
    beta*inner(outer(u,v),Identity(u.cell().d) -
        outer(n,n))*dss(7)

# Compute directional derivative about w in the
   direction of dw  (Jacobian)
J = derivative(F, w, dw)
# Define variational problem
pde = NonlinearVariationalProblem(F, w, bcs, J)
solver = NonlinearVariationalSolver(pde)
solver.solve()
```

Figure 7.2: Implementation of variational forms for the G2 solver.

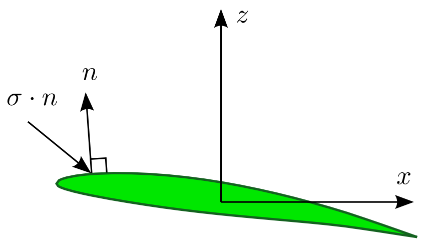


Figure 7.3: The lift and drag of an airfoil, are the integrals of the vertical and horizontal components, respectively, of the stress $\sigma \cdot n$ over the surface Γ of the airfoil. At each point, the product of the stress tensor σ and the outward unit normal vector $\boldsymbol{n}$ gives the force per unit area acting on the surface.

Python code

```
D = -dot(sigma(u,p,nu),n)[0]*dss(7)
L = -dot(sigma(u,p,nu),n)[2]*dss(7)
drag = assemble(D)
lift = assemble(L)
Cl = lift / (0.5*chord*span*flow_velocity**2)
Cd = drag / (0.5*chord*span*flow_velocity**2)
```

Figure 7.4: Computation of lift and drag in FEniCS.

vector $\boldsymbol{n}$ against piecewise linear vector test functions `k`: we thus obtain, for each mesh node, the force acting on it (since the integrals are defined over the surface Γ, the only non-zero forces are those of the nodes lying on Γ).

Python code

```
V = VectorFunctionSpace(mesh,'CG',1)
k = TestFunction(V)
s_k = -inner(dot(sigma(u,p,nu), n), k)*dss(7)
forces = assemble(s_k)
```

If the adaptive version of the G2 method is use, to compute the forces acting on the nodes on Γ after that the mesh has been adaptively refined, one can "test" the product $\sigma \cdot \boldsymbol{n}$ against piecewise linear vector test functions `k0`, which are now defined on the initial mesh (`mesh0`): in this way we obtain the forces acting on the vertices lying on the interface surfaces of the initial mesh (`bfc_mesh0`). This because the interpolation matrix H is computed only once, using the "aerodynamic" nodes given by the initial mesh, and, if the adaptive process adds some nodes on the interface surface, we don't have to re-compute matrix H.

Python code

```
V0 = VectorFunctionSpace(mesh0,'CG',1)
k0 = TestFunction(V0)
bfc_mesh0 = mesh0.data().mesh_function('bfc')
ds_0 = ds[bfc_mesh0]
s_n = -inner(dot(sigma(u,p,nu), n), k0)*ds_0(7)
forces = assemble(s_n, mesh = mesh0)
```

7.2.5 ALE

If we want to be able to calculate a reliable solution whilst moving the mesh, we have to implement the ALE description. As we have also learned in Chapter 4, this can quite easily be done by just taking into account the mesh movement velocity $\boldsymbol{v}_\mathrm{m}$ alongside the existing velocity $\boldsymbol{u}$. To illustrate this small change, here is the implementation in DOLFIN of the ALE friendly incompressible Navier-Stokes equations.

Python code

```python
U = 0.5*(u + u0)
F = inner((u - u0)/k, v)*dx +
  inner(dot(grad(U), (U-V_m)), v)*dx +
  2.*knu*inner(epsilon(U), epsilon(v))*dx -
  p*div(v)*dx + div(U)*q*dx +
  delta1*inner((dot(grad(U), (U-V_m))+ grad(p)),
     dot(grad(v),(U-V_m))+grad(q))*dx +
  delta2*div(U)*div(v)*dx+
  1./alfa *dot(u-v_m, n)*dot(v,n)*dss(7) +
  beta*inner(outer(u-v_m,v),
     Identity(u.cell().d)-outer(n,n))*dss(7)
```

with $\boldsymbol{V}_\mathrm{m} = \frac{1}{2}(\boldsymbol{v}_\mathrm{m} + \boldsymbol{v}_\mathrm{m}^0)$, where $\boldsymbol{v}_\mathrm{m}^0$ is the mesh movement velocity at previous converged step.

7.2.6 Adaptive G2

For our purpose we adopted the DOLFIN ("Goal-oriented") `ErrorControl` class. Once we created suitable `ErrorControl` object from our problem and defined the goal of interest, we can estimate the error relative to the goal `M` of the discrete approximation `w` relative to the variational formulation. The dual approximation is defined by dual variational problem and dual boundary conditions given by homogenized primal boundary conditions. For more details, see §Rognes and Logg [24]. A number of optional parameters may be specified to control the behavior of the adaptive algorithm, including the choice of error estimate, the marking strategy, and the refinement fraction. The marking strategy adopted is the so-called Dörfler marking with a refinement fraction of $\alpha = 0.5$, in which a large enough subset $\mathcal{T}'$ of all cells (sorted by decreasing indicators) are marked for refinement such that the sum of the corresponding indicators constitute a given fraction α of the total error estimate. In Figure 7.5 is shown the implementation of such a scheme.

Python code

```
# Create suitable ErrorControl object from problem
    and the goal
pde = NonlinearVariationalProblem(F, w, bcs, J)
EC =  generate_error_control(pde, M)
solver = NonlinearVariationalSolver(pde)
solver.solve()
# Estimate the error relative to the goal M
estimated_error = abs(EC.estimate_error(w, bcs))
if (estimated_error <= error_tolerance):
   break # do not need to refine
# Compute error indicators
gamma=Vector(w.function_space().mesh().num_cells())
EC.compute_indicators(gamma, w)
# Refine mesh and update meshfunction
cell_markers=CellFunction("bool",mesh,
   mesh.topology().dim())
cell_markers.set_all(False)
cpp.mark(cell_markers, gamma, 'dorfler', 0.5)
bfc = mesh.data().mesh_function('bfc')
adapt(mesh, cell_markers)
adapt(bfc, mesh.child())
newmesh = mesh.child()
newmesh.set_parent(Mesh())
mesh.set_child(Mesh())
mesh.set_parent(Mesh())
mesh = newmesh
bfcnew = mesh.data().create_mesh_function('bfc',
   bfc.child().dim())
bfcnew.array()[:] = bfc.child().array()
bfc = bfcnew
```

Figure 7.5: Implementation of the adaptive version of G2 solver.

8 Mesh generation

DOLFIN provides functionality for creating simple meshes, such as meshes of unit squares and unit cubes, spheres, rectangles and boxes. Although the built-in classes are useful for testing, a typical application will need to read from file a mesh that has been generated by an external mesh generator. Meshes must be stored in the DOLFIN XML format. In this work, a library named "TriTetMesh" is used to produce high quality meshes for DOLFIN. TriTetMesh is a package that includes TriMesh and TetMesh, which are wrappers of the Triangle and Tetgen libraries. These comes as standalone C++ libraries and python modules. TriTetMesh is freely available on Launchpad at `https://launchpad.net/tritetmesh`.

8.1 TriTetMesh

TriTetMesh provides an intuitive interface to Triangle and Tetgen. These packages produces quality mesh in 2D and 3D based on triangles and tetrahedrons. The interface is provided as a C++/Python (through swig) wrapper. The output format is a DOLFIN `mesh` and a DOLFIN `MeshFunctions` for markers (see Chapter 7.2.3). It is also possible to output the native Tetgen and Triangle formats.

8.1.1 Triangle

Triangle is a C program for two-dimensional mesh generation. Triangle is fast, memory-efficient, and robust; features include user-specified constraints on angles and triangle areas, user-specified holes and concavities, and the economical use of exact arithmetic to improve robustness. Triangle is freely available on the Web at `http://www.cs.cmu.edu/~quake/triangle.html` and from Netlib. More details are presented in §Shewchuk [25].
Triangle reads a Planar Straight Line Graph[1]'s file, which can spec-

[1]A Planar Straight Line Graph (PSLG) is a collection of vertices and segments. Segments are simply edges whose endpoints are vertices in the PSLG, and whose

ify vertices, segments, holes, regional attributes, and regional area constraints, and then generates a mesh. One can also define "boundary markers" that are tags used mainly to identify which output vertices and edges are associated with which PSLG segment, and to identify which vertices and edges occur on a boundary of the triangulation. A common use is to determine where boundary conditions should be applied to a finite element mesh.

Triangle comes with a separate program named "Show Me", whose primary purpose is to draw meshes on the screen or in PostScript. Its secondary purpose is to check the validity of input files, and do so more thoroughly than Triangle does.

8.1.2 TetGen

TetGen creates tetrahedral meshes suitable for solving partial differential equations (PDEs) by finite element methods (FEM) and finite volume methods (FVM). The problem is to generate a tetrahedral mesh conforming to a given (polyhedral or piecewise linear) domain together with certain constraints for the size and shape of the mesh elements. It is a typical problem of provably good mesh generation or quality mesh generation. The techniques of quality mesh generation provide the "shape" and "size" guarantees on the meshes: all elements have a certain quality measure bounded, and the number of elements is within a constant factor of the minimum number.

TetGen uses general input called Piecewise Linear Complex (PLC). A PLC is a set of vertices, segments and facets. Each facet is a polygonal region, it may have any number of sides and may be non-convex, possibly with holes, segments and vertices in it. A facet can represent any planar straight line graph (PSLG), which is a popular input model used by many two-dimensional mesh algorithms. A facet is actually a PSLG embedded in three dimensions.

One can assign a boundary marker (an integer) for each facet of a PLC. In the final mesh, all boundary faces on that facet will have the same boundary marker (as shown in Figure 8.1). A common use of the boundary marker is to determine where boundary conditions should be applied to a finite element or finite volume mesh. Also one can assign different regions (separated by the internal facets) of the PLC with different region attributes (or number). Physically, they associate different

presence in any mesh generated from the PSLG is enforced. Segments may intersect each other only at their endpoints.

materials to these regions. In the final mesh, all tetrahedra in the same region will have the same region number.

TetView is a small graphic program for viewing tetrahedral meshes and piecewise linear complexes. It is created specifically for viewing and analyzing the input and output files of TetGen.

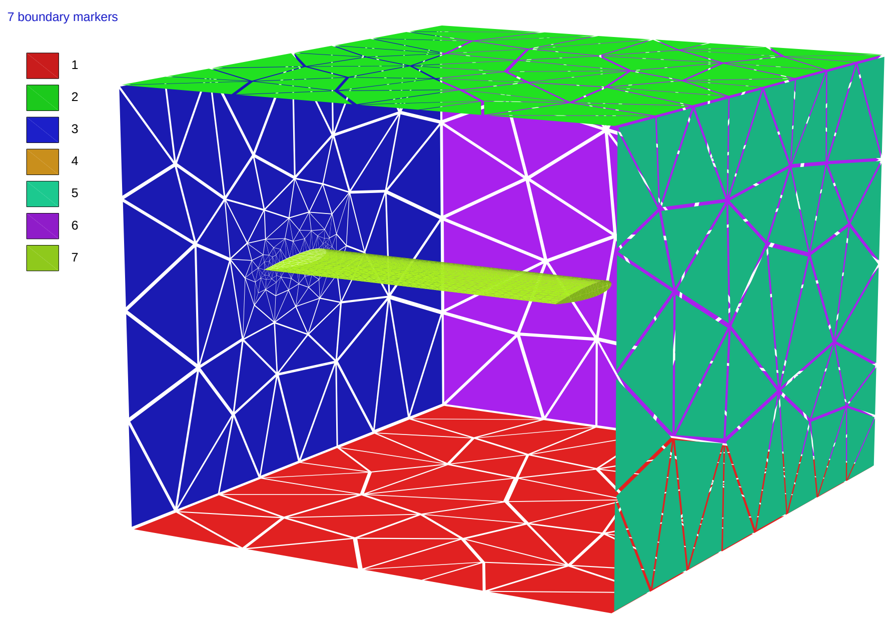


Figure 8.1: Example of usage of boundary markers, displayed by TetView.

Part II

Numerical Experiments

9 2D Unsteady Incompressible Viscous Flows

In aerodynamics, the investigation of the aerodynamic performance of a wing is usually started from the corresponding airfoil, which is the key factor of a wing's aerodynamic characteristics. Using the code implemented in this thesis, the unsteady flow past an unit chord NACA 0012 airfoil with an angle of incidence of $34°$ and at $Re = 1000$ has been calculated. The computational domain is the rectangle $[-2,5] \times [-3,3]$, and the airfoil center is placed at the origin. The results are then compared to those presented in §Guermond and Quartapelle [7]. Figure 9.1 is the unstructured triangular grids for the viscous flow around the considered airfoil: 3800 nodes as well as 7500 triangular elements are contained, and the mesh near the airfoil is refined. The mesh was generated when the airfoil is a $0°$ of incidence, then the airfoil was rotated by an angle $\alpha = 34°$ around its center and the fluid mesh follows it according to that we exposed in Chapter 4.4.1 (see Figure 4.3).

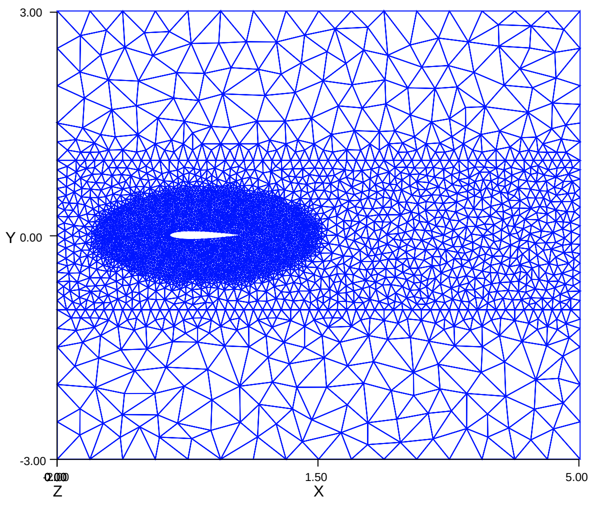


Figure 9.1: Mesh adopted for computation: it consists of about 3800 nodes and 7500 cells. The domain is the rectangle $[-2,5] \times [-3,3]$, and the airfoil center is placed at the origin. The airfoil is rotated by an angle $\alpha = 34°$ around the origin after the mesh has been generated.

At first, as in the above cited work, no attempt has been made to refine the mesh according to the computed solution and we have used a fixed dimensionless time step $\delta t = 0.02$. The boundary conditions for this external problem are:

- slip with linear friction and penetration with resistance b.c. on the airfoil,
- for $x = -2$: $\boldsymbol{u} = U\boldsymbol{i}$,
- for $y = \pm 3$: $\boldsymbol{u}_y = 0$,
- for $x = 5$: $p = 0$

The results for the unsteady flow past a NACA 0012 airfoil at an angle of incidence of $34°$ for $Re = 1000$ are reported in Figure 9.2. In this figure, the streamlines obtained from the present method at $t = 1.6$ are compared with those calculated at the same time by means of a fractional-step projection method ($\boldsymbol{u} - p$ solution, see §Guermond and Quartapelle [7]). The contour lines of pressure at $t = 1.6$ provided by the present method are compared in Figure 9.3 with those of the solution calculated by means of the fractional-step projection method. Figures 9.4 and 9.5 contain the streamlines at times $t = 2.8$ and 3.6, respectively, obtained by the present method and always compared with those of the fractional-step projection method solution.

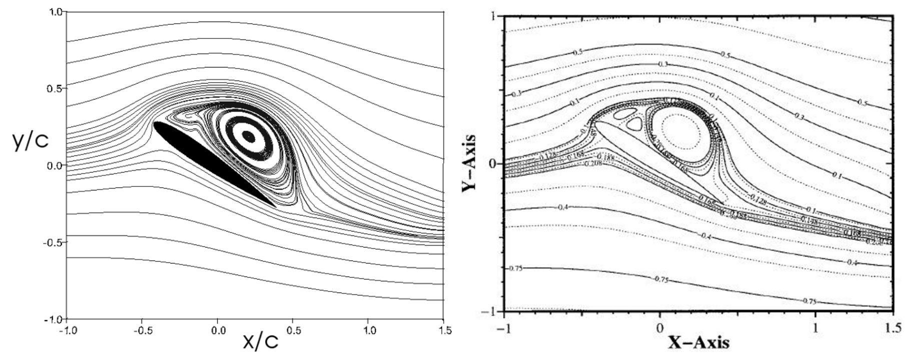
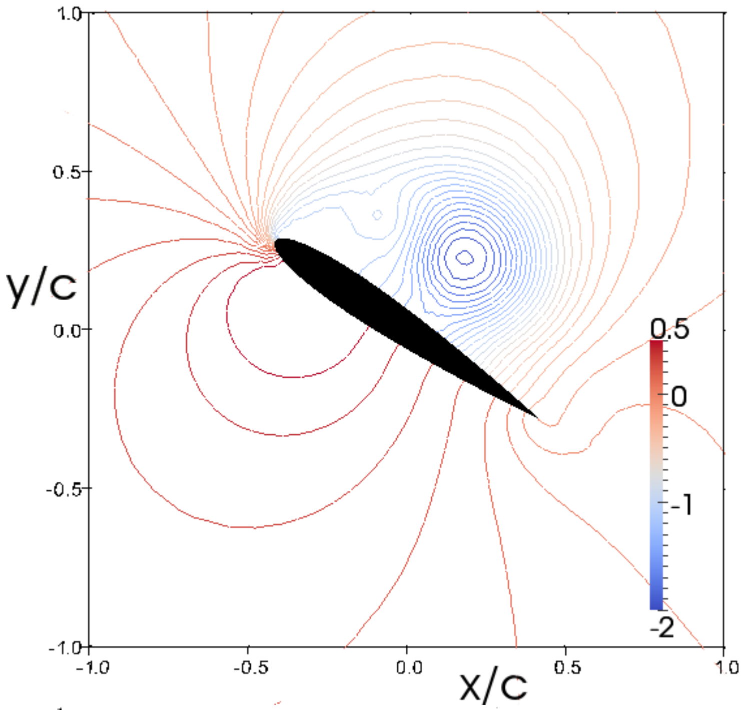


Figure 9.2: NACA 0012 airfoil at $\alpha = 34°$ and $Re = 1000$, solutions by G2 method (left) and by projection method (right) at $t = 1.6$, dimensionless quantities. Figure on the right taken from §Guermond and Quartapelle [7].



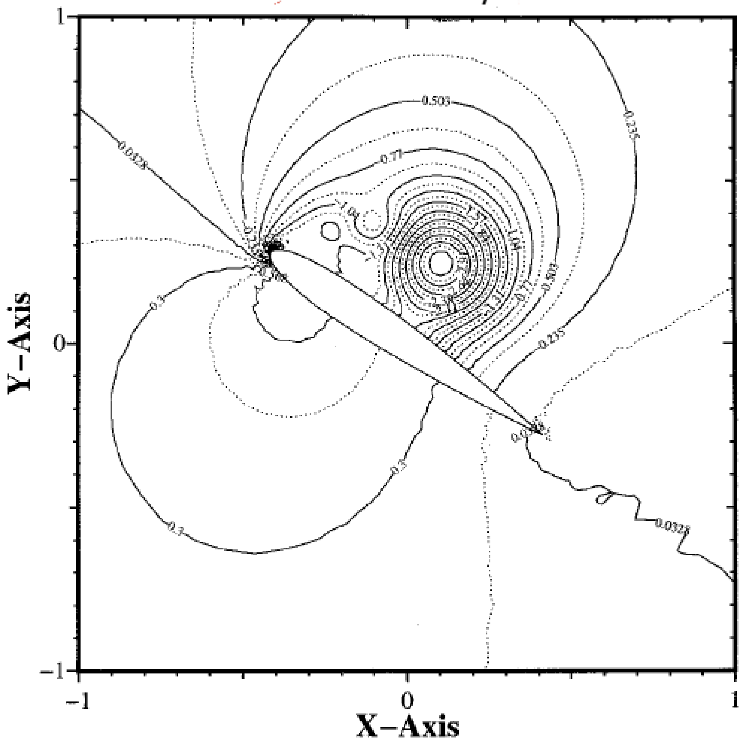


Figure 9.3: NACA 0012 airfoil at $\alpha = 34°$ and $Re = 1000$, pressure fields of the G2 method (up) and projection method solutions (down) at $t = 1.6$, dimensionless quantities. Figure on the bottom taken from §Guermond and Quartapelle [7].

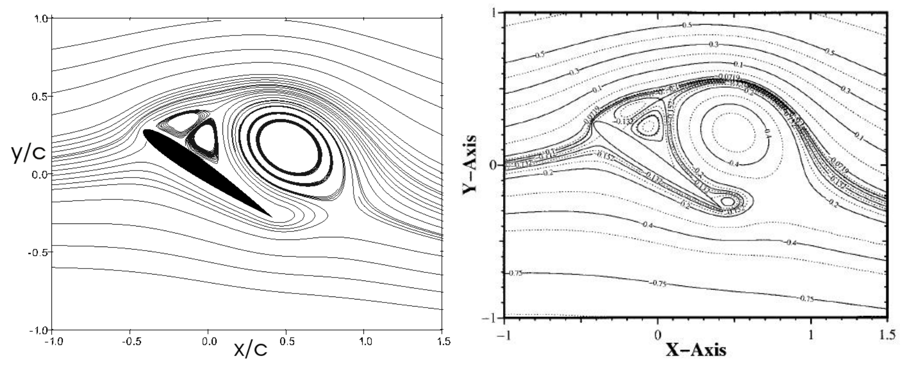


Figure 9.4: NACA 0012 airfoil at $\alpha = 34°$ and $Re = 1000$, solutions by G2 method (left) and by projection method (right) at $t = 2.6$, dimensionless quantities. Figure on the right taken from §Guermond and Quartapelle [7].

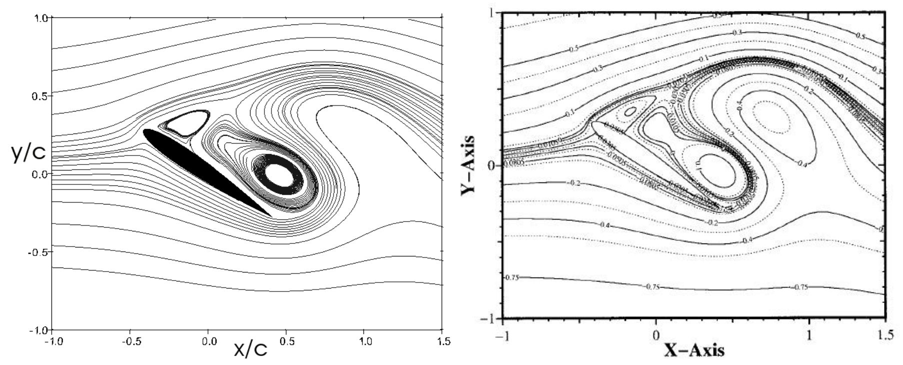


Figure 9.5: NACA 0012 airfoil at $\alpha = 34°$ and $Re = 1000$, solutions by G2 method (left) and by projection method (right) at $t = 3.6$, dimensionless quantities. Figure on the right taken from §Guermond and Quartapelle [7].

9.1 Adaptive G2

The adaptive algorithm is designed to compute the correct drag using a minimal number of degrees of freedom. Starting from the coarse mesh shown in Figure 9.1, the mesh has been adaptively refined according to the computed solution (with the goal of satisfying a given tolerance with respect to the error in the specified output of interest) and it finally consisted of about 9200 nodes and 17 600 cells. Since the adaptive algorithm is designed to minimize computational work for the computation of drag, the resulting computational mesh is optimized for the approximation of drag, as we can seen in Figure 9.6.

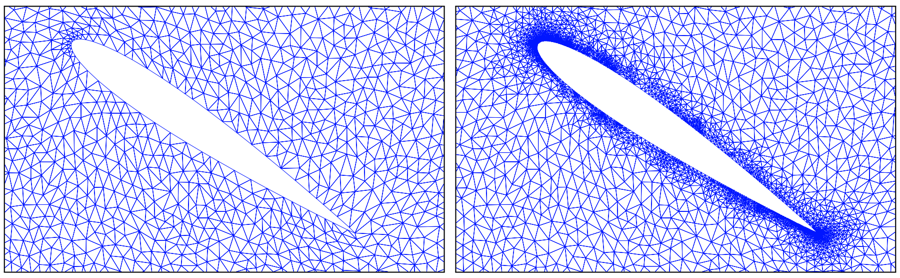


Figure 9.6: Zoom of computational mesh before (left) and after (right) adaptive mesh refinements with respect to drag.

Figures 9.7, 9.9 and 9.10 give the streamlines, at the three previously considered (dimensionless) times, obtained from the adaptive G2 method with respect to drag, compared with those of the solution calculated by §Guermond and Quartapelle [7]. Finally, the contour lines of pressure at $t = 1.6$ provided by the adaptive method are compared in Figure 9.8 with those of the solution calculated by means of the fractional-step projection method.

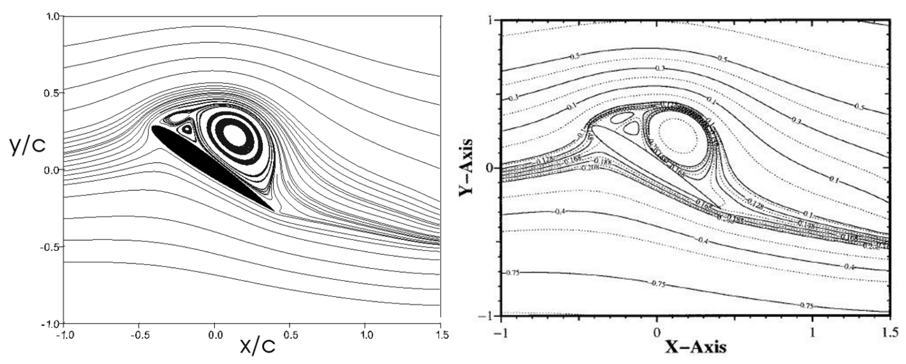


Figure 9.7: NACA 0012 airfoil at $\alpha = 34°$ and $Re = 1000$, solutions by adaptive G2 method (left) and by projection method (right) at $t = 1.6$, dimensionless quantities. Figure on the right taken from §Guermond and Quartapelle [7].

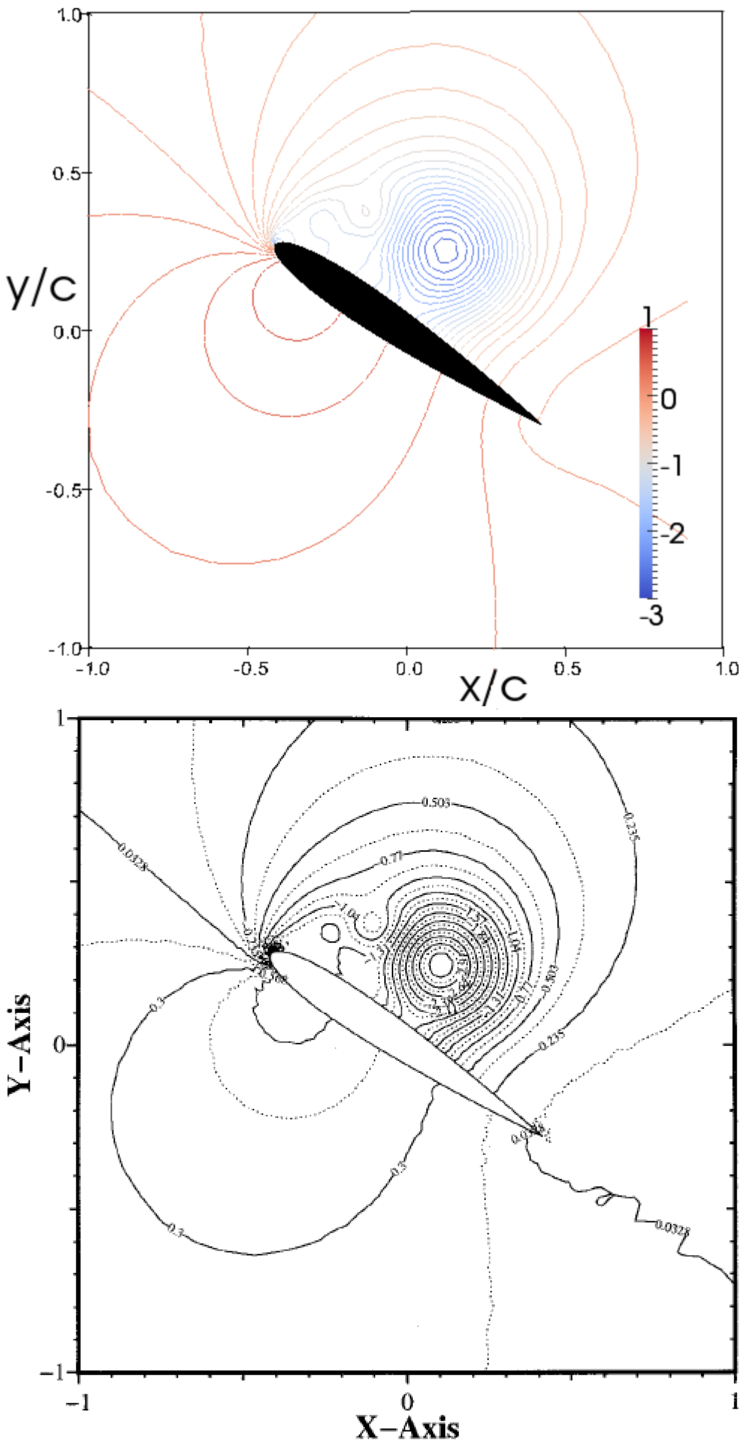


Figure 9.8: NACA 0012 airfoil at $\alpha = 34°$ and $Re = 1000$, pressure fields of the adaptive G2 method (up) and projection method solutions (down) at $t = 1.6$, dimensionless quantities. Figure on the bottom taken from §Guermond and Quartapelle [7].

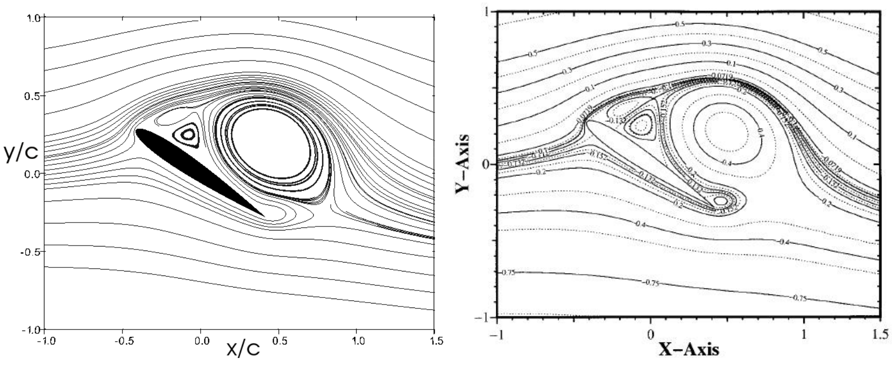


Figure 9.9: NACA 0012 airfoil at $\alpha = 34°$ and $Re = 1000$, solutions by adaptive G2 method (left) and by projection method (right) at $t = 2.6$, dimensionless quantities. Figure on the right taken from §Guermond and Quartapelle [7].

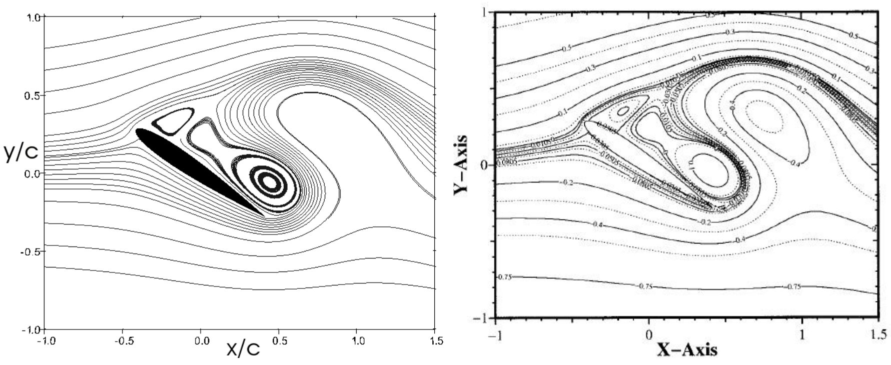


Figure 9.10: NACA 0012 airfoil at $\alpha = 34°$ and $Re = 1000$, solutions by adaptive G2 method (left) and by projection method (right) at $t = 3.6$, dimensionless quantities. Figure on the right taken from §Guermond and Quartapelle [7].

9.2 Conclusions

The comparison of the numerical results provided by the G2 method with reference solutions is quite satisfactory, especially for the adaptive version. The comparisons show that this method is capable of predicting the dynamics of the flow field quite correctly, even in the presence of sharp geometrical singularities of the boundary, like those at the trailing edges of the considered airfoil.

The method is found to be capable of predicting incompressible flows with re-circulations and separation accurately, without requiring any tuning of the algorithm.

The streamlines of the solution provided by the adaptive algorithm, which is displayed in Figure 9.7 show a re-circulatory region that is not visible in Figure 9.2, but which is present in the reference solution. Furthermore, the pressure fields displayed in Figure 9.3 and 9.8 are qualitatively similar, but that one provided by the adaptive method is more accurate with respect to that of the reference solution. These discrepancies can be explained, at least partly, by the lack of refinement of the mesh employed.

10 Plunging Airfoil at Low Reynolds Numbers

The aim of the present chapter is to examine the thrust, lift and propulsive efficiency of an inflexible 2-D airfoil oscillating in heave at “low” Reynolds numbers. The flow over a NACA 0012 airfoil, oscillated sinusoidally in plunge (Figure 10.1), is thus simulated numerically using a two dimensional adaptive Navier-Stokes solver at Reynolds number of 2×10^4 (see Chapter 3). The results are the compared with those proposed in §Heathcote and Gursul [8]; Heathcote et al. [9]; Young and Lai [26], where, in order to validate the force measurement system, a set of thrust and power-input measurements were carried out for a 100 mm chord, 400 mm span, NACA 0012 airfoil oscillating with constant amplitude ($h = 0.175$) between two end plates in a free-surface closed-loop water tunnel. Tests were carried out for Reynolds numbers of 10 000, 20 000 and 30 000, and for a frequency range of $0 < k_G < 7$.

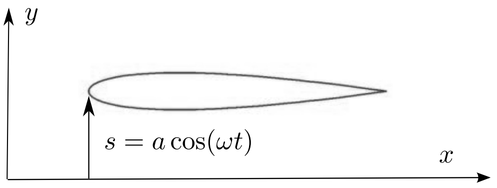


Figure 10.1: Schematic of the airfoil heaving periodically in the vertical direction.

The following appropriate parameters are introduced: Re, h, and k_{G}. The Reynolds number is based on the chord length of the airfoil. The dimensionless heave amplitude is $h = \frac{a_{\mathrm{LE}}}{c}$ and the Garrick frequency can be expressed as:

$$k_{\mathrm{G}} = \frac{\pi f c}{U} = \frac{\omega c}{2U} \quad \text{since} \quad \omega = 2\pi f.$$

The displacement of the leading-edge is given by $s = a\cos(\omega t)$.

The forces applied to the wing in the x and y directions, F_x and F_y, were measured in §Heathcote and Gursul [8]; Heathcote et al. [9]. The force F_x is equal to the drag (or thrust) on the wing. The force

F_y is equal to the lift on the wing, plus a contribution arising from the inertia of the wing. This contribution is proportional to the mass m of the wing and the wing acceleration in the y-direction ($\frac{dv}{dt}$), in which v is the instantaneous plunging velocity. Hence, total force measured is

$$F_y = L + m\frac{dv}{dt}$$

and the power input can be calculated as

$$Lv = \left(F_y - m\frac{dv}{dt}\right) v$$

On integrating with time over a complete heave cycle, the term $v\frac{dv}{dt}$ vanishes because the phases of v an $\frac{dv}{dt}$ differ by 90°. Hence the inertia of the airfoil does not contribute to the time-averaged power input. This approach of using F_y for the input power calculations was validated in §Heathcote et al. [9]. Thus, the period-averaged power input therefore equals the period-averaged value of $F_y v$, where v is the instantaneous velocity of the root.
Drive force and thrust force data were collected for a finite number of oscillations for each test condition. The thrust coefficient, C_T, is given by:

$$C_\mathrm{T} = \frac{T}{\frac{1}{2}\rho U^2 c}$$

where T is the thrust per unit span. The time-averaged thrust coefficient is found by averaging over a complete number of cycles. The time-averaged power input is given by:

$$\bar{C}_\mathrm{P} = \frac{\overline{F_y v}}{\frac{1}{2}\rho U^3 c}$$

where $F_y v$ is the instantaneous power input, and the overbar denotes an average over time.

The thrust coefficient and power-input coefficient data obtained by §Heathcote and Gursul [8]; Heathcote et al. [9] are plotted in Figure 10.2 and Figure 10.3, respectively, which also show, for comparison, the predictions of Garrick[1], a panel method and a viscous Navier–Stokes code

[1]For a plunging flat plate, Garrick, using linearized potential flow theory, determines the time-averaged thrust coefficient per unit span $C_{\mathrm{T}_\mathrm{mean}}$ as a function of the nondimensional plunge amplitude h and reduced frequency k_G (valid in the limit of small amplitude oscillations): $C_{\mathrm{T}_\mathrm{mean}} = 4\pi(k_\mathrm{G}h)^2(F^2 + G^2)$, where $F = F(k_\mathrm{G})$ and $G = G(k_\mathrm{G})$ are the real and imaginary components of the Theodorsen function $C(k_\mathrm{G}) = F(k_\mathrm{G}) + iG(k_\mathrm{G})$.

(the panel method and Navier Stokes predictions are of §Young and Lai [26], and the reader is referred to this source for details of the methods). Propulsive efficiency, i.d., the ratio of time-average thrust coefficient to the time-average power-input coefficient, is shown in Figure 10.4. Although the panel method correctly predicts a decrease in efficiency with increasing frequency, both inviscid methods are seen to significantly overestimate efficiency.

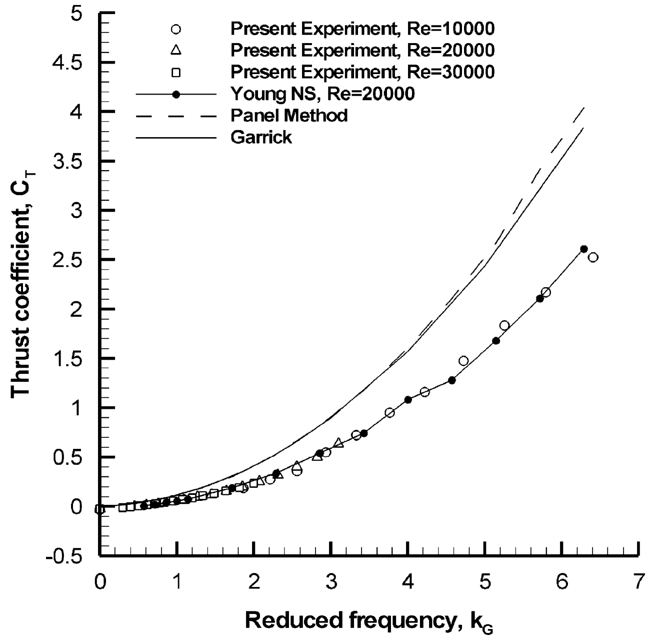


Figure 10.2: Comparison of the force validation experimental data (2-D NACA 0012 airfoil in pure heave) with linear theory, panel method and Navier Stokes solver. Thrust coefficient, taken from §Heathcote et al. [9]

It is seen from previous figures that both inviscid methods overestimate the experimentally measured thrust coefficient and that the experimental data shows the effect of Reynolds number to be very small for the range $10000 < Re < 30000$. Furthermore, Garrick theory and the panel method are seen to underestimate the power-input coefficient, due to the absence of a model of leading-edge vorticity shedding (in the Navier–Stokes model the leading-edge vortices were shown to augment the surface pressures, leading to a greater power-input requirement, see §Young and Lai [26]). Here again, the effect of Reynolds number is observed to be small for the experimental data.

In order to show the difference between the adaptive version and the non-adaptive version of the cG(1)cG(1) method implemented in this the-

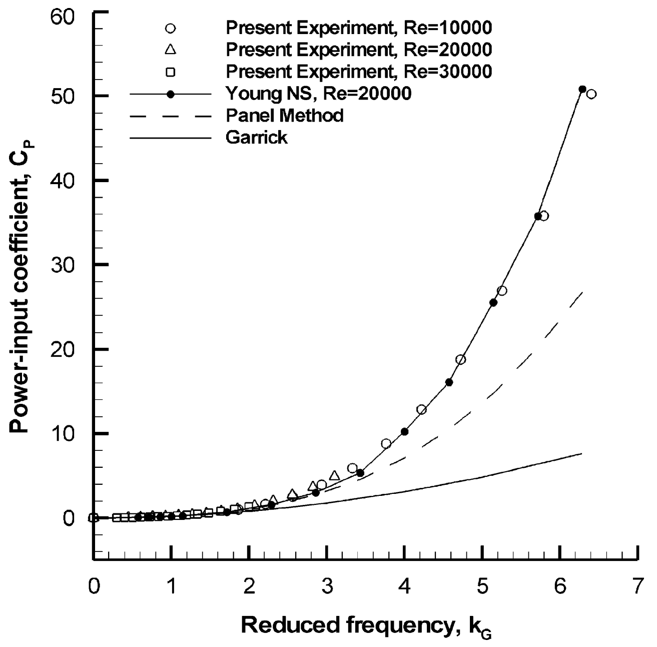


Figure 10.3: Comparison of the force validation experimental data (2-D NACA 0012 airfoil in pure heave) with linear theory, panel method and Navier Stokes solver. Power-input coefficient, taken from §Heathcote et al. [9]

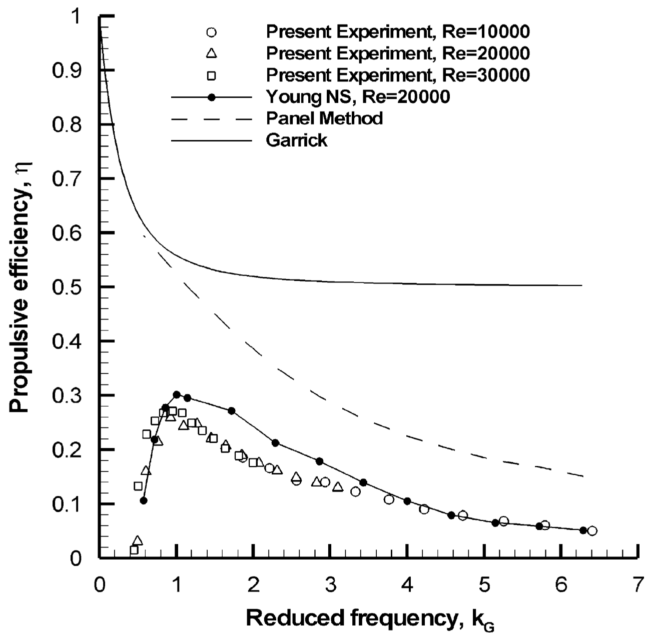


Figure 10.4: Comparison of the force validation experimental data (2-D NACA 0012 airfoil in pure heave) with linear theory, panel method and Navier Stokes solver. Propulsive efficiency, taken from §Heathcote et al. [9]

sis, we reported in Figure 10.7 the instantaneous thrust coefficient as a function of time over two heave cycles, calculated at the same Reynolds number and Garrick frequency, using the same fixed (dimensionless) time step for both methods. The adaptive algorithm is designed to compute the correct drag using a minimal number of degrees of freedom. Starting from a coarse mesh with ≈ 3800 nodes and 7500 cells (as computational domain we considered the rectangle given by the dimensions of the water tunnel test section), the mesh has been adaptively refined according to the computed solution until the error in the specified output, in this case drag, is less than a given tolerance, and it finally consisted of about 4200 nodes and 8200 cells. Since the adaptive algorithm is designed to minimize computational work for the computation of drag, the resulting computational mesh is optimized for the approximation of drag, as we can seen in Figure 10.5 and 10.6, which show some particulars of the computational meshes before and after adaptive mesh refinements. Unneccessary refinement is avoided in parts of the domain not critical for the approximation of drag. Instead, the mesh for solving the problem with the non-adaptive method consisted of about $10\,700$ nodes and $21\,200$ cells.

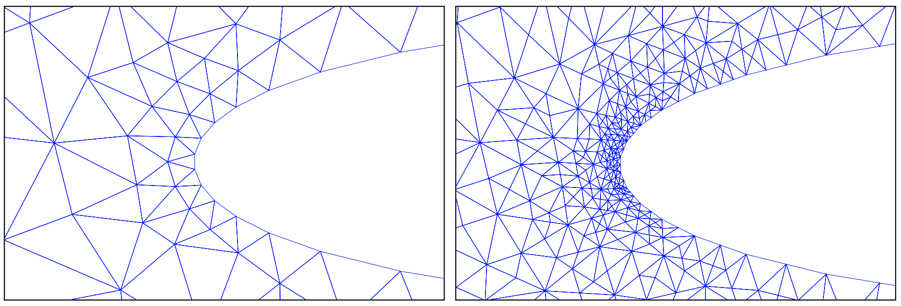


Figure 10.5: Coarse mesh (left) and refined mesh (right), leading edge particular; $Re = 20000$, $k_{\mathrm{G}} = 5$.

Two peaks in thrust are consistent with symmetry of the geometry, and the shedding of two vortices per cycle. We also note two main features of the adaptive method: a good approximation of the thrust coefficient is obtained using very few degrees of freedom, and the mesh is automatically constructed from a coarse mesh, thus bypassing the cost and challenge of ad hoc mesh design.

The adaptive algorithm is constructed for approximation of drag, but it may be interesting to measure also other output from the resulting solutions: for the frequency range of interest ($0 < k_{\mathrm{G}} < 7$), the time-averaged thrust coefficient, the time-averaged power-input coef-

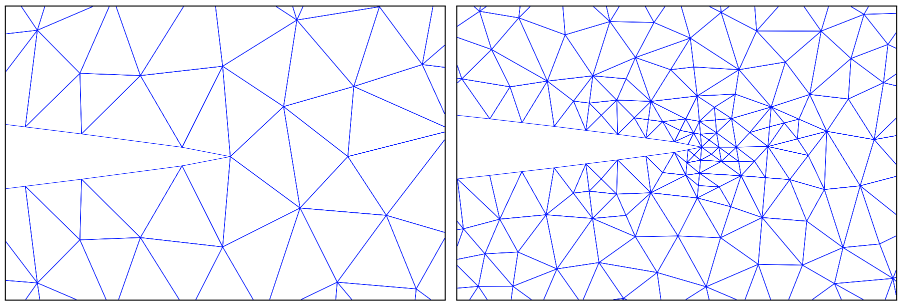


Figure 10.6: Coarse mesh (left) and refined mesh (right), trailing edge particular; $Re = 20000$, $k_G = 5$.

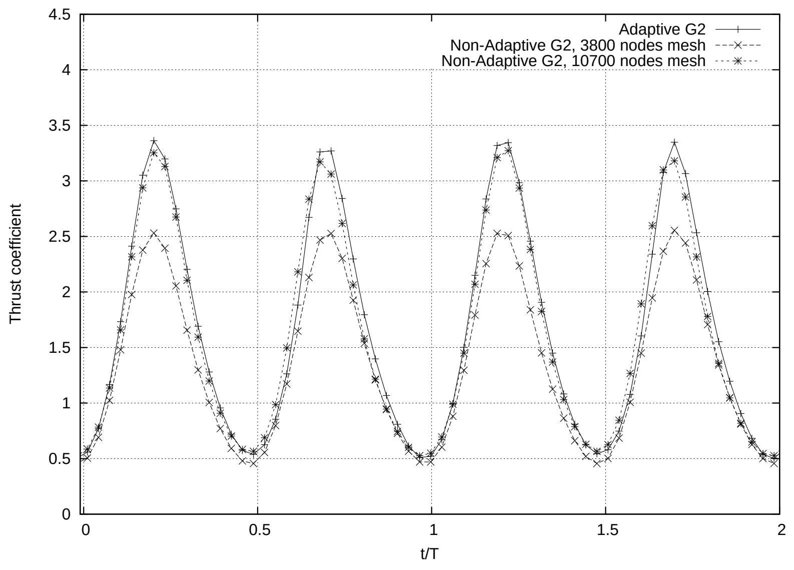


Figure 10.7: Instantaneous thrust coefficient as a function of time over two heave cycles; $Re = 20000$, $k_G = 5$.

ficient and the propulsive efficiency are calculated using the adaptive G2 method with respect to drag, and they are plotted in Figures 10.8, 10.9 and 10.10 respectively. Also reported for comparison are the data from §Heathcote et al. [9] (Figure 10.2).

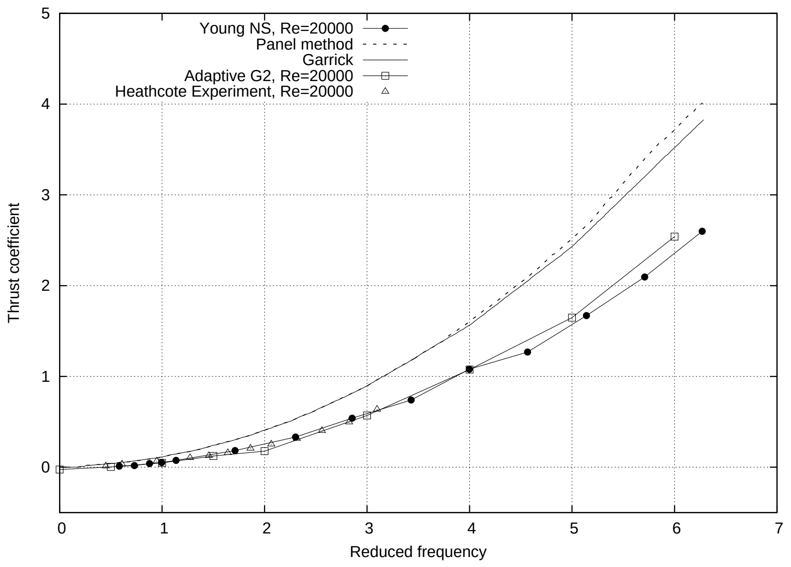


Figure 10.8: Comparison of the time-averaged thrust coefficient (2-D NACA 0012 airfoil in pure heave) with §Heathcote et al. [9]

10.1 Conclusion

It is seen from Figures 10.8, 10.9 and 10.10 that the Navier–Stokes predictions for thrust coefficient, power-input coefficient and propulsive efficiency, provided by G2 method, are in agreement with the experimental values in §Heathcote et al. [9] and with the predictions of §Young and Lai [26], over the complete frequency range. In particular, close agreement is found in the trend towards drag at low frequencies, the peak efficiency ($\eta \approx 30\%$), and the optimum frequency ($k_{\mathrm{G}} \approx 1$).

Even if the adaptive algorithm was designed to adaptively refine the mesh according to the computed solution until the error in the specified output, in this case drag, is less than a given tolerance, and so no attempt to refine the mesh with respect to lift has been made, also the power-input coefficient predictions (and consequently the propulsive efficiency) seem to be in agreement with both Young's predictions and Heathcote's experimental data. An adaptive algorithm with the goal of satisfying a given tolerance with respect to the error in two specified output, i.d., lift

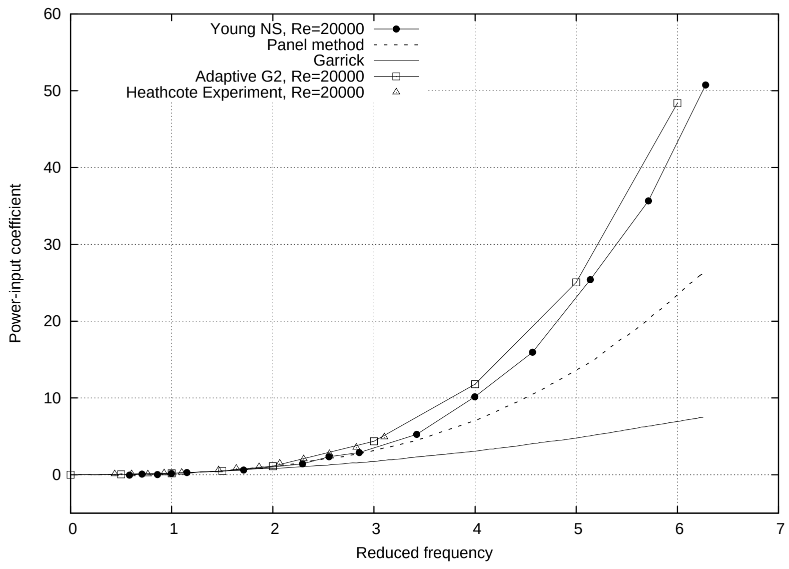


Figure 10.9: Comparison of the time-averaged power-input coefficient (2-D NACA 0012 airfoil in pure heave) with §Heathcote et al. [9]

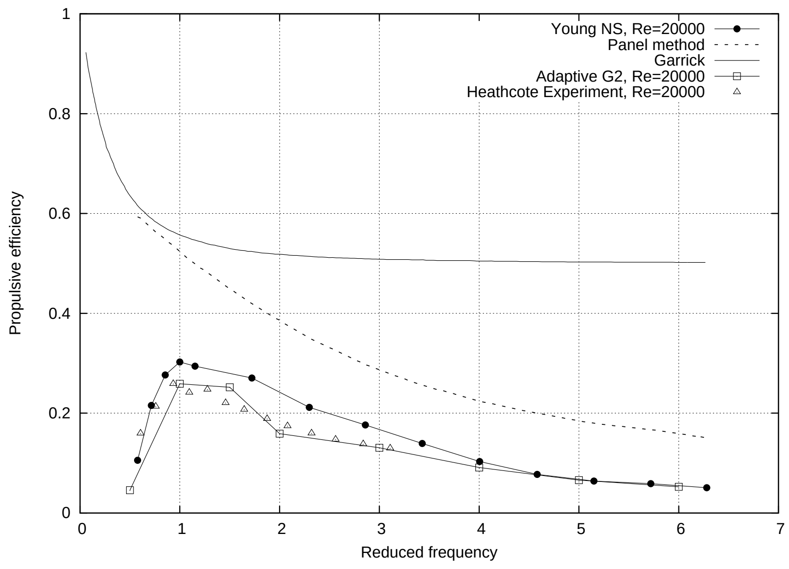


Figure 10.10: Comparison of the propulsive efficiency (2-D NACA 0012 airfoil in pure heave) with §Heathcote et al. [9]

and drag, was also implemented and tested: the mesh was so adaptively refine with respect to lift and drag. The results don't change significantly: this can be explained, at least partly, by the level of refinement of the mesh, which is optimized for the approximation of drag, that was enough to approximate lift.

11 Effect of Spanwise Flexibility on flapping wing propulsion

The purpose of this chapter, and that of the experimental study in §Heathcote et al. [9], is to measure the effect of spanwise flexibility on the thrust of a rectangular wing oscillated in heave at one end. The heave amplitude, $h = a_{\mathrm{root}}/c = 0.175$, is constant for all experiments. Three additional dimensionless parameters may be considered: the Reynolds number, Garrick frequency, and Strouhal number based on the amplitude of the mid-span ($z = b/2$):

$$Re = \frac{\rho U c}{\mu}, \quad k_G = \frac{\pi f c}{U}, \quad Sr = \frac{2 f a_{\mathrm{mid}}}{U}.$$

The displacement of the root was given by $s = a_{\mathrm{root}} \cos(\omega t)$. Three wings of $300\,\mathrm{mm}$ span, $100\,\mathrm{mm}$ chord, NACA 0012 cross-section, and rectangular platform were constructed for the experiment in §Heathcote et al. [9]. The first, termed *inflexible*, which was constructed from nylon ($E = 5\,\mathrm{GPa}$) in a rapid prototyping machine, was designed to be as stiff as possible: a hollowed structure and two $8\,\mathrm{mm}$ diameter steel rods ($E = 200\,\mathrm{GPa}$) spanning from root to tip ensure a high spanwise stiffness. The second, termed *flexible*, which was stiffened with $1\,\mathrm{mm}$ stainless steel sheet, was designed to be of intermediate flexibility. The third, termed *highly flexible*, which was stiffened with $1\,\mathrm{mm}$ aluminum sheet, was designed to be overly flexible. Each of the two flexible wings was constructed from polydimethylsiloxane rubber (PDMS, $E = 250\,\mathrm{kPa}$) cast in a NACA 0012 mould. Cross-sections of the three wings are shown in Figure 11.1. All wings were designed to be stiff in the chordwise direction.

In the present simulations, we substituted the *inflexible* wing with a rigid one (see Chapter 11.1), so fluid-structure interaction can't occur and the aerodynamic solver was not coupled with MBDyn; instead, for the two flexible wings, two MBDyn structural models were built (see Chapter 11.2).

In the experimental studies of §Heathcote et al. [9], the forces applied to the wing in the x and y directions, F_x and F_y, were measured. As shown

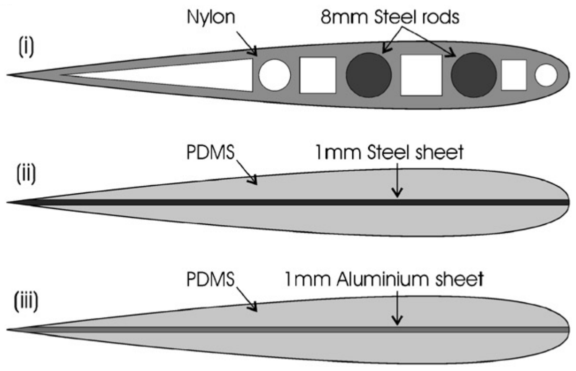


Figure 11.1: Cross-sections of the three NACA 0012 wings: (i) inflexible, (ii) flexible, (iii) highly flexible; taken from §Heathcote et al. [9].

in Chapter 10, the force F_x is equal to the drag (or thrust) on the wing and the force F_y is equal to the lift on the wing, plus a contribution arising from the inertia of the wing. This contribution is proportional to the wing acceleration, and therefore does not contribute to the time-averaged power-input. Thus, the period-averaged power input therefore equals the period-averaged value of $F_y v$, where v is the instantaneous velocity of the root. Drive force and thrust force data were collected for a finite number of oscillations for each test condition. The thrust coefficient, C_{T}, is given by:

$$C_{\mathrm{T}} = \frac{T}{\frac{1}{2}\rho U^2 c}$$

where T is the thrust per unit span. The time-averaged thrust coefficient is found by averaging over a complete number of cycles. The time-averaged power input is given by:

$$\bar{C}_{\mathrm{P}} = \frac{\overline{F_y v}}{\frac{1}{2}\rho U^3 c}$$

where $F_y v$ is the instantaneous power input, and the over-bar denotes an average over time.

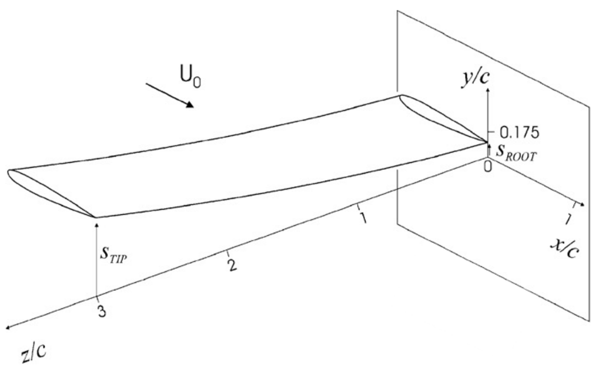


Figure 11.2: Schematic of the spanwise flexible wing heaving periodically, taken from §Heathcote et al. [9].

For the following simulations, the computational grid has been generated in Abaqus, and it consists of about 238 200 nodes and 1 212 200 tetrahedrons. The dimensions of the domain was given by the dimensions of the water tunnel test section adopted in §Heathcote et al. [9], see Figure 11.3.

Since in this phase we were mainly interested in developing a procedure to study coupled fluid-structure problems, we didn't utilize the adaptive G2 method for our computations, and utilized its non adaptive version, because, up to now, the developed adaptive algorithm for 3D problems takes a long time in the calculations.

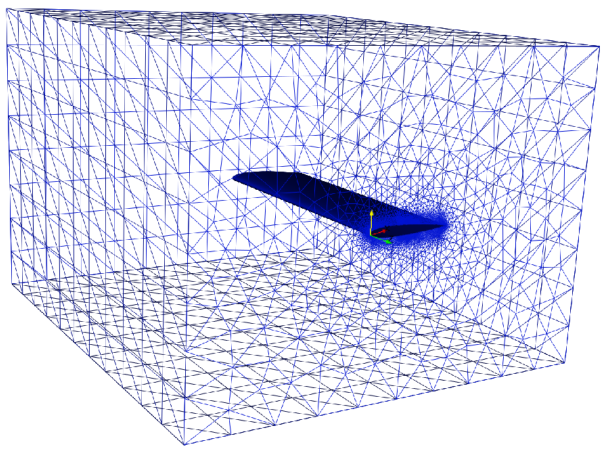


Figure 11.3: Computational grid. The dimensions of the domain are given by the water tunnel test section dimensions in §Heathcote et al. [9].

11.1 Rigid Wing Results

Rigid wing simulations were used to test the aerodynamic model prediction capabilities. In Figure 11.4 the values of time-averaged thrust coefficient provided by the G2 method are plotted as a function of Garrick frequency, for $Re = 30000$, together with the experimental data for the *inflexible* wing from §Heathcote et al. [9].

The values of the time-average thrust coefficient obtained by G2 method for the rigid wing are summarized in Table 11.1, in which they are also present the Heathcote's experimental data for the *inflexible* wing.

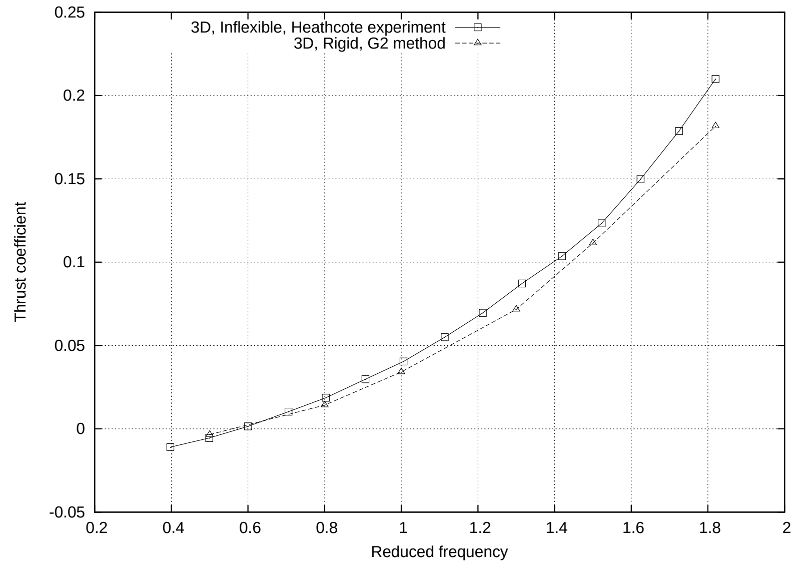


Figure 11.4: Time-average thrust coefficient of the rigid and the *inflexible* wing as a function of Garrick frequency, $Re = 30000$.

Table 11.1: Time-average thrust coefficient ($\bar{C}_T$) data of the rigid and the *inflexible* wing, $Re = 30000$.

Wing	$k_G = 0.5$	$k_G = 0.8$	$k_G = 1.0$	$k_G = 1.3$	$k_G = 1.5$	$k_G = 1.82$
rigid	-0.0035	0.014	0.034	0.072	0.1115	0.182
inflexible	-0.005	0.02	0.04	0.085	0.12	0.21

The comparison of the numerical results provided by G2 method with reference solutions is quite satisfactory: in particular, close agreement is found at low Garrick frequencies, while a small inaccuracy is observed towards high frequencies. These discrepancies can be explained, at least partly, by the fact that, although the *inflexible* wing in §Heathcote et al. [9] was designed to possess a high spanwise stiffness, it's cannot be considered rigid (see Figure 11.8). Instead, for our computation, a rigid wing was adopted. At the lowest frequencies the wings experience drag.

11.2 Flexible Wing Simulations

The MBDyn structural model, which was being correlated with experimental results from §Heathcote et al. [9], consists of:

- 96 structural nodes,
- 96 rigid bodies,
- 6 joints,
- 75 shells (four-node shell, see Chapter 5.2).

The materials investigated are shown in Table 11.2.

Table 11.2: Flapping wing: structural properties.

	Tensile modulus E	Poisson's modulus ν	Density ρ
steel	2.1×10^{11} Pa	0.3	7.8×10^{3} kg/m^3
aluminum	7.0×10^{10} Pa	0.3	2.7×10^{3} kg/m^3
pdms			9.65×10^{2} kg/m^3

`Body` elements are used to assign a mass to structural nodes. Two masses were referred to each structural node by means of the keyword `condense`: they are the fractions of the steel/aluminum sheet mass and the pdms mass that are relative to that node.
`Joint` elements connect structural nodes: they allow to arbitrarily constrain specific component of the absolute position and orientation of a node. The value of the constrained components of the absolute position and orientation was imposed by means of drives. The `drive` essentially represents a scalar function, whose value can change over time: in our case we imposed a cosine excitation of the type:

$$f(t) = a_{\text{root}}(1 - \cos(\omega t)), \quad \omega = 2\pi f.$$

We exploited the declaration and definition of some reference frames to place entities around in the model more efficiently, as shown in Figure 11.5. They are the "global" reference system (which has position $[0, 0, 0]$, orientation matrix $\mathbb{I}$ and angular velocity $[0, 0, 0]$), the "wing" reference system (which has position $[c/2, -b/2, 0]$, orientation matrix $\mathbb{I}$ and angular velocity $[0, 0, 0]$) and the "peer" reference frame (which has position $[0, -b/2, 0]$, orientation matrix $\mathbb{I}$ and angular velocity $[0, 0, 0]$). These

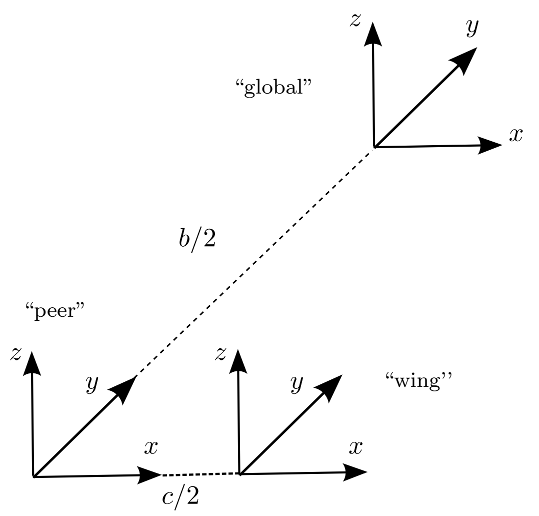


Figure 11.5: Reference systems. The global reference frame has position $[0, 0, 0]$, orientation matrix $\mathbb{I}$ and angular velocity $[0, 0, 0]$.

reference frames are used only during the input phase, where they help referring entities either absolute or relative to other entities. The output results, however, are expressed in the global frame.

The disposition of the MBDyn's nodes is shown in Figure 11.6: they are distributed with respect to the "wing" reference frame.

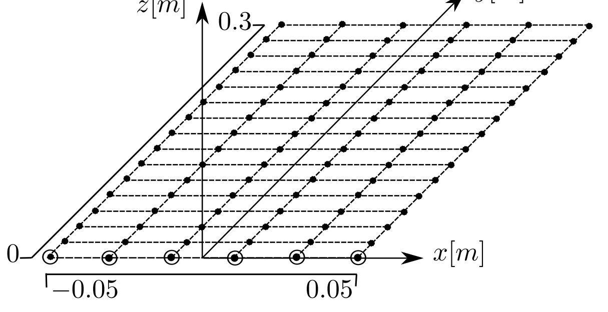


Figure 11.6: Disposition of MBDyn's structural nodes with respect to the "wing" reference frame. The [•] sign represents a structural node while the [◯] sign stands for a joint element.

As seen in Chapter 6.3, the fish-bone structure is obtained by adding two rigid arms at each node, in a direction perpendicular to the shell reference plane. The two arms have opposite sign to obtain a final layout which respects the symmetry. In this way all the possible rotation of each node are correctly transmitted to the aerodynamic wet surface. A set of points is thus generated by computing the kinematics of the points originating from the rigid-body motion of the structural nodes according to the specified offset (two offset blocks for each node). The positions of those points was written in a file: this is useful to generate the bulk data that is needed to compute the linear mapping matrix. After generating the computational grid, shown in Figure 11.3, we wrote in a file the coordinates of the peer's points (those lying on the interface surface, i.d., the internal boundary of the fluid mesh): since a reference node is defined[1], these points need to be expressed in a reference frame coincident with that of the reference node (the "peer" reference system).

As mentioned in Chapter 6.2, the mapping consists in a constant matrix that allows to compute the position and the velocity of the mapped points (subscript "peer") as functions of a set of points rigidly offset from MBDyn's nodes (subscript "mb"),

$$\boldsymbol{x}_{\text{peer}} = H\boldsymbol{x}_{\text{mb}}$$
$$\dot{\boldsymbol{x}}_{\text{peer}} = H\dot{\boldsymbol{x}}_{\text{mb}}$$

The same matrix is used to map back the forces onto MBDyn's nodes based on the preservation of the work done in the two domains,

$$\delta\boldsymbol{x}_{\text{mb}}^{\top}\boldsymbol{f}_{\text{mb}} = \delta\boldsymbol{x}_{\text{peer}}^{\top}\boldsymbol{f}_{\text{peer}} = \delta\boldsymbol{x}_{\text{mb}}^{\top}H^{\top}\boldsymbol{f}_{\text{peer}}$$

which implies

$$\boldsymbol{f}_{\text{mb}} = H^{\top}\boldsymbol{f}_{\text{peer}}$$

so the loads computed at the aerodynamic nodes can be transferred to the structural mesh. In Figure 11.7 are shown, as example, the set of points rigidly offset (in z direction) from MBDyn's nodes and the set of points used by the peer process for the present wing.

Using Octave we computed the matrix H and stored it in a file: when MBDyn is executed it loads the mapping matrix from this file (using sparse storage since for usual problems the matrix is significantly sparse). The interface method uses quadratic polynomial basis with C^2 weight function and a local support of 12 points (see Chapter 6.2).

In order to communicate with the external software that computes forces applied to the peer nodes and might depend on the kinematics of those nodes through a linear mapping, the `External structural`

[1]In this case the kinematics of the points is formulated in the reference frame of the reference node. The forces are expected in the same reference frame.

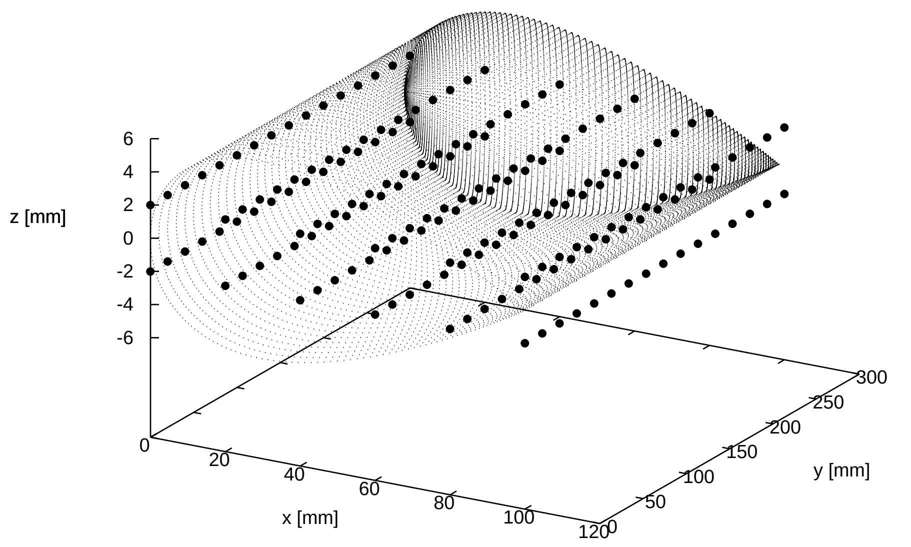


Figure 11.7: Example of set of points rigidly offset from MBDyn's nodes and those used by the peer process.

`mapping` element were used. As communication schemes we choose to adopt the socket communicator: by an optional parameter we gave instructions to MBDyn to create the socket, to which the peer have to connect to, and specified the type of coupling (we imposed a tight coupling forcing MBDyn to communicate each iteration, as a consequence, MBDyn solves the kinematics at the current time step, say k, at iteration j, using forces evaluated for the kinematics at the same time step, k, but at iteration $j - 1$). The communication pattern can be summarized as:

1. MBDyn sends the predicted kinematics for step k,

2. MBDyn receives a set of forces sent by the external peer each time the residual is assembled; those forces are computed based on the kinematics at iteration j,

3. as soon as, while reading the forces, MBDyn is informed that the external peer converged, it jumps to the next time step $k + 1$; otherwise, it continues iterating until convergence.

As a convergence criteria the fluid-dynamic code checks the relative change of the fluid-dynamics resultant between two iterations. A rought tolerance of 0.1 is adopted; it has been verified that this allows to dramatically reduce the computational time at the expense of a reduced loss of precision.

The forces acting on peer nodes belonging to the boundary of the body are calculated as shown in Chapter 7.2.4; they are then map back onto MBDyn's nodes using the transpose of matrix H.

11.2.1 Deformation - single case

The shape response of the *flexible* and *inflexible*/rigid wings for the single case of $Re = 30000$, $k_G = 1.82$ is represented in Figure 11.8. The tip displacements of the wings are plotted over a period of one cycle. The displacement of the wing tip may be considered to arise from the sum of the root displacement and the deformation of the wing. The curve

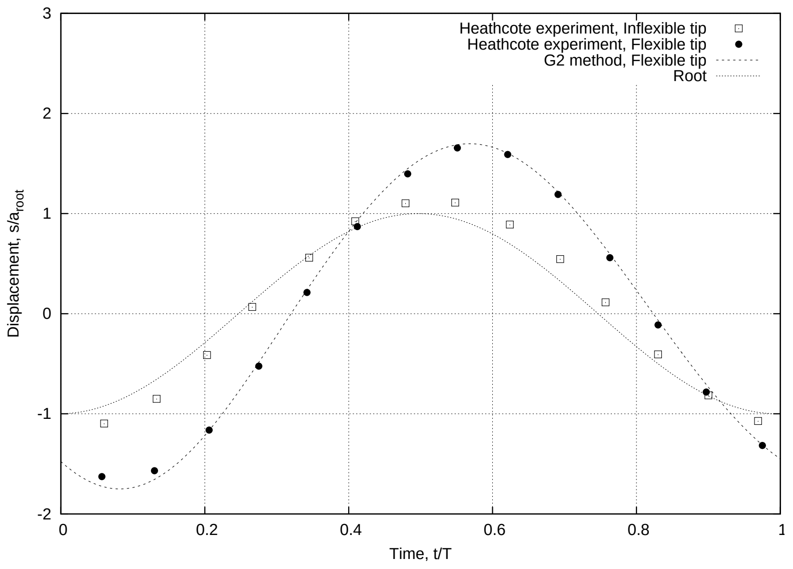


Figure 11.8: Tip displacements as a function of time; $Re = 30000$, $k_G = 1.82$.

labeled `'Root'` indicates the displacement of the root, and it coincides with the displacement of the rigid wing's tip.

11.2.2 Deformation - parametric study

The variation of tip amplitude response of the *flexible* wing with Garrick frequency, for a single Reynolds number, i.d., $Re = 30000$, is presented in Figure 11.9.

It is seen in Figure 11.9 that the normalized tip amplitude of both wings tends to unity as the frequency approaches zero. It is also seen that the tip amplitude of both wings increases with oscillation frequency over the whole frequency range.

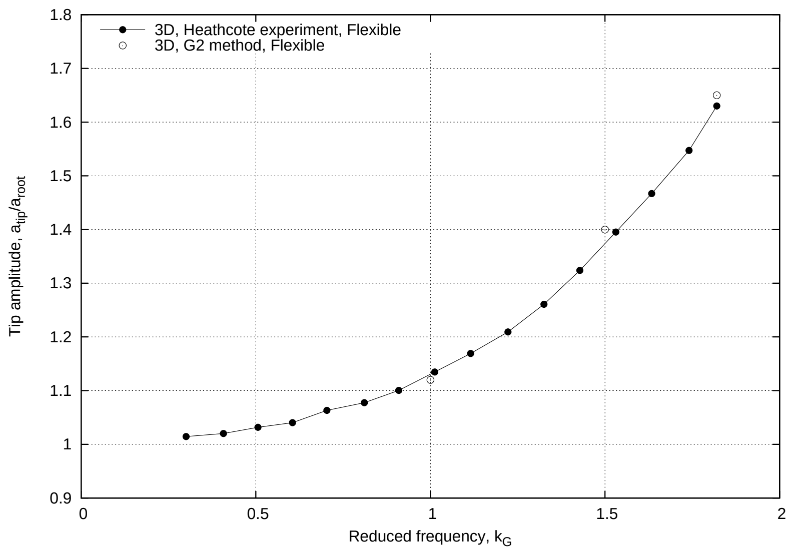


Figure 11.9: Tip amplitude as a function of Garrick frequency; $Re = 30000$.

11.2.3 Thrust force - single case

Instantaneous thrust coefficient curves for the case $Re = 30000$, $k_G = 1.82$, are shown in Figure 11.10. Two peaks in thrust are consistent with symmetry of the geometry, and the shedding of two vortices per cycle.

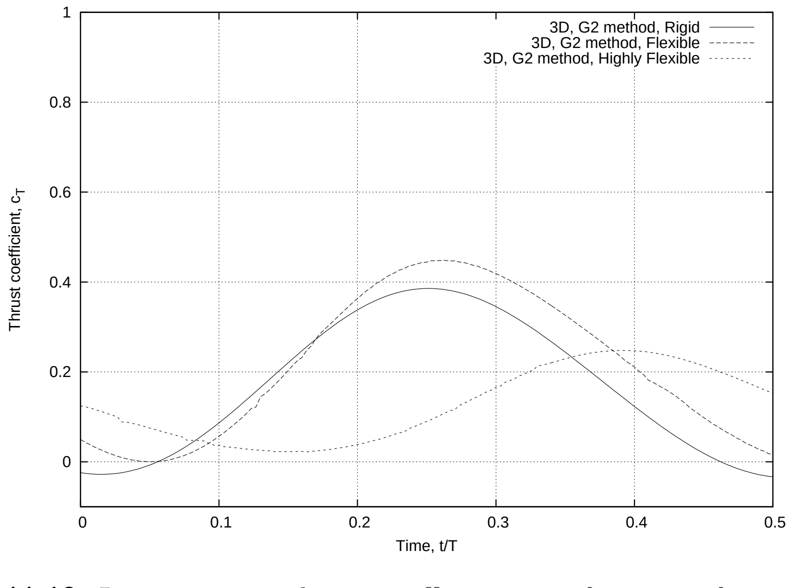


Figure 11.10: Instantaneous thrust coefficient as a function of time; $Re = 30000$, $k_G = 1.82$.

It is seen that the thrust coefficient of the *flexible* wing is greater than that of the *inflexible* wing, indicating that introducing a degree of spanwise flexibility increases the thrust coefficient for this Reynolds number and frequency. It is also seen that the thrust coefficient of the *highly flexible* wing is the lowest of the three. Another interesting feature is observed for the *highly flexible* wing: the instantaneous thrust coefficient is always positive. In summary, the flexibility of a wing is seen to affect the thrust characteristics of the wing.

The small oscillations of the thrust coefficient curves of the *flexible* and *highly flexible* wings, as seen in Figure 11.10, can be explained by the tolerance we choose to specify the coupling between MBDyn and the external aerodynamic solver.

11.2.4 Thrust force - parametric studies

In order to establish whether the flexible wing experiences greater thrust over a range of frequencies, a parametric study was carried out. The complete set of thrust coefficient data for $Re = 30000$ is plotted in Figure 11.11, where the experimental data from §Heathcote et al. [9] are also present for comparison. The single case discussed above corresponds to the highest frequency in Figure 11.11.

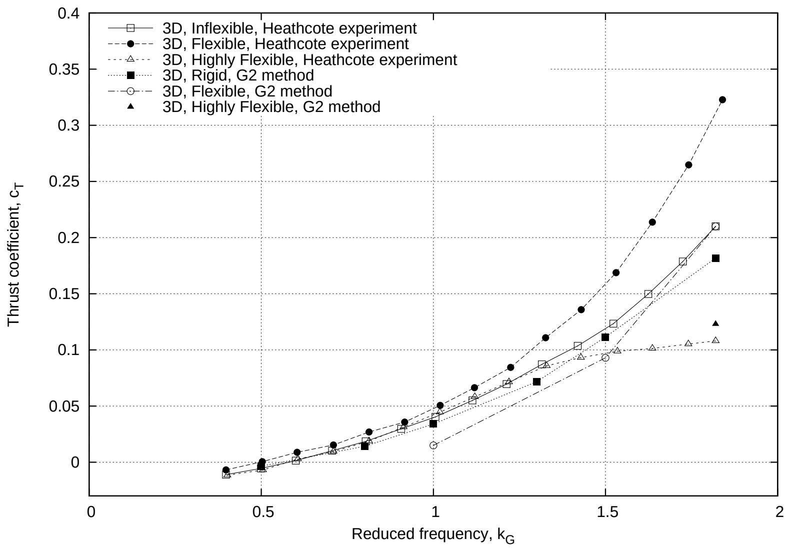


Figure 11.11: Thrust coefficient as a function of Garrick frequency, $Re = 30000$.

It is seen that the benefit in thrust for the *flexible* wing over the *inflexible* wing doesn't persist to lower frequencies, unlike the Heathcote's experimental data do. The discrepancy between the numerical values of the thrust coefficient provided by G2 method and experimental data can be probably explained by the somewhat coarse mesh for the Re at hand; since it is not wise to uniformly refine the mesh, it would be necessary to resort to the adaptive algorithm in order to selectively refine it. Unfortunately, the procedure for the error estimation for 3D problems takes a very long time, and this nullifies the benefit of a selectively mesh refinement.

12 Conclusion

In this thesis we have presented our work on adaptive finite element methods for incompressible turbulent fluid flow, and its implementation in FEniCS has been demonstrated. The incompressible Navier-Stokes equations are solved using a stabilized Galerkin finite element method. Both the stopping criterion and the mesh refinement strategy are based on "a posteriori" error estimates. There is no filtering of the equations, and thus no Reynolds stresses are introduced. Instead the stabilization in the numerical method is acting as a simple turbulence model.

This work also presented the employment of a nonlinear 4-node shell element within the free multibody solver MBDyn to support the modeling and the analysis of flexible wing oscillating in plunge. In order to provide realistic aerodynamic forces and high-fidelity fluid-structure coupling, the structural solver has been coupled with unsteady Navier-Stokes. An original general-purpose, meshless boundary interfacing approach based on Moving Least Squares with Radial Basis Functions has been used.

Using the G2 method, we were able to simulate the drag force of a non-oscillating NACA 0012 airfoil traveling in air. Also, we computed the drag and lift forces on an oscillating in heave NACA 0012 airfoil and investigated the effect of spanwise flexibility on flapping wings.

Encouraging preliminary results have been obtained from the modeling and analysis of the dynamics and aeroelasticity of a flexible oscillating wing model.

12.1 Future work

Future activity will address to:

- model β as a function of velocity and even roughness of the wing instead of a costant value (the β chosen here was to simplify the problem and we can compute the flow around a wing/airfoil in a more realistic situation by choosing β as an accurate function of velocity);

- extend the adaptive DNS/LES method to 3D fully coupled fluid-structure problems;
- design of a parallel implementation of the developed finite element method;
- add a flap hinge and/or a pitch hinge to the flexible wing models to achieve a more realistic insect-like flight.
- extensive parametric investigation of the flexible wing models to improve the understanding of the physics of the problem and support future designs.

Bibliography

[1] M. Alnæs, H. P. Langtangen, A. Logg, K.-A. Mardal, and O. Skavhaug. *UFC Specification and User Manual*, 2007. URL `http://fenicsproject.org`.

[2] M. S. Alnæs and A. Logg. *UFL Specification and User Manual*, 2009. URL `http://fenicsproject.org`.

[3] M. S. Alnæs and A. Logg. *UFL User Manual*, November 2010.

[4] M. S. Alnaes, A. Logg, Garth Wells, H. P. Langtangen, J. Hake, and R. C. Kirby. *FEniCS Manual*, October 2011.

[5] J. Donea and A. Huerta. *Finite Element Methods for Flow Problems.* Wiley, 2003.

[6] J. Donea, A. Huerta, J. P. Ponthot, and A. Rodrıguez-Ferran. Arbitrary Lagrangian–Eulerian methods. In E. Stein, R. de Borst, and T. J. R. Hughes, editors, *Fundametals*, volume 1 of *Encyclopedia of Computational Mechanics*, pages 1–25. Wiley & sons, 2004.

[7] J.L. Guermond and L. Quartapelle. Calculation of incompressible viscous flows by an unconditionally stable projection fem. *Journal of Computational Physics*, 132(1):12–33, 1997.

[8] S. Heathcote and I. Gursul. Flexible flapping airfoil propulsion at low reynolds numbers. *AIAA Journal*, 45:1066–1079, 2007.

[9] S. Heathcote, I. Gursul, and Z. Wang. Effect of spanwise flexibility on flapping wing propulsion. *Journal of Fluids and Structures*, 24: 183–199, 2008.

[10] Johan Hoffman. Computation of mean drag for bluff body problems using adaptive dns/les. *SIAM J. Sci. Comput.*, 27(1):184–207, 2005.

[11] Johan Hoffman. Computation of turbulent flow past bluff bodies using adaptive general Galerkin methods: drag crisis and turbulent Euler solutions. *Comput. Mech.*, 38:390–402, 2006.

[12] Johan Hoffman. Adaptive simulation of the sub-critical flow past a sphere. *J. Fluid Mech.*, 568:77–88, 2006.

[13] Johan Hoffman. Efficient computation of mean drag for the sub-critical flow past a circular cylinder using general Galerkin G2. *Int. J. Numer. Meth. Fluids*, 2009.

[14] Johan Hoffman and Claes Johnson. A new approach to computational turbulence modeling. *Comput. Methods Appl. Mech. Engrg.*, 195:2865–2880, 2006.

[15] Johan Hoffman and Claes Johnson. Stability of the dual navier-stokes equations and efficient computation of mean output in turbulent flow using adaptive DNS/LES. *Comput. Meth. Appl. Mech. Eng.*, 195:1709–1721, 2006.

[16] Johan Hoffman and Claes Johnson. *Computational Turbulent Incompressible Flow*, volume 4 of *Applied Mathematics: Body and Soul*. Springer, 2007. URL `http://dx.doi.org/10.1007/978-3-540-46533-1`.

[17] Volker John. Slip with friction and penetration with resistance boundary conditions for the navier stokes equations - numerical tests and aspects of the implementation. *Journal of Computational and Applied Mathematics*, 147:287–300, 2002.

[18] A. Logg and G. N. Wells. DOLFIN: Automated finite element computing. *ACM Transactions on Mathematical Software*, 32(2):1–28, 2010. URL `http://dx.doi.org/10.1145/1731022.1731030`.

[19] Anders Logg and Garth Wells. *DOLFIN User Manual*, November 2010.

[20] Anders Logg, Garth N. Wells, and Kent-Andre Mardal. *Automated Solution of Differential Equations by the Finite Element Method*, April 2011.

[21] Pierangelo Masarati. *MBDyn Input File Format*, November 2011.

[22] Giuseppe Quaranta. *A Study of Fluid-Structure Interactions for Rotorcraft Aeromechanics : Modeling and Analysis Methods*. PhD thesis, Politecnico di Milano, 2004.

[23] Giuseppe Quaranta, Marco Morandini, Pierangelo Masarati, and Riccardo Vescovini. Multibody analysis of a micro-aerial vehicle fapping wing. *Multibody Dynamics 2011, ECCOMAS Thematic Conference*, July 2011.

[24] M. E. Rognes and A. Logg. Automated goal-oriented error control I: Stationary variational problems. *SIAM Journal on Scientific Computing*, 2010.

[25] Jonathan Richard Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In Ming C. Lin and Dinesh Manocha, editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag, May 1996. URL `http://www.cs.cmu.edu/~quake/triangle.html`. From the First ACM Workshop on Applied Computational Geometry.

[26] J. Young and J. C. S. Lai. Oscillation frequency and amplitude effects on the wake of a plunging airfoil. *AIAA Journal*, 42(10): 2042–2052, October 2004.